2022

Towards Automated Data Mining: Reinforcement Intelligence for Self-Optimizing Feature Engineering

Kunpeng Liu
University of Central Florida

TOWARDS AUTOMATED DATA MINING: REINFORCEMENT INTELLIGENCE FOR SELF-OPTIMIZING FEATURE ENGINEERING

by

KUNPENG LIU
B.S. University of Science and Technology of China, 2011
M.S. University of Science and Technology of China, 2015

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2022

Major Professor: Yanjie Fu

ABSTRACT

Feature engineering is one of the most important components in data mining and machine learning. One of the key thrusts in data mining is to answer: How should a low-dimensional geometry structure be extracted and reconstructed from high-dimensional data? To solve this issue, researchers proposed feature selection, PCA, sparsity regularization, factorization, embedding, and deep learning. However, existing techniques are limited in achieving full automation, globally optimal, and explainable explicitness. Can I address the automation, optimal, and explainability challenges in data geometry reconstruction? A low-dimensional data geometry structure is crucial for SciML methods (e.g., GP models), and the accuracy of these methods depends on how one can learn the data geometry structure from data or physics-based models. This dissertation will target the problem of automated identification of an optimal and explicit low-dimensional data geometry from high dimensional data. I will propose a novel principled self-optimizing data geometry reconstruction framework by viewing feature generation and selection from the lens of Reinforcement Learning (RL). I will show that reconstructing a low-dimensional data geometry (a.k.a., feature space) can be accomplished by an interactive nested feature generation and selection framework, where feature generation is to generate new meaningful and explicit features, feature selection is to subset redundant features to reduce dimensionality, and an optimized sequential structure of generations and selections will result into an optimized feature space for a downstream machine learning task. Finally, I will highlight that the search for such an optimized sequential structure can be generalized as an advanced cascading reinforcement learning system.

I dedicate this dissertation to my family.

ACKNOWLEDGMENTS

I would like to appreciate all the invaluable help I gratefully received during my Ph.D. career.

First, I would like to express my gratitude to my Ph.D. advisor, Prof. Yanjie Fu, for his strong support and continuous help. His endless passion, professional dedication, and unswerving devotion to scientific research greatly encourage me to pursue a higher research target. His critical thinking and flexible solutions to challenging problems inspire me a lot, not only in academic study but also in my daily life.

I also would like to thank the rest of my dissertation committee: Prof. Kien A. Hua, Prof. Cliff C. Zou, and Prof. Qunzhou Sun. Their professional comments on my study benefit me a lot in improving my research quality. Prof. Hua has been a great mentor in my career choice, as well as a good friend to encourage me when I was not confident. I have learned a lot from him about how to be an excellent researcher.

I would like to acknowledge all my co-authors for their collaborations with me. My internship mentors, Dr. Yirui Hu, Dr. Xiting Wang, and Dr. Jin Cao provided me with tremendous support in so many aspects. I learned more about time management and coordination in industry, which is so different from university. I thank all my fellow labmates: Pengyang Wang, Pengfei Wang, Dongjie Wang, Wei Fan, and Weijieying Ren for their mutual improvement and pure friendship.

Finally, I would like to express my deepest gratitude to my family. Their unconditional love, tacit understanding, prompt encouragement, and selfless support used to be and always will be my endless source of strength.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Motivation

High dimensionality has been a well-known challenge for data mining and machine learning. It can cause the issue of pairwise distances between samples converging to the same value, increase data sparsity, and reduce model generalization. One of the key thrusts in Data Intensive SciML is to answer: **How should a low-dimensional geometry structure be extracted from high-dimensional data?** [3] To solve this issue, researchers proposed feature selection [41, 62], principle component analysis [48, 59, 10], sparsity regularization [96, 33, 45, 22], factorization [77, 84, 34, 80], embedding [63, 36], and deep representation learning [5, 44]. However, existing techniques are limited in achieving full automation, global optimal, and explainable explicitness. **Can we address the automation, optimal, and traceable explainability challenges in geometry structure extraction?** A low-dimensional geometry structure is crucial for SciML methods (e.g., GP models), and the accuracy of these methods depends on how well one can learn the structure from data or physics-based models [3]. This dissertation will target the problem of automated identification of an optimal and explicit low-dimensional geometry from high dimensional data. I will develop a new concept: **Self-optimizing Data Geometry Reconstruction (SDGR)**. This dissertation will **develop a machine learning system for SDGR**, by blending the power of feature generation, feature selection, and reinforcement learning. In the following, I name a data geometry as a feature space, and name a dimension (variable) of the geometry as a feature.

Research Gap

To identify the research gap, we start with a brief review of three bodies of literature: 1) feature selection; 2) feature generation, 3) reinforcement learning. Each body of literature is vast. Thus, instead of aiming to be comprehensive, we focus on identifying the commonalities and key distinctions of the three areas, then investigate the critical gap in integrating them.

