Knowledge and Information Systems (2023) 65:1935–1962
https://doi.org/10.1007/s10115-022-01812-3

REGULAR PAPER

Interactive reinforced feature selection with traverse strategy

Kunpeng Liu[1] · Dongjie Wang[2] · Wan Du[3] · Dapeng Oliver Wu[4] · Yanjie Fu[2]

Received: 30 December 2021 / Revised: 7 December 2022 / Accepted: 12 December 2022 /
Published online: 21 January 2023

Abstract

In this paper, we propose a single-agent Monte Carlo-based reinforced feature selection method, as well as two efficiency improvement strategies, i.e., early stopping strategy and reward-level interactive strategy. Feature selection is one of the most important technologies in data prepossessing, aiming to find the optimal feature subset for a given downstream machine learning task. Enormous research has been done to improve its effectiveness and efficiency. Recently, the multi-agent reinforced feature selection (MARFS) has achieved great success in improving the performance of feature selection. However, MARFS suffers from the heavy burden of computational cost, which greatly limits its application in real-world scenarios. In this paper, we propose an efficient reinforcement feature selection method, which uses one agent to traverse the whole feature set and decides to select or not select each feature one by one. Specifically, we first develop one behavior policy and use it to traverse the feature set and generate training data. And then, we evaluate the target policy based on the training data and improve the target policy by Bellman equation. Besides, we conduct the importance sampling in an incremental way and propose an early stopping strategy to improve the training efficiency by the removal of skew data. In the early stopping strategy, the behavior policy stops traversing with a probability inversely proportional to the importance sampling weight. In addition, we propose a reward-level and training-level interactive strategy to improve the training efficiency via external advice. What's more, we propose an

✉ Yanjie Fu
yanjie.fu@ucf.edu

Kunpeng Liu
kunpeng@pdx.edu

Dongjie Wang
wangdongjie@knights.ucf.edu

Wan Du
wdu3@ucmerced.edu

Dapeng Oliver Wu
dapengwu@cityu.edu.hk

1 Portland State University, 1825 SW Broadway, Portland, OR 97201, USA

2 University of Central Florida, 4000 Central Florida Blvd, Orlando, FL 32816, USA

3 University of California, Merced, 5200 North Lake Rd, Merced, CA 95343, USA

4 City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong

incremental descriptive statistics method to represent the state with low computational cost. Finally, we design extensive experiments on real-world data to demonstrate the superiority of the proposed method.

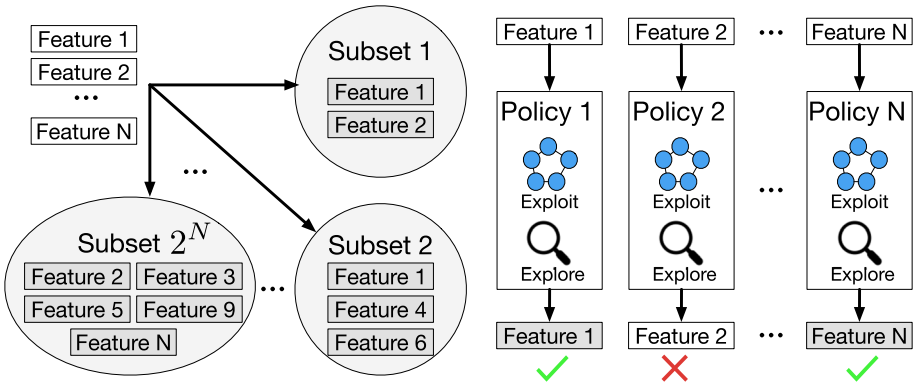
Keywords Feature selection · Reinforcement learning · Monte Carlo

1 Introduction

In general data mining and machine learning pipelines, before proceeding with machine learning tasks, people need to preprocess the data first. Preprocessing technologies include data cleaning, data transformation and feature engineering. As one of the most important feature engineering techniques, feature selection aims to select the optimal feature subset from the original feature set for the downstream task. Traditional feature selection methods can be categorized into three families: (i) filter methods, in which features are ranked by a specific score (e.g., univariate feature selection [1, 2], correlation-based feature selection [3, 4]); (ii) wrapper methods, in which optimal feature subset is identified by a search strategy that collaborates with predictive tasks (e.g., evolutionary algorithms [5, 6], branch and bound algorithms [7, 8]); (iii) embedded methods, in which feature selection is part of the optimization objective of predictive tasks (e.g., LASSO [9], decision tree [10]). However, these studies have shown not just strengths but also some limitations. For example, filter methods ignore the feature dependencies and interactions between feature selection and predictors. Wrapper methods have to directly search a very large feature space of 2^N feature subspace candidates, where N is the number of features. Embedded methods are subject to the strong structured assumptions of predictive models, i.e., in LASSO, the nonzero weighted features are considered to be important.

Recently, reinforcement learning has been incorporated with feature selection and produces an emerging feature selection method, called reinforced feature selection [11, 12]. In the reinforced feature selection, there are multiple agents to control the selection of features, one agent for one feature. All agents cooperate to generate the optimal feature set. It has been proved to be superior to traditional feature selection methods due to its powerful global search ability. However, each agent adapts a neural network as its policy. Since the agent number equals the feature number (N agents for N features), when the feature set is extremely large, we need to train a large number of neural networks, which is computationally high and not applicable for large-scale datasets. Our research question is as follows: Can we design a more practical and efficient method to address the feature selection problem while preserving the effectiveness of reinforced feature selection? To answer this questions, there are three challenges.

The first challenge is to reformulate the feature selection problem with smaller number of agents. Intuitively, we can define the action of the agent as the selected feature subset. For a given feature set, we input it to the agent's policy and the agent can directly output the optimal subset. However, the feature subset space is as large as 2^N, where N is the feature number (Fig. 1). When the dataset is large, the action space is too large for the agent to explore directly. To tackle this problem, we design a traverse strategy, where one single agent visits each feature one by one to decide its selection (to select or deselect). After traversing all the feature set, we can obtain the selected feature subset. We adapt the off-policy Monte Carlo method to our framework. In the implementation, we design two policies, i.e., one behavior policy and one target policy. The behavior policy is to generate the training data



(a) Feature subspace candidates (b) Reinforced feature selection

Fig. 1 Reinforced feature selection explores the feature subspace by assigning each feature one agent, and the agent's policy decides the selection of its corresponding feature

and the target policy is to generate the final feature subset. In each training iteration, we use the behavior policy to traverse the feature set and generate one training episode. The training episode consists of a series of training samples, each of which contains the state, the action and the reward. Similar with [11], we regard the selected feature subset as the environment, and its representation as the state. The action 1/0 denotes selection/deselection, and the reward is composed of predictive accuracy, feature subset relevance and feature subset redundancy. Using the training episode, we evaluate the target policy by calculating its Q value with importance sampling and improve it by the Bellman equation. After more and more iterations, the target policy becomes better and better. After the training is done, we use the target policy to traverse the feature set and can derive the optimal feature subset. Besides, the behavior policy is supposed to cover the target policy as much as possible so as to generate more high-quality training data and should introduce randomness to enable exploration [13]. We design an ϵ-greedy behavior policy, to better balance the coverage and the diversity.

The second challenge is to improve the training efficiency of the proposed traverse strategy. In this paper, we improve the efficiency from two aspects. One improvement is to conduct the importance sampling in an incremental way, which saves repeated calculations between samples. In the off-policy Monte Carlo method, since the reward comes from the behavior policy, when we use it to evaluate the target policy, we need to multiply it by an importance sampling weight. We decompose the sampling weight into an incremental format, where the calculation of the sampling weight can directly use the result of previous calculations. The other improvement is to propose an early stopping criteria to assure the quality of training samples as well as stopping the meaningless traverse by behavior policy. In Monte Carlo method, if the behavior policy is too far away from the target policy, the samples from the behavior policy are considered harmful to the evaluation of the target policy. As the traverse method is continuous and the importance sampling weight calculation depends on the previous result, once the sample at time t is skewed, the following samples are skew. We propose a stopping criteria based on the importance sampling weight and recalculate a more appropriate weight to make the samples from the behavior policy more close to the target policy. The dynamic-graph-based GCN in [11] has been proved effective in representing the state when the selected features are dynamically changing. However, this representation

Table 1 Commonly used notations

Notations	Definition
s_t, a_t	State at time t and action at time t
s^i, a^j	The i-th state and the j-th action
$\mathcal{S}$	State space defined as $\{s^i \vert i < inf\}$
$\mathcal{A}$	Action space defined as $\{a^j \vert j \in [1, N]\}$
γ	Discount factor in range [0,1]
$\mathcal{P}(s_t, a_t, s_{t+1})$	Transition probability
$\pi(s)/\pi^*(s)$	Policy/optimal policy
$\mathcal{M}$	Markov decision process (MDP) defined as $\{\mathcal{S}, \mathcal{A}, \mathcal{R}, \gamma, \mathcal{P}\}$
$\mathcal{U}$	Utility function from advisor's perspective
$\mathcal{F}$	Feature space (set) defined as $\{\mathbf{f}^k \vert k \in [0, M]\}$

method is based on deep neural network, and it requires a lot of computational resources to train. In this paper, we propose an incremental descriptive statistics method to reduce the computational burden, as well as capture the feature-feature relationship.

The third challenge is how to improve the training efficiency by external advice. In classic interactive reinforcement learning, the only source of reward is from the environment, and the advisor does have access to the reward function. However, in many cases, the advisor can not give direct advice on action, but can evaluate the state-action pair. In this paper, we define a utility function $\mathcal{U}$ which can evaluate state-action pair and provide feedback to the agent just like the environment reward does. When integrating the advisor utility $\mathcal{U}$ with the environment reward $\mathcal{R}$ to a more guiding reward $\mathcal{R}'$, we should not change the optimal policy, namely the optimal policy guided by $\mathcal{R}'$ should be identical to the optimal policy guided by $\mathcal{R}$. In this paper, we propose a state-based reward integration strategy, which leads to a more inspiring integrated reward as well as preserving the optimal policy. In addition, we propose a training-level interactive strategy. We design a comprehensive importance function to evaluate the training data. Experience replay is a widely used technology which can remember and reuse transition data from the past experience. In the original version of experience replay, all the data in the replay memory are randomly sampled without considering their importance and contribution to the policy training. To tackle this problem, people propose prioritized experience replay [14]. However, the 'priority' here only considers information from the agent's own policy, e.g., time difference (TD) error. In this paper, we design a comprehensive importance function which combines the TD error and the utility improvement (UI) together.

