Research on Web Robot Detection Technology for Concept Drift

Xue Chen¹, Yang Song², Wei Xiong³, Yutao Lu⁴ and Xingen Wang⁵*

¹ Chongqing Municipal Housing Provident Fund Administration Center, Chongqing, 401121, China
² Chongqing Municipal Housing Provident Fund Administration Center, Chongqing, 401121, China
³ Chongqing Municipal Housing Provident Fund Administration Center, Chongqing, 401121, China
⁴ Scotiabank, Mississauga, L5B0E9, Canada
⁵ Computer Science and Technology, Zhejiang University, Hangzhou, Zhejiang, 310012, China
*Corresponding author’s e-mail: newroot@zju.edu.cn

Abstract. Web robots have long been significant participants of the internet and the detection is deemed to be essential. Contemporary web robot detection techniques lack the ability to detect when the characteristics of web robots drastically change over time, namely a phenomenon called concept drift. Such change of web robot's behavior may lead to the deterioration of detection model performance. In order to maintain high detection performance over time, in this paper, we propose a novel web robot detection framework that consists of a model pool, a reinforcement learning algorithm integrating all the models and a concept drift detection module. First, we employ reinforcement learning to integrate a number of detection models that targets different types of web robots, by dynamically adjusting model weights. Then we apply the drift detection method (DDM) to monitor concept drift and identify the need to retrain the model over time. Experiments are conducted using real website datasets under concept drift. The results demonstrate that our model significantly outperforms the state-of-the-art approaches.

1. Introduction
Web robots, also known as crawlers, make up half of the world’s total web traffic, according to various studies. With the majority of web robots being malicious, bot detection is a crucial security priority for any business with Internet-based applications and services. As web robot detection approaches have become increasingly sophisticated over the past decades, bot developers also take on new technologies and deliberately design web bots to mimic human behaviour to bypass the detection process. Therefore, it is common to see concept drift occur where bot developers change the bots’ behaviour over time.

In predictive analytics and machine learning, concept drift is a phenomenon in which the statistical properties of the target or input variables change over time in unpredictable ways. Such change may negatively impact the prediction accuracy of the web robot detection system. Consequently, the research on the detection algorithm under the phenomenon of concept drift is an important means to ensure that the anti-crawler system can effectively fight against crawlers for a long time.
There has been a great deal of researches on web robot detection being conducted over the past years but few focuses on dealing with concept drift. In this paper, we study the abovementioned web robot detection problem under concept drift and propose a novel web robot detection framework that integrates several models and dynamically adjust model integration strategy based on reinforcement learning.

2. Related Work
In this section, we briefly review some related works on web robot detection algorithm. Web crawler detection algorithms can be divided into two categories, the method based on heuristic rules and the method based on machine learning.

The method based on heuristic rules designs rules to identify web crawlers according to the predefined characteristics and expert experience. Fan et al identified crawlers based on the trap technique[1]. Doran et al compared to human and network behaviour characteristics of crawler when access to network resources[2].

The research of crawler detection based on machine learning can be further divided into supervised method and unsupervised method. Supervised method is the first machine learning algorithm applied in the field of crawler detection, such as Bayesian classifiers, decision trees, support vector machines, association rule mining, or ensemble methods. In some recent studies, several researches show unsupervised techniques also reveal a high potential in this classification task. Mahdieh Zabihi et al proposed a density based clustering for web robot detection[3]. Hamidzadeh proposed a method based on fuzzy rough set theory to better characterize and cluster web visitors from three real websites[4].

3. Web Robot Detection Based on Web Log Data
Web robot detection based on web log data contains three steps: session identification, feature extraction and web robot detection.

Session identification is to split an access log of a website into proper sessions. Two consecutive HTTP requests that have the same IP addresses or same user agents will belongs to a same session if the time-lapse between them is within a pre-defined threshold. The majority of existing works of threshold[5] is 30 minutes, that means, the time interval between two consecutive requests is no more than 30 minutes in the same session. At the same time, if no new requests are added to a session for more than 30 minutes, the session is considered expired and removed from the system.

Feature extraction is for computing the statistic of each session which is used to capture differences between web robot and normal user. After obtaining the session feature vector, we utilize web robot detection model to mark session whether is web robot.