Existing Literature

Data geometry reconstruction includes removing variables (i.e., feature selection) and adding variables (i.e., feature generation). Feature selection is to remove redundant features and retain important features, whereas feature generation is to create and add meaningful variables. Data geometry reconstruction can be achieved by combining feature generation and selection. *Feature Selection:* Feature selection approaches include: (i) filter methods (e.g., univariate selection [105, 29], correlation based selection [43, 106]), in which features are ranked by a specific score like redundancy, relevance; (ii) wrapper methods (e.g., Evolutionary Algorithms [103, 55], Branch and Bound [79, 58]), in which the optimized feature subset is identified by a search strategy under a predictive task; (iii) embedded methods (e.g., LASSO [96], decision tree [93]), in which selection is part of the optimization objective of a predictive task. *Feature Generation:* Feature generation studies include: (i) latent representation learning based methods, e.g., deep factorization machine [39] and XDeepFM [64]. Such methods generate latent embedding feature (geometry) space, but are difficult to interpret; (ii) transformation graph based methods, e.g., traversal transformation graph [53]. Such methods generate explicit features by literately applying the same operations on each feature of the entire feature set to generate new features. These methods have two weaknesses: (a) ignore feature-feature heterogeneity and lack customized generation strategies

among different feature pairs; (b) grow exponentially when the number of steps increase, and, thus, will include many redundant features. *Reinforcement Learning:* as another relevant literature, reinforcement learning can interact with environments, generate exploration and exploitation experiences, and learn global optimized decision policies, has raised significantly interests in recent years. **Commonalities:** Reinforcement learning can conceptualize the SDGR problem in a remarkably consistent way, because either feature selection or generation is a repeated process: select/generate, test, re-select/re-generate, re-test. This is a sequential decision process. The decision of each iteration is made after observing previous states, actions, and rewards, and thus, can be viewed as a Markov Decision Process (MDP). Reinforcement learning can iterate decision (selection or generation) and reward evaluation (test results) to search the best sequential structure of generation and selection. Moreover, both feature selection/generation and reinforcement learning involve exploration and exploitation. **Key Distinctions:** Despite the commonalities, classic feature selection or generation methods cannot fully free scientists from repeatedly applying, testing and adjusting variable selection, instance filtering in data geometry reconstruction. Classic methods need a more globally-optimized, automated strategy with traceable explainability for maximum information preservation in the data geometry, while tailored for downstream SciML tasks. Prior studies of factorization, embedding, and deep learning cannot interpret the explicit semantic meaning of latent dimensionalities. Reinforcement learning can formulate nested feature generation and selection as a sequential iterative decision problem with unique strengths: global reward maximization, self optimization, and traceable decisions.

Gap in Existing Literature.

Viewing the SDGR through the lens of reinforcement learning, an obvious tension arises. Reinforcement learning provides great potential to automate feature generation and selec-

tion. In the contrast, it is challenging to: 1) develop an appropriate reinforcement model structure to unify feature generation and selection to effectively decide the dimensionalities in SDGR; 2) improve the efficiency in SDGR. These two opposite positions create a tension that represents a considerable gap hindering the convergence of the three fields. Resolving the tension and bridging the research gap require novel and fundamental research to integrate reinforcement learning, feature selection and generation, optimal geometry understanding.

Challenges and Project Aims

Below we analyze the framework challenges, algorithm challenges, and computational challenges of self-optimizing geometry reconstruction.

C1. Framework challenges. We found that leveraging both feature generation and feature selection can add and remove dimensionalities to reconstruct a new geometry space. We seek to investigate how the task structures of automated optimal geometry reconstruction are connected to learning systems. The key challenges are to find a novel principled self-optimizing framework governing how agents leverage feature generation and selection to jointly generate meaningful variables and remove redundant variables, and iteratively converge to an optimal geometry space.

C2. Algorithm challenges. After proposing a principle framework, we will focus on technical capabilities that implement the algorithmic components of the self-optimizing systems. The challenges of the algorithmic components include: which operations can help to reconstruct a low-dimensional geometry? who will decide and coordinate these operations? how to ensure a series of operations will convert a data set to an optimal geometry? Since agents need perception capability to perceive situational states, how can we better represent the state of the environment? Since agents learn from trial and error experiences, how can we better

measure a reward function?

C3. Computational challenges. After instantiating the machine learning framework, we will focus on improving its training efficiency and effectiveness. The major challenges include: 1) how can we improve training efficiency so that agents can explore and learn faster? 3) how can we improve scalability?

Organization

Therefore, to bridge the gap between interpretable feature engineering and automated feature engineering, I propose to study the problem of how to develop general frameworks for automatically feature selection and automated feature generation.

Specifically, my dissertation can be categorized into three themes:

- Theme 1: Effective Automated Feature Selection via Multi-Agent Reinforcement Learning. In this theme, I discuss how to effectively automate the classic feature selection problem by a multi-agent reinforcement learning framework. Effective feature selection can help to reduce dimensionality, shorten training times, enhance generalization, avoid overfitting, improve predictive performance, and provide better interpretation and explanation. Existing studies either required expertise in feature selection or were not effective enough in exploring large feature spaces. Hence, it is desirable to develop approaches to automatically and effectively solve the feature selection problem.
- Theme 2: Efficient Automated Feature Selection via Single-Agent Reinforcement Learning. In this theme, I discuss how to efficiently automate the classic feature selection problem by a single-agent reinforcement learning framework. Although the

multi-agent reinforcement learning in Theme 1 enjoys the automation and effectiveness of the selection process, it suffers from not just the data complexity in terms of contents and dimensionality, but also the exponentially-increasing computational costs with regard to the number of agents. Therefore, it is essential to simplify the selection process of agents under reinforcement learning context so as to improve the efficiency and reduce the cost.

- Theme 3: Automated Feature Generation via Reinforcement Learning. In this theme, I discuss how to automatically generate features via reinforcement learning. Traditional hand-crafted feature generation requires human intuition and domain knowledge, therefore it is appealing to develop automated feature generation methods for the convenience of beginners and researchers outside of machine learning area. Existing studies on automated feature generation either generated implicit features with limited interpretation, or applied the same operation on the whole feature set, ignoring the difference between features.