To summarize, the contributions of this paper are (1) We reformulate the reinforced feature selection into a single-agent framework by proposing a traverse strategy; (2) We design an off-policy Monte Carlo method to implement the proposed framework; (3) We propose an early stopping criteria to improve the training efficiency. (4) We propose reward-level and training-level interactive strategies to improve the training efficiency. (5) We propose an incremental descriptive statistics method to reduce the computational burden in state representation. (6) We design extensive experiments to reveal the superiority of the proposed method.

2 Preliminaries

We first introduce some preliminary knowledge about the Markov decision process(MDP) and the Monte Carlo method to solve MDP, then we give a brief description of multi-agent reinforced feature selection.

2.1 Markov decision process

Markov decision process (MDP) is defined by a tuple $\mathbf{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \gamma, \mathcal{P}\}$, where state space $\mathcal{S}$ is finite, action space $\mathcal{A}$ is pre-defined, reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a mapping function from state-action pair to a scalar, $\gamma \in [0, 1]$ is a discount factor and $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the transition probability from state-action pair to the next state. In this paper, we study the most popular case when the environment is deterministic and thus $\mathcal{P} \equiv 1$. We use superscripts to discriminate different episode and use subscripts to denote the time step inside the episode, e.g., s_t^i, a_t^i denote the state and action at time t in the i-th episode.

2.2 Monte Carlo for solving MDP

Monte Carlo method can take samples from the MDP to evaluate and improve its policy. Specifically, at the i-th iteration, with a behavior (sampling) policy b^i, we can derive an episode $\mathbf{x}^i = \{x_1^i, x_2^i, \ldots, x_t^i, \ldots x_N^i\}$, where $x_t^i = (s_t^i, a_t^i, r_t^i)$ is a sample consisting state, action and reward. With the episode, we can evaluate the Q value $Q_{\pi^i}(s_t^i, a_t^i)$ over our policy (detailed in Sect. 3.1) and improve it by Bellman optimality:

$$\pi^{i+1}(s) = \text{argmax}_a Q_{\pi^i}(s, a) \tag{1}$$

With the evaluation-improvement process going on, the policy π becomes better and better and can finally converge to the optimal policy. The general process is

$$\pi^0 \xrightarrow{E} Q_{\pi^0} \xrightarrow{I} \pi^1 \xrightarrow{E} Q_{\pi^1} \xrightarrow{I} \ldots \pi^M \xrightarrow{I} Q_{\pi^M} \tag{2}$$

where $\xrightarrow{E}$ denotes the policy evaluation and $\xrightarrow{I}$ denotes the policy improvement. After M iterations, we can achieve an optimal policy. As Eq. 2 shows, the policy evaluation and improvement need many iterations, and each iteration needs one episode $\mathbf{x}$ ($\mathbf{x}^i$ for the i-th iteration) consisting N samples.

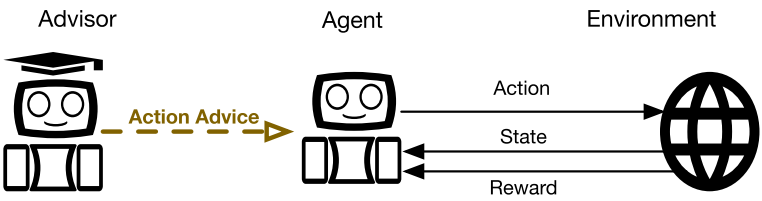


Fig. 2 Classic interactive reinforcement learning. The advisor gives the agent advice at the action level

2.3 Interactive reinforcement learning

As all the steps in this section belong to the same iteration, we omit the superscript i in each denotation for simplicity.

Reinforcement learning is proposed to develop the optimal policy $\pi^*_{\mathcal{M}}(s) = \text{argmax}_a Q^*_{\mathcal{M}}(s, a)$ for an MDP $\mathcal{M}$. The optimal Q-value can be updated by Bellman equation [15]:

$$Q^*_{\mathcal{M}}(s_t, a_t) = \mathcal{R}(s_t, a_t) + \gamma * max_{a_{t+1}} Q^*_{\mathcal{M}}(s_{t+1}, a_{t+1}) \tag{3}$$

Interactive reinforcement learning (IRL) is proposed to accelerate the learning process of reinforcement learning (RL) by providing external action advice to the RL agent [16]. As Fig. 2 shows, for selected advising states, the action of RL agent is decided by the advisor's action advice instead of its own policy. The algorithm to select advising states varies with the problem setting. Typical algorithms for selecting advising states include early advising, importance advising, mistake advising and predictive advising [17]. To better evaluate the utility of the state-action pair (s_t, a_t), we define a utility function $\mathcal{U}(s_t, a_t)$. The utility function can give a feedback of how the action benefits from the state from the advisor's point of view.

2.4 Multi-agent reinforced feature selection

Feature selection aims to find an optimal feature subset $\mathcal{F}'$ from the original feature set $\mathcal{F}$ for a downstream machine learning task $\mathcal{M}$. Recently, the emerging multi-agent reinforced feature selection (MARFS) method [11] formulates the feature selection problem into a multi-agent reinforcement learning task, in order to automate the selection process. As Fig. 3 shows, in the MARFS method, each feature is assigned to a feature agent, and the action of feature agent decides to select/deselect its corresponding feature. It should be noted that the agents simultaneously select features, meaning that there is only one time step inside an iteration, and thus, we omit the subscript here. At the i-th iteration, all agents cooperate to select a feature subset $\mathcal{F}^i$. The next state s^{i+1} is derived by the representation of selected feature subset $\mathcal{F}^i$:

$$s^{i+1} = represent(\mathcal{F}^i) \tag{4}$$

where $\mathcal{F}^i$ is the selected feature subset at time t. $represent$ is a representation learning algorithm which converts the dynamically changing $\mathcal{F}^i_t$ into a fixed-length state vector s^{i+1}. The $represent$ method can be meta descriptive statistics, autoencoder-based deep representation and dynamic-graph-based GCN in [11]. The reward r^i is an evaluation of the selected feature subset $\mathcal{F}^i$:

$$r^i = eval(\mathcal{F}^i) \tag{5}$$

where $eval$ is evaluations of $\mathcal{F}^i$, which can be a supervised metric with the machine learning task $\mathcal{F}$ taking $\mathcal{F}^i$ as input, unsupervised metrics of $\mathcal{F}^i$, or the combination of supervised and unsupervised metrics in [11]. The reward is assigned to each of the feature agent to train their policies. With more and more steps' exploration and exploitation, the policies become more and more smart, and consequently, they can find better and better feature subsets.

3 Proposed method

In this section, we first propose a single-agent Monte Carlo-based reinforced feature selection method. And then, we propose an episode filtering method to improve the sampling efficiency

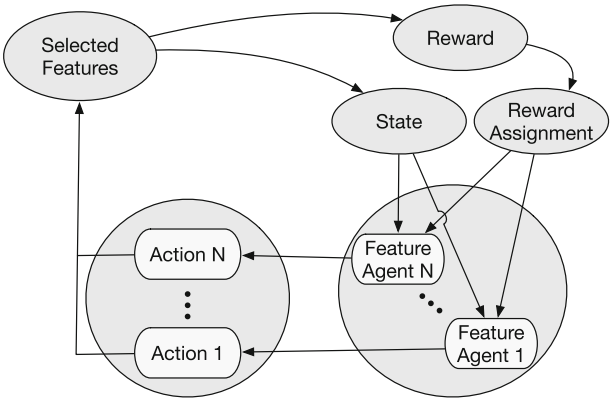


Fig. 3 Multi-agent Reinforced feature selection. Each feature is controlled by one feature agent

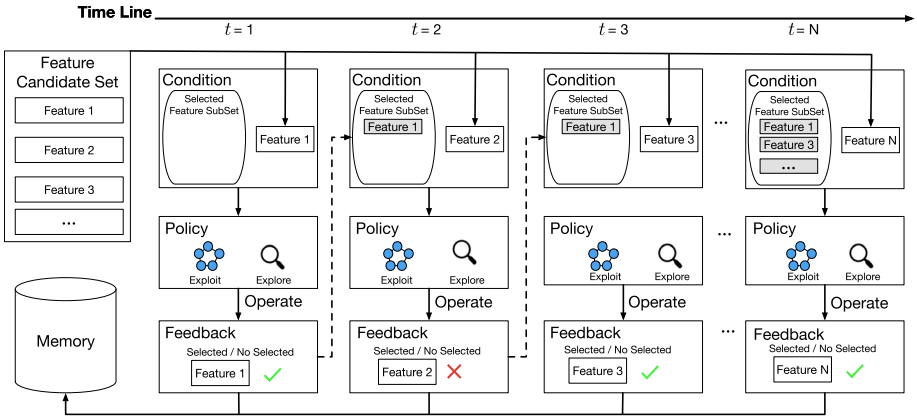


Fig. 4 Single-agent Reinforced feature selection with traverse strategy. At each step, the agent transverses features one by one to decide their selection. The traverse data are stored in the memory to form a training episode

of the Monte Carlo method. In addition, we apply the episode filtering Monte Carlo method to the reinforced feature selection scenario. Finally, we design a reward shaping strategy to improve the training efficiency.

3.1 Monte Carlo-based reinforced feature selection

The MARFS method has proved its effectiveness, however, the multi-agent strategy greatly increases the computational burden and hardware cost. Here, we propose a single-agent traverse strategy and use Monte Carlo method as the reinforcement learning algorithm.

3.1.1 Traverse strategy

As Fig. 4 shows, rather than using N agents to select their corresponding features in the multi-agent strategy, we design one agent to traverse all features one at a time.