4. Web Robot Detection Framework
In this section, we present the framework proposed for web robot detection. Figure 1 shows the framework which consists of the model pool, the integration policies dynamical adjustment network and the concept drift detection module.
4.1. Model Pool
Establishing model pool includes two steps: training data segmentation and base model training. According to literature[6] and crawler design related forum content, crawler traffic is divided into several categories based crawler function. The crawler function refers to the goal or task of the designer when designing the crawler, such as making web page index, collecting data of specific topics, etc. According to the crawler function, this paper divides crawlers into 5 categories, and analyses the characteristics of each type of crawler.

Based on the simulation data, the random forest algorithm is used to evaluate the importance of candidate features, and a set of feature combinations for each type of crawler detection model is constructed for the training of base model. Candidate features are proposed in the previous literature[4]. The feature combinations are constructed for five types of crawler detection models, shown as table 1.

| Crawler Type       | Feature combination                                      |
|--------------------|----------------------------------------------------------|
| Index Crawler      | RequestsNumber, 304Ratio, SessionTime, DeviationInterval, NightRatio, Penalty, Width, Depth, AverageInterval |
| Topic Crawler      | CSR, SessionTime, Depth, DeviationInterval, MaxBrowserFileRate, Penalty, 304Ratio, Width, AverageInterval |
| Collection Crawler | ImageRadio, MaxBrowserFileRate, 304Ratio, SessionTime, DeviationInterval, Penalty, Width, Depth, CSR |
| Validation Crawler | UniqueType, SessionTime, DeviationInterval, MaxBrowserFileRate, Penalty, Width, Depth, RequestsNumber, 304Ratio |
| Copy Crawler       | HTMLRatio, UniqueType, SessionTime, DeviationInterval, RequestsNumber, Penalty, Width, Depth, MaxBrowserFileRate |

4.2. RL Formulation
Supposing $X = \{x_1, x_2, \cdots, x_k\}$ is the set of sessions in the website at $t$, where $k$ is the number of sessions. $Z = \{z_1, z_2, \cdots, z_n\}$ is the set of $n$ base detection models targeting different types of crawlers. For each session in the set $X$, $x_i$ will be encoded as a feature vector $m$ whose dimension is $m$. The purpose of weighting models is to build a system that can determine which models have more reliable detection results based on a collection of $X$ of existing sessions on the site. The following is the definition of reinforcement learning problems under multi-model weight allocation scenarios, including state, action, state transfer, reward function and strategy.

- **State**, $s_t \in S$: will correspond to the state of all sessions in the website, formulation as:

$$s_t = h\left(F^{1:k} = [f_{x_1}, \cdots, f_{x_k}]^T\right)$$

(1)

where $f_{x_i} \in \mathbb{R}^m$ is feature vector of session $i$, $h(\cdot)$ is feature embedding function. $W \in \mathbb{R}^{1 \times k}$ is weight matrix, where $k$ is the number of sessions in the website, column $i$ corresponds to the number of requests contained in session $i$, and feature embedding function is defined as:

$$s_t = h\left(F^{1:k} = [f_{x_1}, \cdots, f_{x_k}]^T\right) = \sigma(W \cdot F^{1:k} + B)$$

(2)

where, $B \in \mathbb{R}^{m \times 1}$ is a bias matrix, $\sigma$ is a nonlinear activation such as ReLU and ELU.

- **Actions**, $A_n^t$: corresponds to the action space that the agent can select at time $t$, $n$ is the number of base models. The action that the agent performs at the current moment $a_t \in A_n^t$ is a $n$-dimensional vector $[a_1^t, \cdots, a_n^t]$, value range of each dimension of vector $a_i^t$ is $[0, 1]$, corresponds to the weight value assigned by the agent to the $i$-th base model $z_i$.

- **State Transition**: $P(\cdot | s_t, a_t): S \times A_n^t \rightarrow S$ means the probability that the website state will change to $s_{t+1}$ after agent accepts the input state $s_t$ and selects the action $a_t$.