All the above three themes will inherently handle the automated feature engineering inside the proposed frameworks. In the following chapters, I first study the problem of automated feature selection by proposing one agent for each feature to control their selection in Chapter 2. Next, I conduct research to study the problem of computationally efficient feature selection by proposing one feature agent to control the selection of all features in Chapter 3. Then, I propose to interpretable automated feature generation in Chapter 4. Finally, I conclude in Chapter 5 with a discussion of potential future research topics.

CHAPTER 2: MULTI_AGENT REINFORCEMENT FEATURE SELECTION

In this chapter, I focus on developing a new multi-agent reinforcement learning based feature selection method. This method can automatically select the optimal feature subset.

Introduction

Feature selection aims to select an optimal subset of relevant features for a downstream predictive task [11, 106]. Effective feature selection can help to reduce dimensionality, shorten training times, enhance generalization, avoid overfitting, improve predictive accuracy, and provide better interpretation and explanation. In this chapter, we study the problem of automated feature subspace exploration for improving downstream predictive tasks.

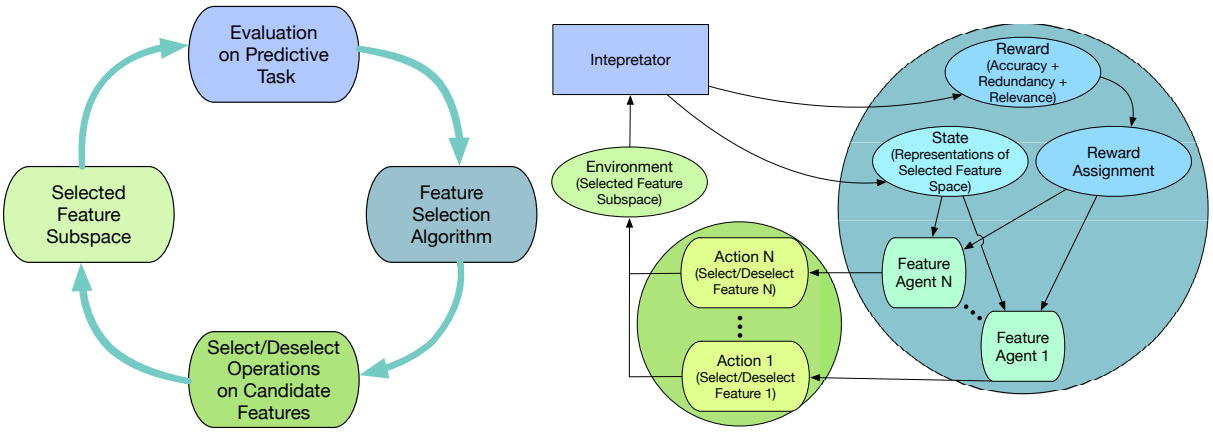
Prior studies in feature selection can be grouped into three categories: (i) filter methods (e.g., univariate feature selection [105, 29], correlation based feature selection [43, 106]), in which features are ranked by a specific score; (ii) wrapper methods (e.g., evolutionary algorithms [103, 55], branch and bound algorithms [79, 58]), in which optimal feature subset is identified by a search strategy that collaborates with predictive tasks; (iii) embedded methods (e.g., LASSO [96], decision tree [93]), in which feature selection is part of the optimization objective of predictive tasks. 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 search a very large feature space of 2^N feature subspace candidates, where N is the feature number. Embedded methods are subject to the strong structured assumptions of predictive models. As can be

seen, feature selection is a complicated process that requires (i) strategic design of feature significance measurement, (ii) accelerated search of near-optimized feature subset, and (iii) meaningful integration of predictive models.

Reinforcement learning can interact with environments, learn from action rewards, balance exploitation and exploration, and search for long-term optimal decisions [109, 66]. These traits provide great potential to automate feature subspace exploration. Existing studies [27, 60] create a single agent to make decisions. In these models, the single agent has to determine the selection or deselection of all N features. In other words, the action space of this agent is 2^N. Such formulation is similar to the evolutionary algorithms [103, 55, 30], which are NP-hard and can merely obtain local optima. In this chapter, we intend to propose a better solution using reinforcement learning for feature selection. However, several challenges arise toward this goal.

First, how can we reformulate the problem so that the action space could be limited? We reformulate the problem with multi-agent reinforcement learning. We assign an agent to each feature, the actions of these feature agents are to select or deselect their corresponding features, and the state of environment is characteristics of the selected feature subspace. One key challenge in multi-agent learning is to coordinate the interactions between agents. The interactions can be from two aspects: (i) cooperation and (ii) competition between agents, which can be quantified by feature-feature mutual information in our case. We propose to integrate feature-feature mutual information with predictive accuracy as the reward scheme. In this way, we guide the cooperation and competition between agents for effective feature exploration.