In the i-th episode, beginning from time $t = 1$, the behavior policy b^i firstly decides the selection decision (select or not select) for feature 1, and then, at time $t = 2$, b^i decides the selection decision for feature 2. With time going on, the features are traversed one by one, and the selected features form a selected feature subset $\mathcal{F}_t^i$. Meanwhile, this process also generates an episode $\mathbf{x}_N^i = \{x_1^i, x_2^i, \ldots, x_t^i, \ldots x_N^i\}$, where $x_t^i = (s_t^i, a_t^i, r_t^i)$ is a tuple of state, action and reward. The action $a_t^i = 1/0$ is the selection/deselection decision of the t-th feature, the next state s_{t+1}^i is derived by $represent(\mathcal{F}_t^i)$ and the reward r_t^i is derived by $eval(\mathcal{F}_t^i)$.

3.1.2 Monte Carlo method for reinforced feature selection

With the episode generated by the behavior policy b^i, we can evaluate our target policy π^i and improve π^i. Both the behavior policy b^i and the target policy π^i provide the probability of taking action a given a specific state s.

Specifically, we generate an episode $\mathbf{x}_N^i$ by b^i. Then, we calculate the accumulated reward by

$$G^i(s_t^i, a_t^i) = \sum_{j=0}^{t} \gamma^{(t-j)} \cdot r_j^i \tag{6}$$

where $0 \leq \gamma \leq 1$ is a discount factor.

As the state space is extremely large, we use a neural network $Q(s, a)$ to approximate $G(s, a)$.

The target policy π is different from the behavior policy b, and the reward comes from samples derived from policy b, therefore, the accumulated reward of π should be calculated by multiplying an importance sampling weight:

$$\rho_t^i = \frac{\Pi_{j=0}^{t} \pi^i(a_j^i | s_j^i)}{\Pi_{j=0}^{t} b^i(a_j^i | s_j^i)} \tag{7}$$

The $Q_{\pi^i}(s, a)$ can be optimized by minimizing the loss:

$$\mathcal{L}_{\pi^i} = ||Q_{\pi^i}(s_t^i, a_t^i) - \rho_t^i * G^i(s_t^i, a_t^i)||^2 \tag{8}$$

The probability of taking action a for state s under policy π in the next iteration can be calculated by

$$\pi^{i+1}\{a \mid s\} = \frac{exp(Q_{\pi^i}\{a, s\})}{exp(Q_{\pi^i}\{a = 0, s\}) + exp(Q_{\pi^i}\{a = 1, s\})} \tag{9}$$

We develop an ϵ-greedy policy of b based on the Q value from π:

$$b^{i+1}\{a \mid s\} = \begin{cases} 1 - \epsilon & a = \text{argmax}_a Q_{\pi^i}(s, a); \\ \epsilon & otherwise; \end{cases} \tag{10}$$

Algorithm 1 shows the process of Monte Carlo-based feature selection (MCRFS) with traverse strategy.

3.2 Early stopping Monte Carlo-based reinforced feature selection

In many cases, the feature set size N can be very large, meaning that there can be a large number of samples in one episode $\mathbf{x}_N^i$. The problem is, if the sample at time T is bad (the

Algorithm 1 Monte Carlo-Based Reinforced Feature Selection with Traverse Strategy

Require: : Feature set $\mathcal{F} = \{f_1, f_2, ..., f_N\}$, downstream machine learning task $\mathcal{T}$. Initialize the behavior policy b^1, target policy π^1, exploration number M, $\mathcal{F}' = \Phi$.

for $i = 1$ to M **do**
  Initialize state s_1^i.
  for $t = 1$ to N **do**
    Derive action a_t^i with behavior policy $b^i(s_t^i)$.
    Perform a_t^i, getting selected feature subset $\mathcal{F}_t^i$.
    Obtain the next state s_{t+1}^i by $represent(\mathcal{F}_t^i)$ and reward r_t^i by $eval(\mathcal{F}_t^i)$.
  end for
  Update target policy π^{i+1} by Eq. 9 and behavior policy b^{i+1} by Eq. 10.
  if $eval(\mathcal{F}_N^i) > eval(\mathcal{F}')$ **then**
    $\mathcal{F}' = \mathcal{F}_N^i$.
  end if
end for
Return $\mathcal{F}'$.

Chi-squared distance between $b^i(s_t^i)$ and $\pi^i(s_t^i)$ is large), all the subsequent samples (from T to N) in the episode are skewed [18]. The skewed samples not only are a waste time to generate, but also do harm to the policy evaluation, therefore, we need to find some way to stop the sampling when the episode becomes skew.

3.2.1 Incremental importance sampling

Rather than calculating the importance sampling weight for each sample directly by Eq. 7, we here decompose it into an incremental format. Specifically, in the i-th iteration, we define the weight increment:

$$w_t^i = \frac{\pi^i(a_t^i|s_t^i)}{b^i(a_t^i|s_t^i)} \tag{11}$$

and the importance sampling weight can be calculated by

$$\rho_t^i = \rho_{t-1}^i \cdot w_t^i \tag{12}$$

Thus, at each time, we just need to calculate a simple increment to update the weight.

3.2.2 Early stopping Monte Carlo method for reinforced feature selection

We first propose the stopping criteria and then propose a decision history-based traversing strategy to enhance diversity.

Early stopping criteria. We stop the traverse by probability:

$$p_t^i = max(0, 1 - \rho_t^i / v) \tag{13}$$

where $0 \leq v \leq 1$ is the stopping threshold.

And for the acquired episode, we recalculate the importance sampling weight for each sample by

$$w_t^i = p_v^i \cdot \rho_t^i / p_t^i \tag{14}$$

where the p_v can be calculated by

$$p_v^i = \int max(0, 1 - \rho_t^i / v) b^i(\mathbf{s}_t)\, d\mathbf{s}_t \tag{15}$$

Algorithm 2 Monte Carlo-Based Reinforced Feature Selection with Early Stopping Traverse Strategy

Require: Feature set $\mathcal{F} = \{f_1, f_2, ..., f_N\}$, downstream machine learning task $\mathcal{T}$.
Initialize the behavior policy b^1, target policy π^1, exploration number M, $\mathcal{F}' = \Phi$.
for $i = 1$ to M **do**
  Initialize state s_1^i.
  Rank features with their decision history.
  for $t = 1$ to N **do**
    Derive action a_t^i with behavior policy $b^i(s_t^i)$.
    Perform a_t^i, getting selected feature subset $\mathcal{F}_t^i$.
    Obtain the next state s_{t+1}^i by $represent(\mathcal{F}_t^i)$ and reward r_t^i by $eval(\mathcal{F}_t^i)$.
    Break the loop with probability p_t^i derived from Eq. 13;
  end for
  Update target policy π^{i+1} by Eq. 9 and behavior policy b^{i+1} by Eq. 10.
  if $eval(\mathcal{F}_N^i) > eval(\mathcal{F}')$ **then**
    $\mathcal{F}' = \mathcal{F}_N^i$.
  end if
end for
Return $\mathcal{F}'$.

As p_v^i is identical for all samples in the i-th episode regardless of t, the calculation of Eq. 14 does almost no increase to the computation.

Decision history-based traversing strategy. In the i-th iteration, the stopping criteria stop the traverse at time t, and the features after t are not traversed. With more and more traverses, the front features (e.g., f_1 and f_2) are always selected/deselected by the agent, while the backside features (e.g., f_N and f_{N-1}) get very few opportunity to be decided. To tackle this problem, we record the decision times we made on each feature and re-rank their orders to diversify the decision process in the next traverse episode. For example, in the past 5 episodes, if the decision times of feature set $\{f_1, f_2, f_3\}$ are {5,2,4}, then in the 6-th episode, the traverse order is $f_2 \rightarrow f_3 \rightarrow f_1$.

Algorithm 2 shows the process of Monte Carlo-based feature selection (MCRFS) with early stopping traverse strategy. Specifically, we implement the early stopping Monte Carlo-based reinforced feature selection method as follows:

Step 1: Use a random behavior policy b^0 to traverse the feature set. Stop the traverse with the probability in Eq. 13 and get an episode $\mathbf{x}_{N_0}^0$.

Step 2: Evaluate the policy π^0 to get the Q value Q^0 by minimizing Eq. 8, and derive the updated policy π^1 and b^1 from Eqs. 9 to 10, respectively.

Step 3: Update the record of traverse times for each feature. Re-rank feature order. The smaller times one feature was traversed, the more forward order it should get.

Step 4: Use the updated policy π^1 and b^1 to traverse the re-ranked feature set for the next M steps. Derive the policy π^M and b^M. Use π^M to traverse the feature set without stopping criteria, and derive the final feature subset.

3.3 Reward-level interactive reinforcement learning

In reinforcement learning (RL), we aim to obtain the optimal policy for the MDP $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \gamma, \mathcal{P}\}$. However, in IRL, when we change the reward function $\mathcal{R}$ to a more inspiring reward function $\mathcal{R}'$, the original MDP $\mathcal{M}$ is changed to a new MDP $\mathcal{M}' = \{\mathcal{S}, \mathcal{A}, \mathcal{R}', \gamma, \mathcal{P}\}$. Without careful design , the optimal policy derived from $\mathcal{M}'$ would be different from the

Algorithm 3 Reward-Level Interactive Reinforcement Learning

1: Initialize replay memory $\mathcal{D}$; Initialize the Q-value function with random weights; Initialize the advising state number N_a, stop time T;
2: **for** $t = 1$ to T **do**
3: $a_t = \begin{cases} \text{random action} & \text{with probability} \epsilon; \\ max_{a_t} Q(s_t, a_t) & \text{with probability} 1-\epsilon; \end{cases}$
4: Perform a_t, obtaining reward $\mathcal{R}(a_t, s_t)$ and next state s_{t+1};
5: $\mathcal{R}'(s_t, a_t) = \begin{cases} \mathcal{R}(s_t, a_t) & t > N_a; \\ \mathcal{R}(s_t, a_t) + c * (\gamma * \mathcal{U}(s_{t+1}) - \mathcal{U}(s_t)) & t \leq N_a; \end{cases}$
6: Store transition $(s_t, a_t, \mathcal{R}'(s_t, a_t), s_{t+1})$ in $\mathcal{D}$;
7: Randomly sample mini-batch of data from $\mathcal{D}$;
8: Update $Q(s, a)$ with the sampled data;
9: **end for**

optimal policy for $\mathcal{M}$. Here, we give a universal form of reward advice without limitation on the form of utility function $\mathcal{U}(s, a)$:

$$\mathcal{R}'(a_t, s_t) = \mathcal{R}(a_t, s_t) + c * (\gamma * \mathcal{U}(s_{t+1}) - \mathcal{U}(s_t)) \tag{16}$$

where $\mathcal{U}(s_t) = E_{a_t}[\mathcal{U}(a_t, s_t)]$, c is the weight to balance the proportion of the utility function.