- **Reward Function**: $r(s_t, a_t): S \times A_n^t \rightarrow R$ will correspond to the reward that agent gains after selecting the action $a_t$ under the state $s_t$. In this paper, we define reward consists of two components: rationality of weight distribution and effectiveness of website traffic control. The rationality of weight allocation is used to measure whether the subject assigns higher weight to
the detection model that can correctly identify the crawler under the current website state, while
the model that can identify the error assigns the lowest weight as possible. We define \( TM \) as the
set of models for correctly detecting crawlers, \( FM \) as the set of models for not correctly detecting
crawlers, and reward function is defined as:

\[
r_1 = \sum_{z_i \in TM} a^i_t - \sum_{z_j \in FM} a^j_t
\]

The effectiveness of website traffic control is used to measure whether the subject can effectively
control the future website traffic level after taking action \( a \) at the time of \( t \), and it is quantified as the
decreased level of website traffic after selecting action at when the state \( s_t \) to change the state to \( s_{t+1} \).
(If traffic goes up, the reward is negative):

\[
r_2 = l(s_t, a_t) - l(s_{t+1}, a_t)
\]

where, \( l(\cdot) \) means website traffic statistics function, and at time \( t \), the total reward agent will gain is
the sum of \( r_1 \) and \( r_2 \):

\[
r(s_t, a_t) = \left( \sum_{z_i \in TM} a^i_t - \sum_{z_j \in FM} a^j_t \right) + \left( l(s_t, a_t) - l(s_{t+1}, a_t) \right)
\]

- Policy: will correspond to model weight allocation strategy which returns the probability that
  the subject performs the action \( a \) under the website status \( s_t \).

According to above description, solving the problem of model weight allocation turns into looking
for a strategy \( \pi(a|s) \) when the website state is \( s \) agent should select actions, so that the accumulated
maximum expected reward:

\[
\pi = \arg \max_{\pi(a|s)} \sum_{t=0}^{\infty} r(s_t, a_t)
\]

### 4.3. Integration policies dynamical adjustment network

Integration policies dynamical adjustment network based on DDPG to allocate weights for base models.
It contains four networks, which are the main Q-network, the target Q-network, the main policy network
and the target policy network. The main policy network selects the action \( a \) and executes it to obtain the
new status \( s' \) and reward \( r \). The target policy network puts the sample into the experience playback pool,
and uses the greedy method to select the action \( a' \) for the next state \( s' \) sampled in the experience playback pool.
The target Q-network calculates the target Q value for evaluating the value of the action based on the
\( s' \) provided by the experience playback pool and the action strategy target network. After the target
Q-network calculates part of the target Q value, the main Q-network will calculate the target Q value,
update the network parameters, and copy the network parameters to the target Q-network regularly. In
addition, the action policy main network will also update the network parameters based on the target Q
value calculated by the target Q-network, and periodically copy the network parameters to the action
policy target network. The loss function of the main Q-network is the mean square error:

\[
J(\omega) = \frac{1}{m} \sum_{j=1}^{m} (y_j - Q(s_j, a_j, \omega))^2
\]

The loss function of the main policy network:

\[
\nabla J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[ \nabla_a Q(s_i, a_i, \omega)|_{s=s_i, a=\pi_{\theta}(s)} \nabla_\theta \pi_{\theta}(s)|_{s=s_i} \right]
\]

Combining the weight and detection results obtained from each model to decide the session category.
\( P = \{p_1, p_2, \cdots, p_n\} \) is the prediction result set of \( n \) for the crawler detection model, and \( p_i \)
the token given to the session for the number \( I \) model (0 or 1, 0 is human, 1 is crawler). \( [a^1_t, \cdots, a^n_t] \)
means the weight of each model allocated by network at time \( t \), \( a^i_t \) corresponding to the number \( I \) model \( z^i_t \) to get
the weight value of detection model. The final session mark is determined by the weight and the greater
of the models with the same predicted results. Defining $y_1 = \sum_{i=1}^{n} p_i \cdot a_i$, and $y_0 = \sum_{i=1}^{n} a_i - y_1$. If $y_1$ is more than $y_0$, then session is marked as 1 which means coming from web robot, else is marked 0 which means coming from normal user.

4.4. Concept drift detection module
This module uses drift detection method (DDM) presented in the literature to detect concept drift. DDM sets two thresholds for the error rate, one is the warning threshold and the other is the drift threshold. If the input data, the first error rate has reached the warning threshold, shows data set probability distribution has the precursor to change that, if the input data of follow-up did not reduce error rate, and when the first data entry error rate reached the drift threshold, then you can judge the probability distribution of the data has changed, in order to adapt to the new sample data, the model need to study new data after that; If subsequent input reduces the error rate, it proves that there was a previous false positive. The magnitude of warning threshold and drift threshold need to be determined by the probability distribution of error rate.