Second, how can we accurately describe the state representation in multi-agent reinforcement learning? We regard the selected feature subspace as the state of environment. To construct



(a) General Process of Feature Selection (b) Multi-Agent Reinforcement Learning on Feature Selection

Figure 2.1: From traditional feature selection to multi-agent reinforcement learning based feature subspace exploration.

state representations, traditional methods are to extract descriptive statistics (e.g., mean, variance) of the state distribution. However, in feature subspace exploration, the number of selected features changes over time during the exploratory process. If we also extract the mean and variance of each selected feature to describe the state, length of the state representation vector will change over time. But, the policy networks in multi-agent reinforcement learning require a fixed-length state representation. To tackle this challenge, we develop three different methods: (i) meta descriptive statistics, (ii) auto-encoder based deep representation, (iii) dynamic graph based graph convolutional network (GCN). In Method i, after extracting the first round descriptive statistics of each selected feature, we extract the second-round descriptive statistics of these first-round descriptive statistics as the state representation vector, so that the state length will not change along with the varying number of selected features. In Method ii, since the number of rows are static in the selected feature subspace, we construct a row-row similarity graph, namely static subspace graph, to describe the state. An autoencoder method is applied to learn the state representation. In Method iii, we construct a feature-feature similarity graph to describe the state. Since nodes are features, the number of nodes changes over time. We exploit GCN to learn state

representations from dynamic graphs.

Third, how can we improve the robustness of our framework against a vastly different state distribution, while accelerating the exploration of optimal features? Traditionally, we can use experience replay [68, 78] to train our multi-agent framework. In the experience replay, an agent takes samples from the agent's memory that stores different types of training samples to train the model. In automatic control area, reinforcement learning usually considers all of the samples in the memory, because all possible states need to be evaluated. However, in feature selection, noise, outliers, or low-rewarded data samples can lead to inaccurate understanding of a feature and feature-feature correlations, and, thus, jeopardize the accuracy of feature selection. Can we create a sampling strategy to select sufficient high-quality samples and avoid low-quality samples? An intuitive method is to oversample high-quality samples by increasing their sampling probabilities. But, this method can not guarantee the independences of samples between different training steps, as it is equivalent to reduce the memory size. To address this issue, we develop a gaussian mixture model (GMM) based generative rectified sampling strategy. Specifically, we first train a GMM with high-quality samples. The trained GMM is then used to generate a sufficient number of independent samples from different mixture distribution components for reinforcement learning.

Additionally, we find that there exist three limitations in the GMM-based sampling strategy: i) The Gaussian mixture model may not be the perfect model to fit sample's distribution; ii) The fitting of GMM is costly; iii) Noise may pollute samples. To address these issues, in this extended version, we develop a softmax based sampling strategy. Specifically, we first rank the samples by their reward, and then derive their priority by their inverse rank. The sampling probability is derived by the softmax of priority thereafter. In the exploration strategy aspect, reinforcement learning agent explores the environment and learns from the reward. With more and more experience accumulated, the agent can find a more and more promising

exploration direction. This exploration strategy is simple and general, which can be easily adapted to almost every reinforcement learning problems. However, when the state space is extremely large, its exploration efficiency would be rather low. To reduce the exploration space, we introduce interactive reinforcement learning (IRL) [17, 18]. In IRL, a pre-trained naive feature selection plays the role of 'advisor' to guide the reinforcement learning feature selection algorithm to quickly pass its apprenticeship period. Specifically, we firstly derive a feature subset $\mathcal{S}^K$ via a naive feature Selection algorithm. In the apprenticeship steps, we randomly choose half of the features in $\mathcal{S}^K$ to add them in the selected feature subset. Through this addition, the state representation is changed and thus guides the reinforcement learning to a better exploration direction. After the apprenticeship period, the multi-agent reinforcement learning leaves $\mathcal{S}^K$ and does feature selection independently.

In summary, in this chapter, we develop an enhanced multi-agent reinforcement learning framework for feature subspace exploration. Specifically, our contributions are as follows: (1) We reformulate feature selection problem with a multi-agent reinforcement learning framework and design a new reward scheme to guide the cooperation and competition between agents. (2) We develop three different methods: meta descriptive statistics, autoencoder based deep representation, and dynamic graph based graph convolutional network (GCN), to derive accurate state representation. (3) We develop three different strategies: GMM-based generative rectified sampling strategy, rank-based softmax sampling strategy and IRL-based exploration strategy to improve the training and exploration. (4) We conduct extensive experiments to demonstrate the enhanced performances of our methods.

Problem Formulation

We study the problem of feature subspace exploration, which is formulated as a multi-agent reinforcement learning task. Figure 2.1 shows an overview of our proposed multi-agent reinforcement learning based feature exploration framework. Given a set of features to be explored, we first create a feature agent for each feature. This feature agent is to decide whether its associated feature is selected or not. The selected feature subset is regarded as the environment, in which feature agents interact with each other. The correlations between features are schemed by reward assignment. Specifically, the components in our multi-agent reinforcement learning framework includes agents, state, environment, reward, reward assignment strategy, and agent actions.

Multi-Agent. Assuming there are N features, we define N agents for the N features. For one agent, it is designed to make the selection decision for the corresponding feature.

Actions. For the i-th feature agent, the feature action $a_i = 1$ indicates the i-th feature is selected, and $a_i = 0$ indicates the i-th feature is deselected.

Environment. In our design, the environment is the feature subspace, representing a selected feature subset. Whenever a feature agent issue an action to select or deselect a feature, the state of feature subspace (environment) changes.

State. The state s is to describe the selected feature subset. To extract the representation of s, we explore three different strategies, i.e., meta descriptive statistics, autoencoder based deep representation and dynamic graph based graph convolutional network (GCN). We will elaborate these three state representation techniques in Section 2.