We prove that the optimal policies of $\mathcal{M}$ and $\mathcal{M}'$ are identical when the reward advice is Eq. 16:

We firstly subtract $c * \mathcal{U}(s_t)$ from both sides of Eq. 3, and we have

$$\begin{aligned} Q^*_{\mathcal{M}}(s_t, a_t) - c * \mathcal{U}(s_t) = \mathcal{R}(s_t, a_t) \\ + \gamma * max_{a_{t+1}} Q^*_{\mathcal{M}}(s_{t+1}, a_{t+1}) - c * \mathcal{U}(s_t) \end{aligned} \tag{17}$$

We add and subtract $c * \gamma * \mathcal{U}(s_{t+1})$ on the right side:

$$\begin{aligned} & Q^*_{\mathcal{M}}(s_t, a_t) - c * \mathcal{U}(s_t) = \mathcal{R}(s_t, a_t) \\ & + \gamma * max_{a_{t+1}} Q^*_{\mathcal{M}}(s_{t+1}, a_{t+1}) - c * \mathcal{U}(s_t) \\ & + c * \gamma * \mathcal{U}(s_{t+1}) - c * \gamma * \mathcal{U}(s_{t+1}) \\ & = \mathcal{R}(s_t, a_t) + c * \gamma * \mathcal{U}(s_{t+1}) - c * \mathcal{U}(s_t) \\ & + \gamma * max_{a_{t+1}} [Q^*_{\mathcal{M}}(s_{t+1}, a_{t+1}) - c * \mathcal{U}(s_{t+1})] \end{aligned} \tag{18}$$

We define

$$Q^{\partial}_{\mathcal{M}'}(s_t, a_t) = Q^*_{\mathcal{M}}(s_t, a_t) - c * \mathcal{U}(s_t) \tag{19}$$

Then, Eq. 18 has the new form:

$$\begin{aligned} & Q^{\partial}_{\mathcal{M}'}(s_t, a_t) = \mathcal{R}(s_t, a_t) + c * [\gamma * \mathcal{U}(s_{t+1}) - \mathcal{U}(s_t)] \\ & + \gamma * max_{a_{t+1}} Q^{\partial}_{\mathcal{M}'}(s_{t+1}, a_{t+1}) \\ & = \mathcal{R}'(s_t, a_t) + \gamma * max_{a_{t+1}} Q^{\partial}_{\mathcal{M}'}(s_{t+1}, a_{t+1}) \end{aligned} \tag{20}$$

which is the Bellman equation of $Q^{\partial}_{\mathcal{M}'}(s_t, a_t)$ with reward $\mathcal{R}'$, meaning $Q^{\partial}_{\mathcal{M}'}(s_t, a_t)$ is the optimal policy Q-value for MDP $\mathcal{M}'$, i.e.,

$$Q^*_{\mathcal{M}'}(s_t, a_t) = Q^{\partial}_{\mathcal{M}'}(s_t, a_t) \tag{21}$$

We combine Eqs. 19 and 21 and have

$$Q^*_{\mathcal{M}'}(s_t, a_t) = Q^*_{\mathcal{M}}(s_t, a_t) - c * \mathcal{U}(s_t) \tag{22}$$

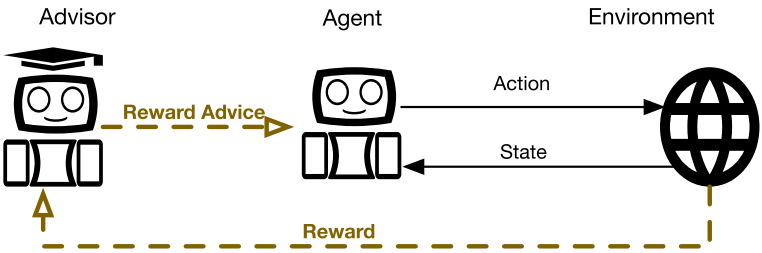


Fig. 5 Reward-level interactive reinforcement learning. The advisor gives advice at the reward level

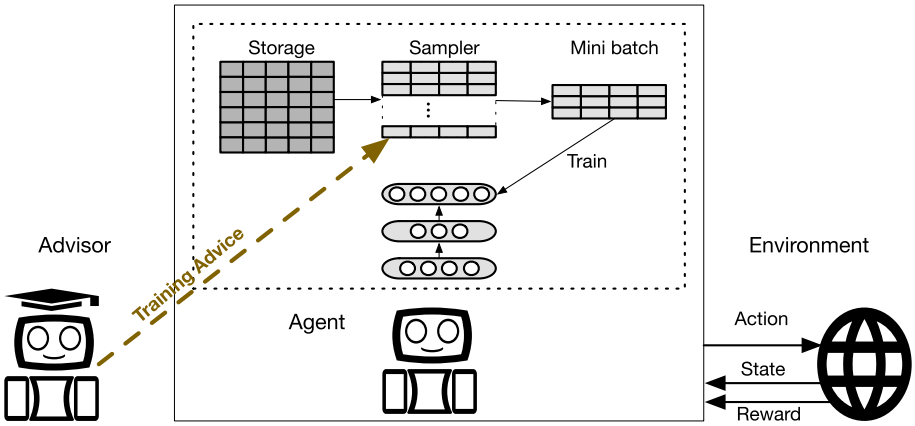


Fig. 6 Training-level interactive reinforcement learning. The advisor gives advice at the training level

Obviously,

$$\begin{aligned} \text{argmax}_{a_t} Q^*_{\mathcal{M}'}(s_t, a_t) &= \text{argmax}_{a_t}[Q^*_{\mathcal{M}}(s_t, a_t) - c * \mathcal{U}(s_t)] \\ &= \text{argmax}_{a_t} Q^*_{\mathcal{M}}(s_t, a_t) \end{aligned} \tag{23}$$

which reveals the optimal policy of MDP $\mathcal{M}'$ with reward $\mathcal{R}'$ is identical to the optimal policy of MDP $\mathcal{M}$ with reward $\mathcal{R}$ (Fig. 5).

As the reward advice $\mathcal{R}'$ consists of more information than the original reward $\mathcal{R}$, it can help the reinforcement learning agent explore the environment more efficiently. We give a detailed description of reward-level IRL in Algorithm 3. Specifically, we adapt the early advising strategy [17] to select the advising states, i.e., the advisor gives advice for the first n states the IRL agent meets.

3.4 Training-level IRL

Reinforcement learning (RL) is a self-improving framework which trains and improves its policy at each step. As Fig. 6 shows, in the training process, the agent stores the training data in the replay memory. In each training step, a mini-batch of samples are randomly picked to train the Q-value function. This store-and-sample technology is called experience replay [19].

Experience replay can remember and reuse transition data from the past experience. However, in the original version of experience replay, all the data in the replay memory are

Algorithm 4 Training-Level Interactive Reinforcement Learning

1: Initialize replay memory $\mathcal{D}$; Initialize the Q-value function with random weights; Initialize the advising state number N_a, stop time T;
2: **for** $t = 1$ to T **do**
3: $a_t = \begin{cases} \text{random action} & \text{with probability} \epsilon; \\ max_{a_t} Q(s_t, a_t) & \text{with probability} 1-\epsilon; \end{cases}$
4: Perform a_t, obtaining reward $\mathcal{R}(s_t, a_t)$ and next state s_{t+1};
5: Store transition $(s_t, a_t, \mathcal{R}(s_t, a_t), s_{t+1})$ in $\mathcal{D}$;
6: Training Data $= \begin{cases} \text{Randomly sample from } \mathcal{D} & t > N_a; \\ \text{Sample with priority in Equation 27 from } \mathcal{D} & t \leq N_a; \end{cases}$
7: Update $Q(s, a)$ with the sampled data;
8: **end for**

randomly sampled without considering their importance and contribution to the policy training. To tackle this problem, people propose prioritized experience replay [14]. Here, we have a research question: Can the advisor in interactive reinforcement learning give advice in the training process with regard to sample importance?

The most popular criterion to evaluate the importance of a transition is time difference (TD) error δ:

$$\delta_t = \mathcal{R}(s_t, a_t) + \gamma * \max_{a_{t+1}} Q_{\mathcal{M}}(s_{t+1}, a_{t+1}) - Q_{\mathcal{M}}(s_t, a_t) \tag{24}$$

which measures the difference between the 'ideal' Q-value and 'calculated' Q-value for the state-action pair (s_t, a_t).

Here, we define utility improvement (UI) η as

$$\eta_t = \mathcal{U}(s_t, a_t) - E_{a_t}[\mathcal{U}(a_t, s_t)] \tag{25}$$

which measures the improvement of utility if we take action a_t at state s_t.

We then integrate the two measurements into a comprehensive importance function:

$$p_t = \mid \delta_t \mid + w * \mid \eta_t \mid \tag{26}$$

where w is a weight to balance the proportion between ID error and utility improvement

Finally, we define the sampling probability of the transitions in replay memory $\mathcal{D}$:

$$P_t = \frac{exp(p_t)}{\sum_{d_t \in \mathcal{D}} exp(p_t)} \tag{27}$$

where $d_t = (s_t, a_t, \mathcal{R}'(a_t, s_t), s_{t+1})$ denotes the transition from t to $t+1$.

As the importance function p_t consists more information than the original importance measurement δ_t, it can help the reinforcement learning agent explore the environment more efficiently. We give a detailed description of reward-level IRL in Algorithm 4. Specifically, we adapt the early advising strategy [17] to select the advising states, i.e., the advisor gives advice for the first n states the IRL agent meets.