5. Experiments
In this section, we use the access log data of a real website to evaluate our proposed approach. First, we split dataset to three data in chronological order: data1, data2, data3. Requests in data1 are earlier than data2, and in data2 earlier than data3. We use the training set segmented from data1 to train the model, and conduct experiments on the test sets segmented from data1, data2, and data3 to compare the performance of the detection model on the data sets in different time periods. The horizontal axis "time" represents the time slices arranged in sequence in the corresponding data set. It can be seen from the figure 2 that the overall performance of the detection model decreases obviously when data2 and data3 are collected without updating the model. It is proved that concept drift does exist in the field of crawler detection, and it will lead to the performance degradation of the model.

![Performance Degradation](image)

Figure 2. Performance Degradation.

We compare our proposed method with some existing state-of-the-art methods to verify the effectiveness.
- DBC_WRD[3]: This method adopts DBSCAN algorithm to cluster and proposes two new features, Penalty and 304Ratio.
- RO_WRD[5]: This method extracts the request resource type sequence from the session, and calculates the probability of the session from human and crawler respectively by using the discrete time Markov chain model.
- FRS_WRD[4]: This method adopts FRS (fuzzy rough set theory) to dynamically select features, and adopts SMO algorithm to classify crawlers and normal users.
**SC_WRD[7]**: This method uses a deep neural network combined with a Wald order probability ratio test to express the relationship between subsequent HTTP requests in an ongoing session to assess the likelihood that each session will be generated by either a robot or a human before it ends.

Table 2 shows the overall performance evaluation results of our approach and all baselines. We choose the evaluation criteria precision, recall and F1-score.

|                | precision | recall | F1-score |
|----------------|-----------|--------|----------|
| **DBC_WRD**    | 0.8846    | 0.9104 | 0.8973   |
| **RO_WRD**     | 0.7143    | 0.6923 | 0.7031   |
| **FRS_WRD**    | 0.9128    | 0.9285 | 0.9206   |
| **SC_WRD**     | 0.9214    | 0.9265 | 0.9239   |
| **Ens_Model**  | 0.9394    | 0.9167 | 0.9279   |

Shown as table 2, the performance of the integrated model proposed in this paper is outperform than excellent crawler detection models in dataset which exists concept drift. This experiment is set to simulate the real environment of the website. The test set was divided into time slices and put into the model in batches. The results of all time slices are averaged. Due to the small number of sessions contained in a partial time slice, both precision and recall of the calculation are very low in case of misjudgement, which will drag down the overall data.

### 6. Conclusion

In this paper, we propose a novel framework for web robot detection based on reinforcement learning and several models. Specifically, we propose an integration policies dynamical adjustment network to allocate weight for base models. The extensive experiments on dataset of a real website demonstrate the effectiveness of our method.

### Acknowledgments

This work is supported by the National Key R&D Program of China under Grant No. 2019YFB1600700.

### References

[1] Fan, C.L., Yuan, B., Yu, Z.H. (2010) Spider detection based on trap techniques. Journal of Computer Applications, 30(07): 1782-1784.

[2] Doran, D., Morillo, K., Gokhale, S.S. (2013) A comparison of web robot and human requests. In: Advances in Social Network Analysis and Mining. Niagara Falls. pp. 1374–1380.

[3] Zabihi, M., Jahan, M.V., Hamidzadeh, J. (2014) A density based clustering approach to distinguish between web robot and human requests to a web server. The ISC International Journal of Information Security, 6: 77-89.

[4] Hamidzadeh, J., Zabihimayvan, M., Sadeghi, R. (2018) Detection of web site visitors based on fuzzy rough sets. Soft Computing, 22: 2175–2188.

[5] Doran, D., Gokhale, S.S. (2016) An integrated method for real time and offline web robot detection. Expert Systems, 33(06): 592–606.

[6] Doran, D., Gokhale, S.S. (2012) A classification framework for web robots. Journal of the Association for Information Science and Technology, 63(12): 2549–2554.

[7] Cabri, A., Suchacka, G., Rovetta, S., Masulli, F. (2018) Online web bot detection using a sequential classification approach. In: HPCC/SmartCity/DSS. Exeter. pp. 1536–1540.