Reward. We design a measurement to quantify the overall reward R generated by the

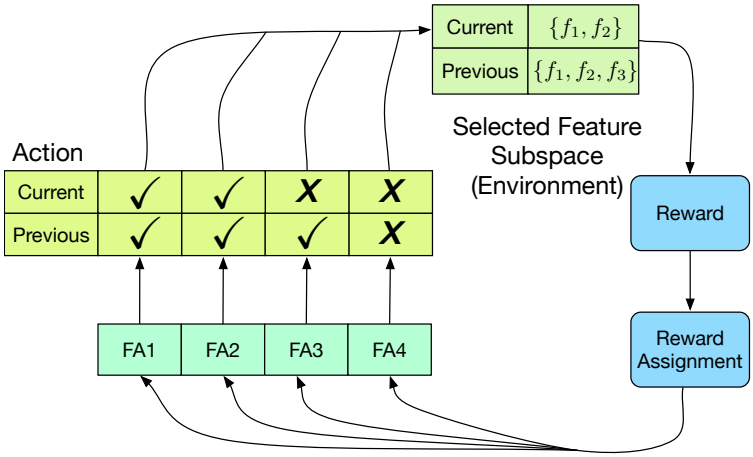


Figure 2.2: The demonstration of reward assignment process. The feature agent 1 and 2 issue an action to select the feature 1 and feature 2. The feature agent 3 issues an action to disselect the feature 3.

selected feature subset, which is defined the weighted sum of (i) predictive accuracy of the selected feature subset Acc, (ii) redundancy of the selected feature subset Rv, and (iii) relevance of the selected feature subset Rd.

Reward Assignment Strategy. We develop a strategy to allocate the overall reward to each feature agent. The assignment of the overall reward to each agent, indeed, shows the coordination and competition relationship among agents. In principle, we should recognize and reward all of the participated feature agents. Figure 2.2 shows an example of reward assignment. There are four features with four corresponding feature agents. In the previous iteration, the feature 1, 2, 3 are selected, and the feature 4 is unselected. In the current iteration, feature agent 1 and feature agent 2 issue actions to select feature 1 and feature 2; feature agent 3 issues an action to deselect feature 3; feature agent 4 does not participate and issue any action to change the status of feature 4. In summary, there are only three feature agents (FA1, FA2, FA3) that participate and issue actions. Therefore, the current reward R is equally shared by these three agents.

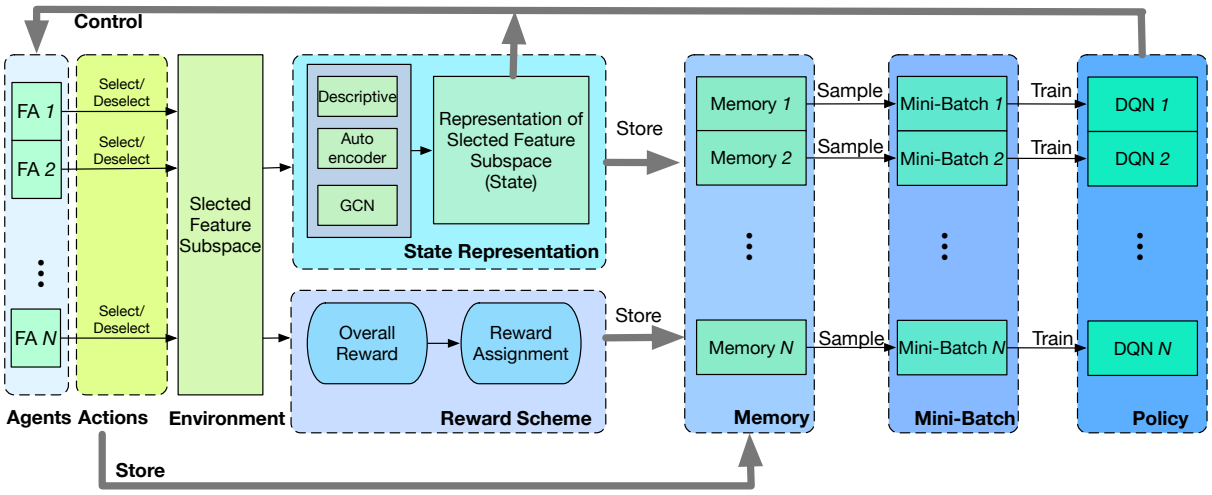


Figure 2.3: Framework. The framework consists of two stages. In the control stage, feature agents select or drop their corresponding features based on policies. In the training stage, the policies are trained via samples from memories.

Proposed Method

We first present the multi-agent reinforcement learning framework for automated feature subspace exploration. Later, we discuss how to measure reward, how to improve state representation, and how to accelerate feature subspace exploration.

Framework Overview

Figure 2.3 shows our proposed framework consists of many feature subspace exploration steps. Each exploration step includes two stages, i.e., control stage and training stage.

In the control stage, each feature agent takes actions based on their policy networks, which take current state as input and output recommended actions and next state. The select/deselect actions of each feature agent will change the size and contents of the selected feature subset, and thus, lead to a new selected feature subspace. We regard the selected feature subset as environment. The state represents the statistical characteristics of the

selected feature subspace. We derive a comprehensive representations of the state through three different methods, i.e., descriptive statistics, autoencoder and GCN (refer to Section 2). Meanwhile, the actions taken by feature agents generate an overall reward. This reward will then be assigned to each of the participating agents.

In the training stage, agents train their policy via experience replay independently. For agent i, at time t, a newly-created tuple $\{s_i^t, a_i^t, r_i^t, s_i^{t+1}\}$, including the state ($s_i^t$), the action ($a_i^t$), the reward ($r_i^t$) and the next state ($s_i^{t+1}$), is stored into each agent's memory. We then propose a GMM-based generative rectified sampling (refer to Section 2) to derive mini-batches from memories. The agent i uses its corresponding mini-batch samples to train its Deep Q-Network (DQN), in order to obtain the maximum long-term reward based on the Bellman Equation [94]:

$$Q(s_i^t, a_i^t|\theta_t) = r_i^t + \gamma \max Q(s_i^{t+1}, a_i^{t+1}|\theta_{t+1}) \tag{2.1}$$

where θ is the parameter set of Q network, and γ is the discount.