3.5 Improving state representation

Assuming we have M data samples and N selected features, the dimensionality of dataset $\boldsymbol{D}$ is $M * N$. We would like to represent $\boldsymbol{D}$ into the state vector. Let n_j be the number of selected features at the j-th iteration step. Then, $M * n_j$ is the dimension of the selected data matrix $\boldsymbol{S}$, which changes over iteration steps. However, the behavior network and target network

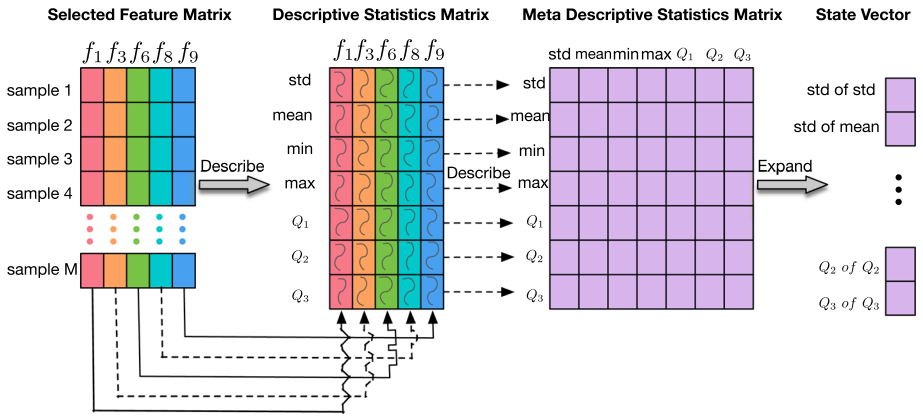


Fig. 7 Incremental descriptive statistics. We extract descriptive statistics twice from the feature subspace to obtain a fixed-length state vector

in Monte Carlo method require a fixed-length vector as the state vector $\boldsymbol{s}$. In one word, we are required to obtain a fixed-length vector $\boldsymbol{s}$ from the selected matrix $\boldsymbol{S}$, whose dimension changes over iterations.

The dynamic-graph-based GCN in [11] has been proved effective in representing the state when the selected features are dynamically changing. However, this representation method is based on deep neural network, and it requires a lot of computational resources to train. In this section, we propose an incremental descriptive statistics method to reduce the computational burden, as well as capture the feature-feature relationship.

Incremental descriptive statistics of feature subspace. Figure 7 shows how we extract the meta data of descriptive statistics from the selected data matrix via a two-step procedure.

Step 1: We extract descriptive statistics of the selected data matrix $\boldsymbol{S}$, including the standard deviation, minimum, maximum and Q1 (the first quartile), Q2 (the second quartile), and Q3 (the third quartile). Specifically, we extract the seven descriptive statistics of each feature (**column**) in $\boldsymbol{S}$, and thus, obtain a *descriptive statistics matrix* $\boldsymbol{D}$ with size of $7 * n_j$.

Step 2: We extract the seven descriptive statistics of each **row** in the descriptive statistics matrix $\boldsymbol{D}$, and obtain a *meta descriptive statistics matrix* $\boldsymbol{D}'$ with a size of $\mathbf{7} * 7$.

Finally, we link each column $\boldsymbol{D}'$ together into the state vector $\boldsymbol{s}$ with a fixed length of 49.

Please be notified that, when the behavior policy traverses features, the selected features are added to the selected dataset one by one, thus we don't have to calculate the column descriptive statistics every time. Instead, we record the column statistics, and we only need to calculate the statistics of the newly added feature. This **incremental** update can further improve the efficiency of the state representation.

3.6 Comparison with prior literature

Compared with filter methods, our methods capture feature interactions. For example, the univariate feature selection [2] method selects the best feature set based on univariate statistical scores. Typical scores are χ^2, p value, mutual information, etc. The ranking of these scores is only capable of measuring the importance of features from the perspective of one

feature, instead of a feature set. Our method can capture the set-level interaction relationship and measure features from the set level.

Compared with wrapper methods, our methods reduce the search space. Typical wrapper methods [5, 6] explore the full search space, which is of low efficiency. Some other wrapper methods like beam search [20, 21] can also reduce the search space, but they are lack of stopping rules to stop the selection process when the marginal improvement is not significant.

Compared with embedded methods, our methods don't rely on strong structured assumptions. The most widely used embedded methods are LASSO [9] and decision tree [10]. They can only be embedded in specific downstream tasks. For example, the selected feature subset by decision tree is not optimal for a linear regression model.

Compared with multi-agent reinforcement learning feature selection, our methods achieve parallel performance with lower computational cost. Multi-agent methods [11, 22] require as many agents as the features. When the feature set is large, the training and inferring of multiple agents are computationally expensive. Our methods alleviate this problem by proposing a single-agent traverse strategy with an early stopping criteria.

4 Experimental results

We conduct extensive experiments on real-world datasets to study: (1) the overall performance of early stopping Monte Carlo-based reinforced feature selection (**ES-MCRFS**); (2) the training efficiency of the early stopping criteria; (3) the sensitivity of the threshold in the early stopping criteria; (4) the computational burden of the traverse strategy; (5) the decision history-based traverse strategy; (6) the behavior policy in the ES-MCRFS.

4.1 Experimental setup

4.1.1 Data description

We use six publicly available datasets on classification task to validate our methods, i.e., Forest Cover (FC) dataset [23], Spambase (Spam) dataset [24], Insurance Company Benchmark (ICB) dataset [25], Arrhythmia (Arrhy) dataset [26], AP_Omentum_Ovary (APOO) dataset [27], and Higgs Boson (HB) [28]. The statistics of the datasets are in Table 2.

4.1.2 Evaluation metrics

In the experiments, we have classification as the downstream task for feature selection problem, therefore, we use the two most popular evaluation metrics for classification task:
Accuracy is given by $\text{Acc} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}}$, where TP, TN, FP, FN are true positive, true negative, false positive and false negative for all classes.
F1-score is given by $F1 = \frac{2*P*R}{P+R}$, where $P = \frac{\text{TP}}{\text{TP}+\text{FP}}$ is precision and $R = \frac{\text{TP}}{\text{TP}+\text{FN}}$ is recall.

Table 2 Statistics of datasets

	FC	Spam	ICB	Arrhy	APOO	HB
Features	54	57	86	274	10,936	28
Samples	15,120	4,601	5,000	452	275	50,000

4.1.3 Baseline algorithms

We compare our proposed ES-MCRFS method with the following baselines: (1) **K-Best** ranks features by unsupervised scores with the label and selects the top k highest scoring features [1]. In the experiments, we set k equals to half of the number of input features.(2) **LASSO** conducts feature selection via $l1$ penalty [9]. The hyperparameter in LASSO is its regularization weight λ which is set to 0.15 in the experiments. (3) **GFS** selects features by calculating the fitness level for each feature to generate better feature subsets via crossover and mutation [29]. (4) **mRMR** ranks features by minimizing feature's redundancy and maximizing their relevance with the label[30]. (5) **RFE** selects features by recursively selecting smaller and smaller feature subsets [31]. (6)**MARFS** is a multi-agent reinforcement learning-based feature selection method [11]. It uses M feature agents to control the selection/deselection of the M features. Besides, we also compare our method with its variant without early stopping strategy, i.e., Monte Carlo-based reinforced feature selection **MCRFS**. (6) **FS**, short for forward selection, keeps adding new features to the selected feature set until no further performance improvement is observed. Here, we stop the selection when the improvement is less than 5% compared with base performance without feature selection methods. (7) **BS**, short for beam search, is a generation method of **FG**. It always keeps the best k features at each step [21]. Here, we set $k = 5$ to align with the reference. (8) **DT**, short for decision tree, evaluates feature importance by Gini impurity.

4.1.4 Implementation

In the experiments, for all deep networks, we set mini-batch size to 16 and use AdamOptimizer with a learning rate of 0.01. For all experience replays, we set memory size to 200. We set the Q network in our methods as a two-layer ReLU with 64 and 8 nodes in the first and second layer. The classification algorithm we use for evaluation is a random forest with 100 decision trees. The stop time is set to 3000 steps. The state representation method in reinforced feature selection is an autoencoder method whose encoder/decoder network is a two-layer ReLU with 128 and 32 nodes in the first and second layer.

4.1.5 Environmental setup

The experiments were carried on a server with an I9-9920X 3.50GHz CPU, 128GB memory and a Ubuntu 18.04 LTS operation system.

4.2 Overall performance

We compare the proposed ES-MCRFS method with baseline methods and its variant with regard to the predictive accuracy. As Table 3 shows, the MCRFS, which simplify the reinforced feature selection into a single-agent formulation, achieves similar performance with the multi-agent MARFS. With the help of traverse strategy and early stopping criteria, the ES-MCRFS outperforms all the other methods.

4.3 Sensitivity study of early stopping criteria

We study the threshold sensitivity in the early stopping criteria by differing the threshold v and evaluate the predictive accuracy. Figure 8 shows that the optimal threshold for the four

Table 3 Overall performance

		FC		Spam		ICB		Arrhy		APOO		HB	
		Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Algorithms	K-Best	0.7904	0.8058	0.9207	0.8347	0.8783	0.8321	0.6382	0.6406	0.6683	0.6742	0.6507	0.6136
	LASSO	0.8438	0.8493	0.9143	0.8556	0.8801	0.8507	0.6293	0.6543	0.6746	0.6824	0.6683	0.6342
	GFS	0.8498	0.8350	0.9043	0.8431	0.9099	0.6370	0.6406	0.6550	0.6824	0.6959	0.6812	0.6624
	mRMR	0.8157	0.8241	0.8980	0.8257	0.8998	0.8423	0.6307	0.6368	0.6974	0.7062	0.6742	0.6585
	RFE	0.8046	0.8175	0.9351	0.8480	0.9045	0.8502	0.6452	0.6592	0.7103	0.7189	0.6791	0.6609
	MARFS	0.8653	0.8404	0.9219	0.8742	0.8902	0.8604	0.7238	0.6804	0.7302	0.7399	0.7154	0.6794
	MCRFS	0.8688	0.8496	0.9256	0.8738	0.8956	0.8635	0.7259	0.7152	0.7159	0.7252	0.7057	0.6781
	FS	0.8492	0.8247	0.9211	0.8905	0.8863	0.8517	0.7075	0.6963	0.6703	0.6824	0.7031	0.6503
	BS	0.8524	0.8385	0.9247	0.8941	0.8935	0.8599	0.7239	0.7052	0.7011	0.7084	0.7133	0.6684
	DT	0.8167	0.8083	0.9180	0.8313	0.8792	0.8572	0.6402	0.6582	0.6824	0.6940	0.6723	0.6482
	ES-MCRFS	**0.8942**	**0.8750**	**0.9402**	**0.9067**	**0.9187**	**0.8803**	**0.7563**	**0.7360**	**0.7385**	**0.7469**	**0.7202**	**0.6813**