The exploration of feature subspace continues until convergence or meeting several predefined criteria.

Measuring Reward

We propose to combine the predictive accuracy Acc, the feature subspace relevance Rv, and the feature subspace redundancy Rd as the reward R of actions.

Predictive Accuracy. Our goal is to explore and identify a satisfactory feature subset, which will be used to train a predictive model in a downstream task, such as classification and outlier detection. We propose to use the accuracy Acc of the predictive model to quantify

the reward. Specifically, if the predictive accuracy is high, the actions that produce the selected feature subset should receive a high reward; if the predictive accuracy is low, the actions that produce the selected feature subset should receive low rewards.

Feature Subspace Characteristics. Aside from exploiting the predictive accuracy as reward, we propose to take into account the characteristics of the selected feature subset. Specifically, a qualified feature subset is usually of low information redundancy and of high information relevance to the predictive labels (responses). Both the information relevance and redundancy can be quantified by the mutual information, denoted by I. Formally, I by:

$$\boldsymbol{I}(\boldsymbol{x};\boldsymbol{y}) = \sum_{i,j} p(x_i, y_j) log(\frac{p(x_i, y_j)}{p(x_i)p(y_j)}) \tag{2.2}$$

where x_i, y_i is the i-th and j-th feature, $p(\boldsymbol{x}, \boldsymbol{y})$ is the joint distribution of $\boldsymbol{x}$ and $\boldsymbol{y}$, while $p(\boldsymbol{x})$ and $p(\boldsymbol{y})$ are marginal distribution of $\boldsymbol{x}$ and $\boldsymbol{y}$.

- The *information redundancy* of a feature subset, denoted by Rd, can be quantified by the sum of pairwise mutual information among features. Formally, Rd is given by:

$$Rd = \frac{1}{|\boldsymbol{S}|^2} \sum_{\boldsymbol{x_i}, \boldsymbol{x_j} \in \boldsymbol{S}} \boldsymbol{I}(\boldsymbol{x_i}; \boldsymbol{x_j}) \tag{2.3}$$

  where $\boldsymbol{S}$ is the feature subset, $\boldsymbol{x_i}$ is the i-th feature,

- The *information relevance* of a feature subset, denoted by Rv, can be quantified by the mutual information between features and labels. Formally, Rv is given by:

$$Rv = \frac{1}{|\boldsymbol{S}|} \sum_{\boldsymbol{x_i} \in \boldsymbol{S}} I(\boldsymbol{x_i}; \boldsymbol{c}) \tag{2.4}$$

  where $\boldsymbol{c}$ is the label vector.

Improving State Representation

Assuming there is a $M * N$ dataset $\boldsymbol{D}$, which includes M data samples and N features. Let n_j be the number of selected features at the j-th exploration step. Then, $M * n_j$ is the dimension of the selected data matrix $\boldsymbol{S}$, which varies over exploration steps. However, the policy network and target network in DQN require the state representation vector $\boldsymbol{s}$ to be a fixed-length vector at each exploration step. We thus, need to derive a fixed-length state vector $\boldsymbol{s}$ from the selected data matrix $\boldsymbol{S}$, whose dimensions change over time.

To derive accurate state representation with fixed length, we develop three different methods, including (i) meta descriptive statistics of feature subspace; (ii) static subspace graphs based autoencoder; (iii) dynamic feature-feature similarity graphs based graph convolutional network (GCN). The commonness between these three methods is that they all first learn representations for each feature, and then aggregate them to get a state representation. The differences between them lie on the representation learning algorithms and aggregation strategies.

Method 1: Meta Descriptive Statistics of Feature Subspace. Figure 2.4 shows how we extract the meta data of descriptive statistics from the selected data matrix through 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 $\mathbf{7} * 7$.

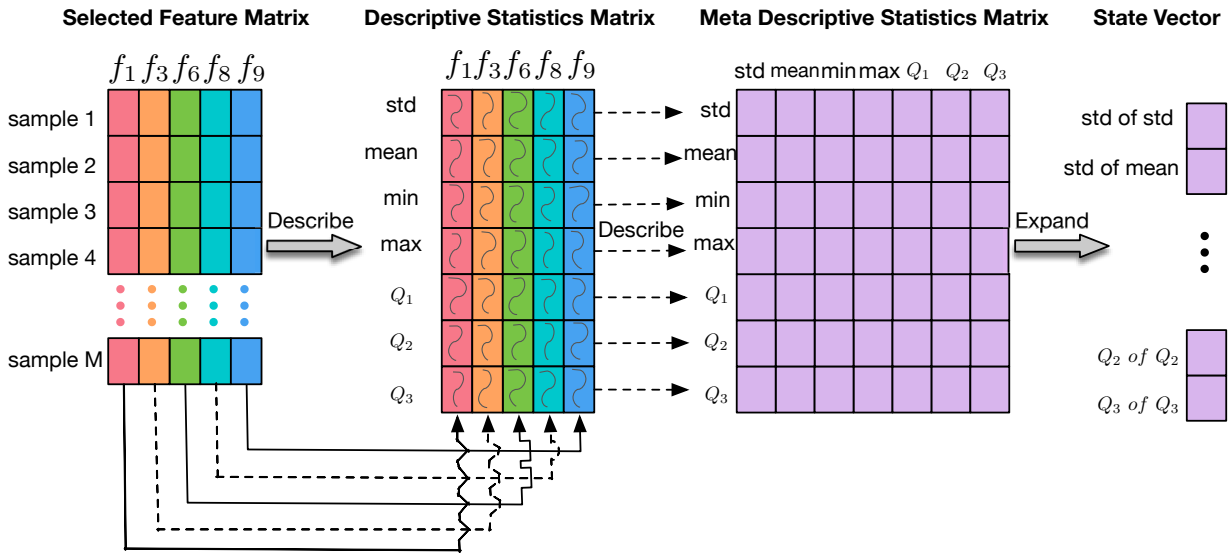


Figure 2.4: Meta descriptive statistics. We extract descriptive statistics twice from the feature subspace to obtain a fixed-length state vector.

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

Method 2: Autoencoder Based Deep Representation of Feature Subspace. Autoencoder has been widely used for representation learning by minimizing the reconstruction loss between an original input and a reconstructed output [6]. An autoencoder contains an encoder that maps the input into a latent representation, and an decoder that reconstructs the original input based on the latent representation.

Figure 2.5 shows a two-step algorithm to learn the state vector.

Step 1: Assuming at the j-th exploration step, $\boldsymbol{S}$ is the selected data matrix, and n_j is the number of selected features. For each feature (**column**) in $\boldsymbol{S}$, we apply an autoencoder to convert each feature column into a k-length latent vector, and thus, obtain a *latent matrix* $\boldsymbol{L}$ with a dimension of $k * n_j$. However, $\boldsymbol{L}$ cannot represent the state, because the size of $\boldsymbol{L}$ is not static and still varies over number of selected features n_j at the j-th exploration step.

Step 2: We apply another auto-encoder to map each *row* of $\boldsymbol{L}$ into a o-length latent vector, and obtain a *static encoded matrix* $\boldsymbol{L}'$ with a fixed dimension of $k * o$.

Finally, we link each column in $\boldsymbol{L}'$ together into the state vector $\boldsymbol{s}$ with a fixed length of $k * o$.

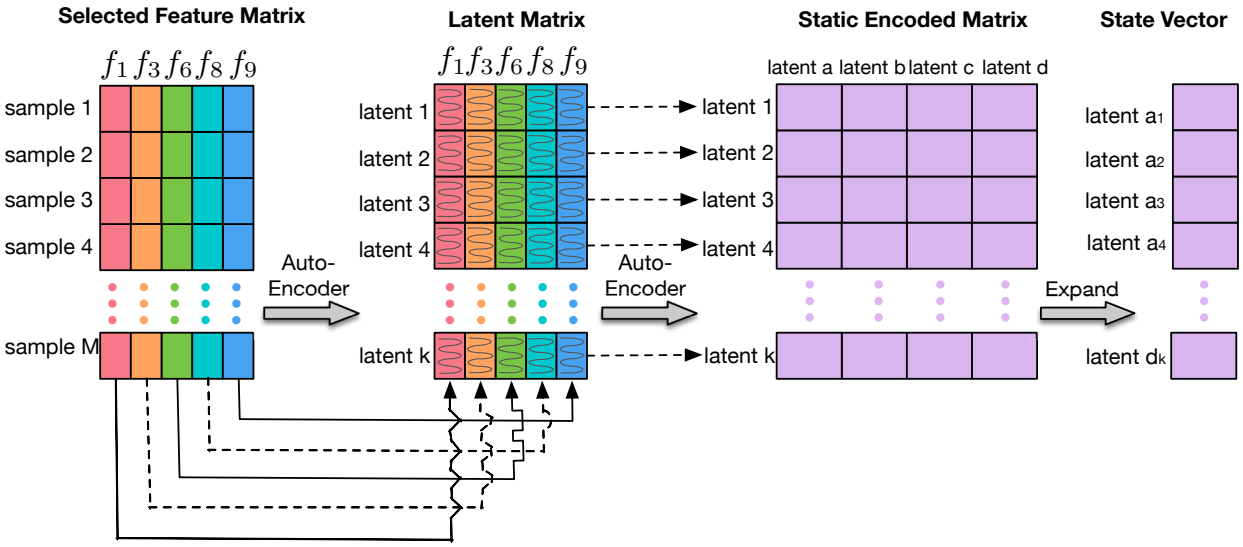


Figure 2.5: Autoencoder based deep representations. We use two auto-encoders to map the feature subspace into a fixed-length state vector.

Method 3: Dynamic-Graph Based Graph Convolutional Network (GCN). Method 1 and method 2 extract explicit and latent representations of each feature. In this method, we consider not just a feature's individual representations, but also the correlations among features. Figure 2.6 shows how the GCN works. To better capture the relationship among features, we first convert the selected data matrix $\boldsymbol{S}$ into a dynamic complete graph $\mathbf{G}$, where a node is a feature column in $\boldsymbol{S}$. With the *feature correlation graph* $\mathbf{G}$, any graph node embedding techniques could be used for node latent representation by exploiting the correlation among features. As the focus of this chapter is not to design more sophisticated node embedding models, we choose GCN as it is a state-of-the-art graph embedding models

and shows competing effectiveness in many graph based tasks.