Bold means the best performance

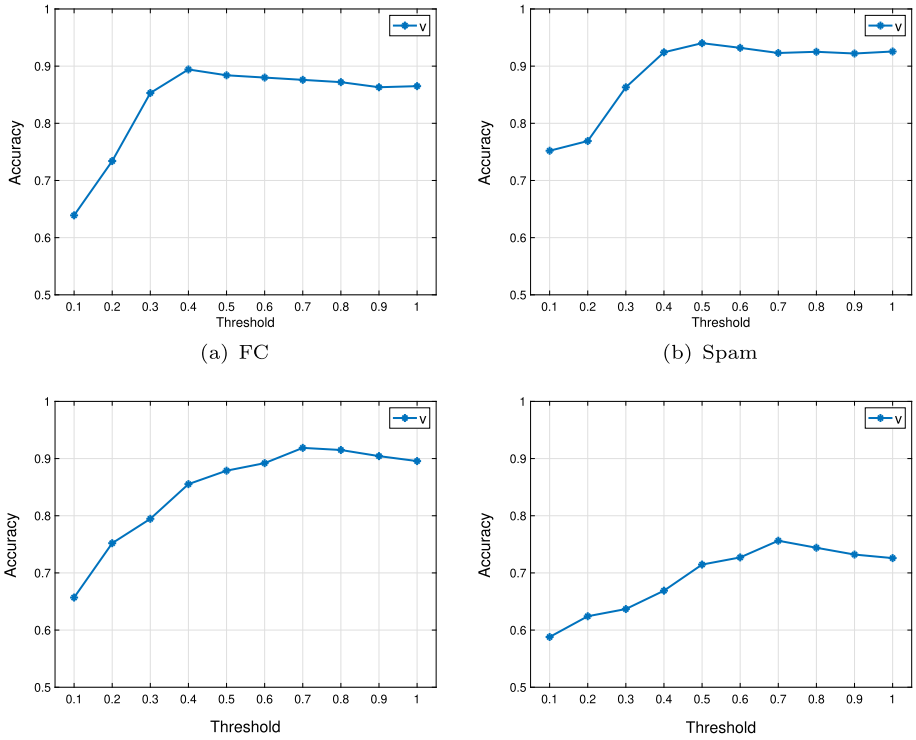


Fig. 8 Threshold sensitivity of early stopping criteria

datasets are 0.4, 0.5, 0.7, 0.7. It reveals that the early stopping criteria are sensitive to the pre-defined threshold, and the optimal threshold varies on different datasets.

4.4 Training efficiency of early stopping criteria

We compare the predictive accuracy with different numbers of training episodes to study the training efficiency of the early stopping. Figure 9 shows that with early stopping criteria, the Monte Carlo reinforced feature selection can achieve convergence more quickly, and the predictive accuracy can be higher after convergence.

4.5 Study of the behavior policy

We study the difference between random behavior policy and the ϵ-greedy policy presented in Eq. 10. We combine the two policies with MCRFS and ES-MCRFS, respectively. Figure 10 shows that the ϵ-greedy policy outperforms the random behavior policy on all datasets.

4.6 Computational burden of traverse strategy

We compare the computational burden of the MCRFS which uses single agent and the traverse strategy to substitute the multi-agent strategy in the MARFS. Table 4 shows that the CPU

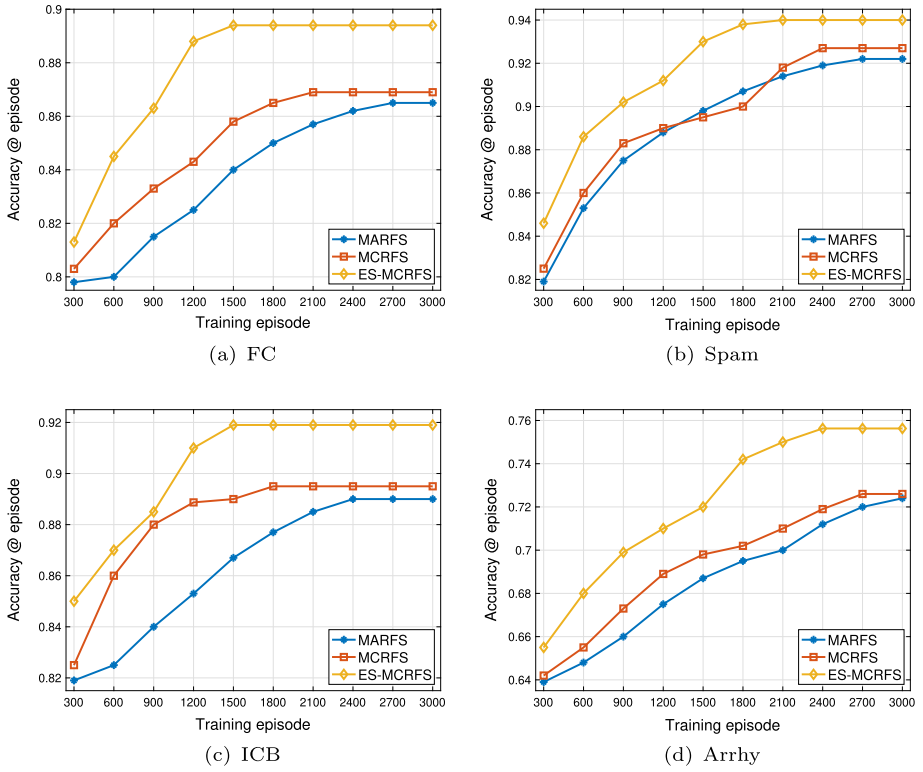


Fig. 9 Predictive accuracy on training step

and memory cost when implementing the two methods. Our method MCRFS requires less computational resources than the multi-agent MARFS.

4.7 Decision history-based traverse strategy

We study the decision history-based traverse strategy by comparing its performance with the vanilla traverse strategy on ES-MCRFS. Table 5 shows that the decision history can significantly improve performance of the traverse strategy.

4.8 Training efficiency of reward-level interactive strategy

We compare the predictive accuracy with different numbers of training episodes to study the training efficiency of the reward-level interactive (RI) strategy. Figure 11 shows that with RI, the Monte Carlo reinforced feature selection can achieve convergence more quickly. However, as the ES-MCRFS already achieves good performance, the RI can not improve its final performance.

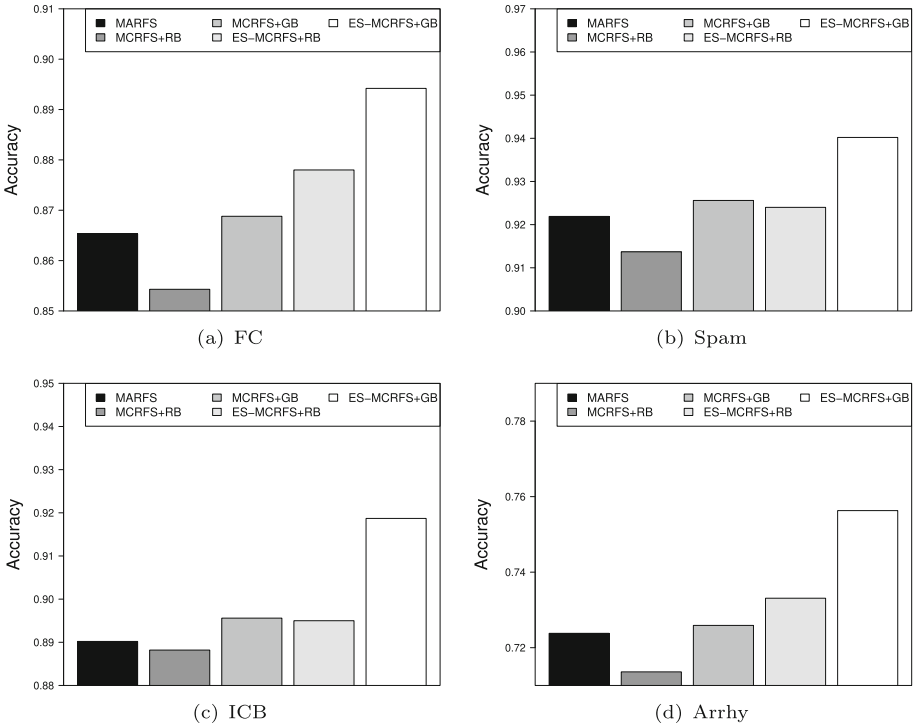


Fig. 10 Predictive accuracy on different training strategies. RB for random behavior policy and GB for ϵ-greedy behavior policy

Table 4 CPU and memory (in MB) occupation

	FC		Spam		ICB		Arrhy	
	CPU	Mem	CPU	Mem	CPU	Mem	CPU	Mem
MARFS	72%	1531	75%	1502	86%	1797	97%	4759
MCRFS	57%	1429	54%	1395	59%	1438	55%	1520

Table 5 Traverse strategy ablation. DH for decision history

	FC		Spam		ICB		Arrhy	
	Acc	F1	Acc	F1	Acc	F1	Acc	F1
No DH	0.75	0.82	0.83	0.79	0.75	0.81	0.59	0.56
With DH	0.89	0.88	0.94	0.91	0.92	0.88	0.76	0.74

4.9 Study of the utility function

We define the utility function $\mathcal{U}$ as the combination of relevance (Rv) function and redundancy (Rd) function. Here, we study the impact of the two components for the utility function. Table 6 shows that when we use Rv independently as the utility function, its performance is better than the Rd. This is because Rv evaluates the relationship between features and the label,

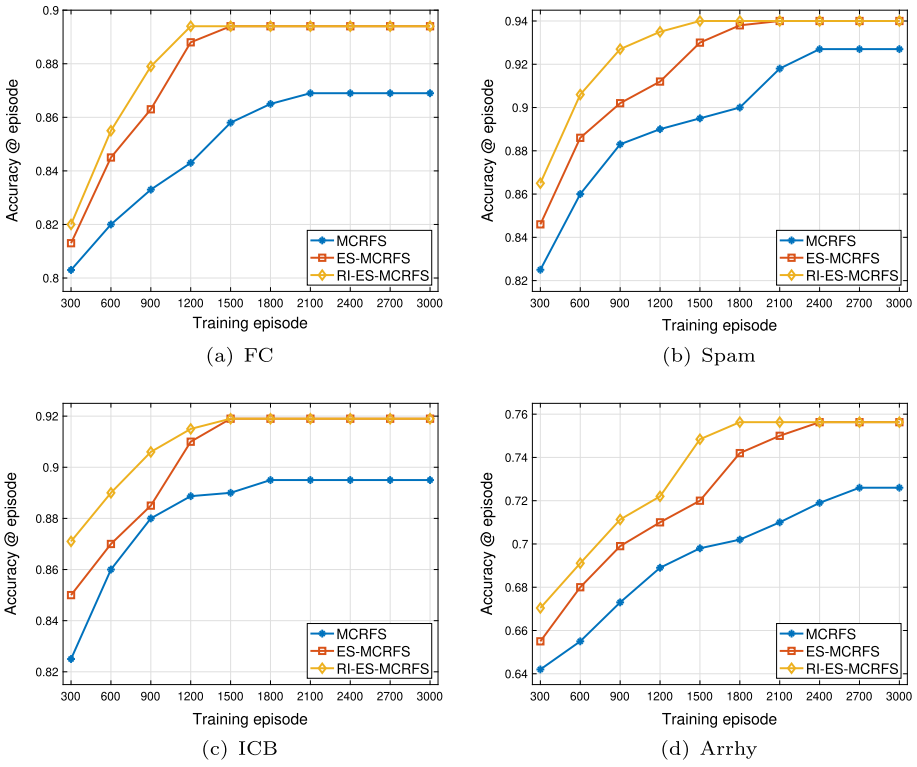


Fig. 11 Predictive accuracy on training step

which is directly related to the classification task, while Rd evaluates the relationship among features, which is an indirect evaluation to the classification task. The combination of the two functions ($Rv - Rd$) as the utility function significantly outperforms each of the independent functions, revealing the Rd and Rv coordinate and make up each other's shortage.

4.10 Study of multi-level IRL methods

After investigating different levels of IRL methods, we are curious about the combination of them. In this section, we study the case when we have more than one level of advice. We consider four cases: (1) RLFS that has no advice; (2) ARLFS that has action-level advice; (3) RRLFS that has reward-level advice; (4) TRLFS that has training-level advice; (5) ARRLFS that has both action-level and reward-level advice; (6) ATRLFS that has both action-level and training-level advice; (7) RTRLFS that has both reward-level and training-level advice; (8) ARTRLFS that has all of the action-level, reward-level and training-level advice.

Figure 12 shows that when combining the training-level advice with the action-level advice (i.e., ATRLFS) or with the reward-level advice, the acceleration can be more significant. However, when combing the action-level advice with the reward-level advice, the improvement is not significant. The reason may be that the mechanism of action-level advice and the reward-level advice are similar, while the training-level advice is different. This makes training-level advice can help to make up the shortage of the other two levels.

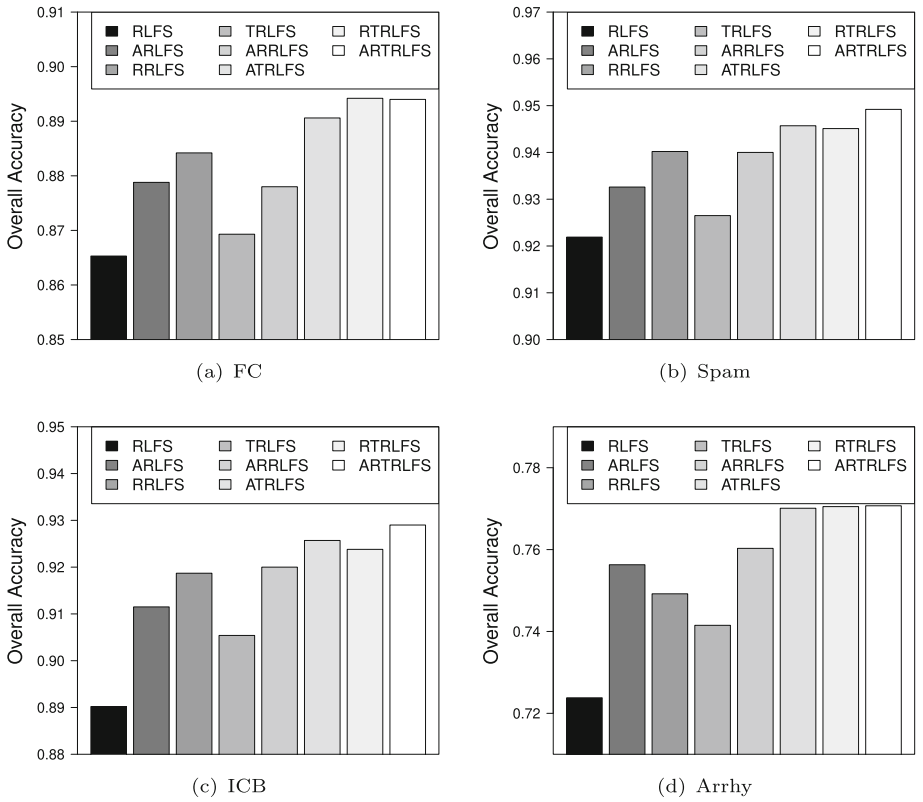


Fig. 12 Exploration accuracy with different advice

Table 6 Performance with different utility function

		FC		Spam		ICB		Arrhy	
		Acc	F1	Acc	F1	Acc	F1	Acc	F1
Utility	Rd	0.8689	0.8433	0.9250	0.8749	0.8997	0.8788	0.7340	0.6955
	Rv	0.8703	0.8507	0.9317	0.8831	0.9001	0.8793	0.7393	0.7143
	$Rv - Rd$	**0.8842**	**0.8650**	**0.9402**	**0.8949**	**0.9117**	**0.8903**	**0.7492**	**0.7258**

Bold means the best performance

4.11 Study of incremental descriptive statistics

We study the incremental descriptive statistics (IDS) method by comparing its performance with the state-of-the-art dynamic-graph-based GCN representation in [11], under our proposed MCRFS method. Table 7 shows that the IDS method can achieve similar performance with the GCN representation method.

Table 8 shows the CPU and memory cost when implementing the two methods. Our method, IDS, requires less computational resources than the GCN representation method.

Table 7 Performance comparison of different state representation methods

	FC		Spam		ICB		Arrhy	
	Acc	F1	Acc	F1	Acc	F1	Acc	F1
GCN	0.89	0.88	0.94	0.91	0.92	0.88	0.76	0.74
IDS	0.87	0.89	0.93	0.92	0.93	0.88	0.76	0.75

Table 8 CPU and memory (in MB) cost of different state representation methods

	FC		Spam		ICB		Arrhy	
	CPU	Mem	CPU	Mem	CPU	Mem	CPU	Mem
GCN	57%	1429	54%	1395	59%	1438	55%	1520
IDS	51%	1369	52%	1206	48%	1201	43%	1130

5 Related work

5.1 Efficient sampling in reinforcement learning

Reinforcement learning is a trial-and-error-based method, which requires high-quality samples to train its policy. It is always a hot topic to pursue efficient sampling for reinforcement learning. One research direction is to generate training samples with high quality based on the importance sampling technology, such as rejection control [32] and marginalized importance sampling [33]. These methods basically control the sampling process based on the importance sampling weight. Another research direction is to sample diversified sample from different policy parameters. The diversity partially contributes to the exploration and thus has better performance on some specific tasks [34]. However, these methods suffer from slow convergence and no theoretical guarantee [35]. Besides, there are other attempts to develop sample efficient reinforcement learning, such as curiosity-driven exploration and hybrid optimization [36, 37].

5.2 Feature selection

Feature selection can be categorized into three types, i.e., filter methods, wrapper methods and embedded methods [22, 38]. Filter methods rank features only by relevance scores and only top-ranking features are selected. The representative filter methods is the univariate feature selection [2]. The representative wrapper methods are branch and bound algorithms [7, 8]. Wrapper methods are supposed to achieve better performance than filter methods since they search on the whole feature subset space. Evolutionary algorithms [5, 6] low down the computational cost but could only promise local optimum results. Embedded methods combine feature selection with predictors more closely than wrapper methods. The most widely used embedded methods are LASSO [9] and decision tree [10].

5.3 Interactive reinforcement learning

Interactive reinforcement learning (IRL) is proposed to accelerate the learning process of reinforcement learning. Early work on the IRL topic can be found in [39], where the authors present a general approach to making robots which can improve their performance from

experiences as well as from being taught. Unlike the imitation learning which intends to learn from an expert other than the environment [40, 41], IRL sticks to learning from the environment and the advisor is only an advice-provider in its apprenticeship [42, 43]. As the task for the advisor is to help the agent pass its apprenticeship, the advisor has to identify which states belong to the apprenticeship. In [17], the authors study the advising state selection and propose four advising strategies, i.e., early advising, importance advising, mistake correcting and predictive advising.

6 Conclusion remarks

6.1 Summary

In this paper, we study the problem of improving the training efficiency of reinforced feature selection (RFS). We propose a traverse strategy to simplify the multi-agent formulation of the RFS to a single-agent framework, an implementation of Monte Carlo method under the framework, and two strategies to improve the efficiency of the framework. We propose reward-level and training-level interactive reinforcement learning to improve the effectiveness and efficiency. We further reduce the computational cost by proposing an incremental descriptive statistics representation method.

6.2 Theoretical implications

The single-agent formulation reduces the requirement of computational resources, the early stopping strategy improves the training efficiency, the decision history-based traversing strategy diversifies the training process, and the interactive reinforcement learning accelerates the training process without changing the optimal policy.

6.3 Practical implications

Experiments show that the Monte Carlo method with the traverse strategy can significantly reduce the hardware occupation in practice, the decision history-based traverse strategy can improve performance of the traverse strategy, the interactive reinforcement learning can improve the training of the framework.

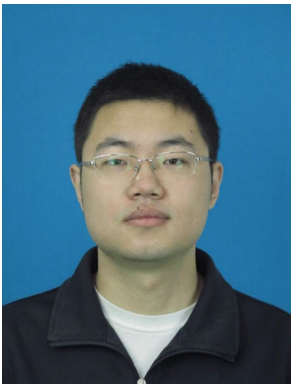
6.4 Limitations and future work

Our method can be further improved from the following aspects: (1) The framework can be adapted into a parallel framework, where more than one (but much smaller than the feature number) agents work together to finish the traverse; (2) Besides reward level and training level, the interactive reinforcement learning can obtain advice from other levels, e.g., policy level. (3) The framework can be implemented on any other reinforcement learning frameworks, e.g., deep Q-network, actor critic and proximal policy optimization (PPO).

References

1. Yang Y, Pedersen JO (1997) A comparative study on feature selection in text categorization. Icml 97:412–420
2. Forman G (2003) An extensive empirical study of feature selection metrics for text classification. J Mach Learn Res 3(3):1289–1305
3. Hall MA (1999) Feature selection for discrete and numeric class machine learning
4. Yu L, Liu H (2003) Feature selection for high-dimensional data: a fast correlation-based filter solution. In: Proceedings of the 20th International conference on machine learning (ICML-03), pp 856–863
5. Yang J, Honavar V (1998) Feature subset selection using a genetic algorithm, pp 117–136
6. Kim Y, Street WN, Menczer F (2000) Feature selection in unsupervised learning via evolutionary search. In: Proceedings of the Sixth ACM SIGKDD International conference on knowledge discovery and data mining, pp 365–369 ACM
7. Narendra PM, Fukunaga K (1977) A branch and bound algorithm for feature subset selection. IEEE Trans Comput 9:917–922
8. Kohavi R, John GH (1997) Wrappers for feature subset selection. Artif Intell 97(1–2):273–324
9. Tibshirani R (1996) Regression shrinkage and selection via the lasso. J R Stat Soc Series B (Methodol), pp 267–288
10. Sugumaran V, Muralidharan V, Ramachandran K (2007) Feature selection using decision tree and classification through proximal support vector machine for fault diagnostics of roller bearing. Mech Syst Signal Process 21(2):930–942
11. Liu K, Fu Y, Wang P, Wu L, Bo R, Li X (2019) Automating feature subspace exploration via multi-agent reinforcement learning. In: Proceedings of the 25th ACM SIGKDD International conference on knowledge discovery and data mining, pp 207–215
12. Fan W, Liu K, Liu H, Wang P, Ge Y, Fu Y (2020) Autofs: automated feature selection via diversity-aware interactive reinforcement learning. arXiv preprint arXiv:2008.12001
13. Sutton RS, Barto AG (2018) Reinforcement learning: an introduction
14. Schaul T, Quan J, Antonoglou I, Silver D (2015) Prioritized experience replay. arXiv preprint arXiv:1511.05952
15. Bellman R (1957) Kalaba R Dynamic programming and statistical communication theory. Proc Natl Acad Sci USA 43(8):749
16. Suay HB, Chernova S (2011) Effect of human guidance and state space size on interactive reinforcement learning. In: 2011 Ro-Man, pp 1–6. IEEE
17. Torrey L, Taylor M (2013) Teaching on a budget: agents advising agents in reinforcement learning. In: Proceedings of the 2013 International conference on autonomous agents and multi-agent systems, pp 1053–1060
18. MacEachern SN, Clyde M, Liu JS (1999) Sequential importance sampling for nonparametric bayes models: the next generation. Can J Stat 27(2):251–267
19. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602
20. Gupta P, Doermann D, DeMenthon D (2002) Beam search for feature selection in automatic svm defect classification. In: 2002 International conference on pattern recognition, vol 2, pp 212–215 . IEEE
21. Fraiman N, Li Z (2022) Beam search for feature selection. arXiv preprint arXiv:2203.04350
22. Liu K, Fu Y, Wu L, Li X, Aggarwal C, Xiong H (2021) Automated feature selection: a reinforcement learning perspective. IEEE Trans Knowl Data Eng. https://doi.org/10.1109/TKDE.2021.3115477
23. Blackard JA (2015) Kaggle forest cover type prediction. [EB/OL]. https://www.kaggle.com/c/forest-cover-type-prediction/data
24. Dua D, Graff C (2017) UCI machine learning repository. http://archive.ics.uci.edu/ml
25. Van Der Putten P, van Someren M (2000) Coil challenge 2000: the insurance company case
26. Guvenir HA, Acar B, Demiroz G, Cekin A (1997) A supervised machine learning algorithm for arrhythmia analysis. Comput Cardiol 1997:433–436
27. Stiglic G, Kokol P (2010) Stability of ranked gene lists in large microarray analysis studies. J Biomed Biotechnol. https://doi.org/10.1155/2010/616358
28. Baldi P, Sadowski P (2014) Whiteson D Searching for exotic particles in high-energy physics with deep learning. Nat Commun 5(1):1–9
29. Leardi R (1996) Genetic algorithms in feature selection, pp 67–86
30. Peng H, Long F, Ding C (2005) Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. IEEE Trans Pattern Anal Mach Intell 27(8):1226–1238
31. Granitto PM, Furlanello C, Biasioli F (2006) Gasperi F Recursive feature elimination with random forest for ptr-ms analysis of agroindustrial products. Chemom Intell Lab Syst 83(2):83–90

32. Liu JS, Chen R, Wong WH (1998) Rejection control and sequential importance sampling. J Am Stat Assoc 93(443):1022–1031
33. Xie T, Ma Y, Wang Y-X (2019) Towards optimal off-policy evaluation for reinforcement learning with marginalized importance sampling. In: Advances in Neural Information Processing Systems, pp 9668–9678
34. Fortunato M, Azar MG, Piot B, Menick J, Osband I, Graves A, Mnih V, Munos R, Hassabis D, Pietquin O, et al (2017) Noisy networks for exploration. arXiv preprint arXiv:1706.10295
35. Yu Y (2018) Towards sample efficient reinforcement learning. In: IJCAI, pp 5739–5743
36. Raginsky M, Rakhlin A, Telgarsky M (2017) Non-convex learning via stochastic gradient langevin dynamics: a nonasymptotic analysis. arXiv preprint arXiv:1702.03849
37. Wang D, Wang P, Zhou J, Sun L, Du B, Fu Y (2020) Defending water treatment networks: Exploiting spatio-temporal effects for cyber attack detection. In: 2020 IEEE International conference on data mining (ICDM), pp 32–41. IEEE
38. Zhao X, Liu K, Fan W, Jiang L, Zhao X, Yin M, Fu Y (2020) Simplifying reinforced feature selection via restructured choice strategy of single agent. In: 2020 IEEE International conference on data mining (ICDM), pp 871–880 . IEEE
39. Lin LJ (1991) Programming robots using reinforcement learning and teaching. In: AAAI, pp 781–786
40. Schaal S (1999) Is imitation learning the route to humanoid robots? Trends Cogn Sci 3(6):233–242
41. Ho J, Ermon S (2016) Generative adversarial imitation learning. In: Advances in neural information processing systems, pp 4565–4573
42. Knox WB, Stone P, Breazeal C (2013) Teaching agents with human feedback: a demonstration of the tamer framework. In: Proceedings of the companion publication of the 2013 international conference on intelligent user interfaces companion, pp 65–66
43. Wang D, Wang P, Liu K, Zhou Y, Hughes CE, Fu Y (2021) Reinforced imitative graph representation learning for mobile user profiling: an adversarial training perspective. Proc AAAI Conf Artif Intell 35:4410–4417



Kunpeng Liu is an Assistant Professor in the Department of Computer Science, Portland State University. Prior to that, he received the Ph.D. degree from the CS Department of the University of Central Florida. He received both his M.E. degree and B.E. degree from the Department of Automation, University of Science and Technology of China (USTC). He has rich research experience in industry research labs, such as Geisinger Medical Research, Microsoft Research Asia, and Nokia Bell Labs. He has published in refereed journals and conference proceedings, such as IEEE TKDE, ACM SIGKDD, IEEE ICDM, AAAI, and IJCAI. He has received a Best Paper Runner-up Award from IEEE ICDM 2021.

Dongjie Wang received the BE degree from the Sichuan University, China, 2016, the MS degree from the Southwest Jiaotong University, China, 2017. He is currently working toward the PhD degree at the University of Central Florida (UCF). His research interests include Anomaly Detection, Generative Adversarial Network, Reinforcement Learning and Spatio-temporal Data Mining.

Wan Du received the BE and MS degrees in electrical engineering from Beihang University, China, in 2005 and 2008, respectively, and the PhD degree in electronics from the University of Lyon (cole centrale de Lyon), France, in 2011. He is currently an assistant professor with the Department of Computer Science and Engineering with the University of California, Merced. His research interests include the Internet of Things, cyber-physical system, distributed networking systems, and mobile systems.

Dapeng Oliver Wu received a B.E. degree in electrical engineering from Huazhong University of Science and Technology, Wuhan, China, in 1990, an M.E. degree in electrical engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 1997, and a Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 2003. He is Yeung Kin Man Chair Professor of Network Science, and Chair Professor of Data Engineering at the Department of Computer Science, City University of Hong Kong. His research interests are in the areas of networking, communications, signal processing, computer vision, machine learning, smart grid, and information and network security. He was elected as a Distinguished Lecturer by IEEE Vehicular Technology Society in 2016. He was an elected member of Multimedia Signal Processing Technical Committee, IEEE Signal Processing Society. He is an IEEE Fellow.

Yanjie Fu is an assistant professor in the Department of Computer Science at the University of Central Florida. He received his Ph.D. degree from Rutgers, the State University of New Jersey in 2016, the B.E. degree from University of Science and Technology of China in 2008, and the M.E. degree from Chinese Academy of Sciences in 2011. He has research experience in industry research labs, such as Microsoft Research Asia and IBM Thomas J. Watson Research Center. He has published prolifically in refereed journals and conference proceedings, such as IEEE TKDE, IEEE TMC, ACM TKDD, ACM SIGKDD, AAAI, IJCAI, VLDB, WWW, and ACM SIGIR.