Let $\boldsymbol{S}$ be the selected data matrix with a dimension of $M*N$, $\boldsymbol{Z}$ be the representation matrix of nodes (features) with a dimension of $k*N$, k is the length of updated representation. The neural network layer in GCN is given by:

$$\boldsymbol{H}^{(l+1)} = f(\boldsymbol{H}^{(l)}, \boldsymbol{A}) \tag{2.5}$$

where $\boldsymbol{H}^{(0)} = \boldsymbol{S}$, $\boldsymbol{H}^{(L)} = \boldsymbol{Z}$, L is the layer number, $\boldsymbol{A}$ is the adjacency matrix of graph G. The regular GCN can be reduced into a simplified version by considering the node's own representation (rather than merely the neighbor structures) and performing symmetric normalization [56]:

$$f(\boldsymbol{H}^{(l)}, \boldsymbol{A}) = \sigma(\hat{\boldsymbol{D}}^{-\frac{1}{2}}\hat{A}\hat{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}), \tag{2.6}$$

where $\hat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ with I being an identity matrix, $\hat{\boldsymbol{D}}$ is the diagonal node degree matrix of $\hat{\boldsymbol{A}}$.

By solving GCN, we obtain the latent representations $\boldsymbol{Z}$ of each feature. We average the representation of each feature into the k-legnth state representation vector.

Accelerating Feature Subspace Exploration via Generative Rectified Sampling

Experience replay is widely used to improve training efficiency of neural networks in reinforcement learning [68, 78]. After taking each action, the latest sample, in the form of a tuple that consists of the action (a), the reward (r), the state (s) and the next state ($s^{'}$), is stored into the memory to replace the oldest sample. In training step, a mini-batch of samples are picked to update the policy network.

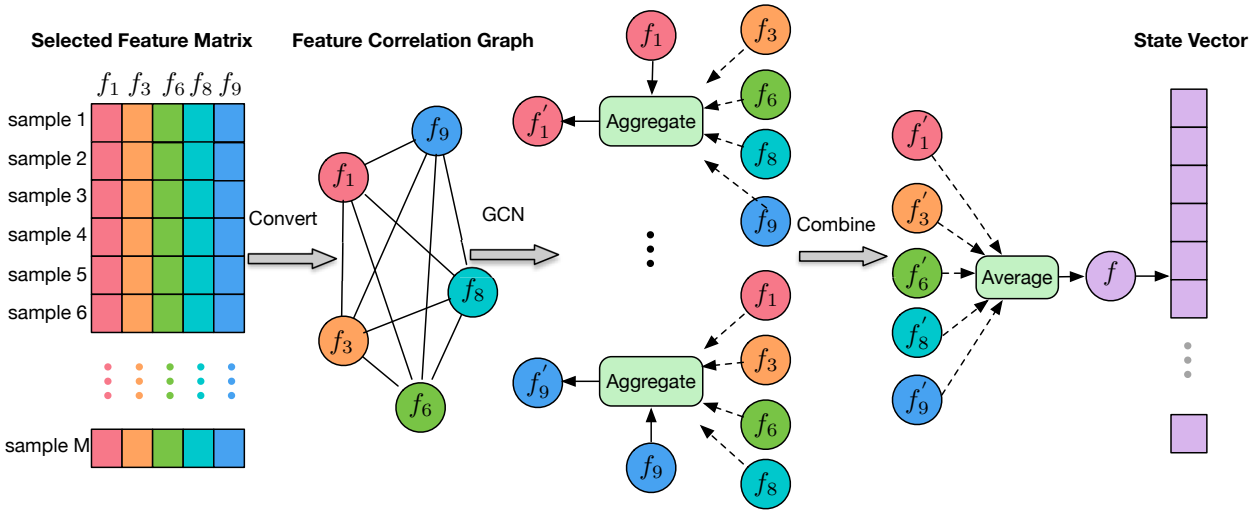


Figure 2.6: Dynamic-graph based GCN. We denote the feature subspace by a dynamic graph and use GCN to update representations of each node.

In the task of feature subspace exploration, we are particularly interested in exploiting high-quality samples to accelerate the exploration speed. Prior studies tackle this problem by increasing the sampling probabilities of high-quality samples [87, 101]. However, such strategy create a new problem: the sampler repeatedly select a very limited number of high-quality samples. Consequently, prior studies can not guarantee the independence of selected samples between different training steps, and can not cover a comprehensive space in the unknown high-quality sample population.

To deal with this problem, we propose a Gaussian mixture model (GMM) based generative rectified sampling algorithm. For each agent, as shown in algorithm 1, we take a set of memory samples $\boldsymbol{T} = \{< a, r, s, s^{'} >\}$ as inputs. We firstly cluster the memory samples into two groups: $\boldsymbol{T_0}$ and $\boldsymbol{T_1}$. Samples with the selected action ($a = 0$) are assigned to group $\boldsymbol{T_0}$, while samples with the deselected action ($a = 1$) are assigned to group $\boldsymbol{T_1}$. Later, we rank the memory samples in $\boldsymbol{T}$ in terms of reward (r) and select the top p proportion of high-reward samples in each group as high-quality samples. The selected high-quality

samples are then used to train two GMM based generative models for their corresponding groups via an expectation maximization (EM) algorithm [19]. After that, for each group, we use its corresponding well-trained GMM model to generate simulated samples to replace the $1-p$ proportion of low-reward samples in the corresponding group. In this way, we create two high-reward, large-size, yet independent memory sample sets for the select-action and deselect-action groups. We combine the two simulated memory sample sets into a new high-quality dataset. The agent will take a mini-batch of samples from the new high-quality dataset for accelerating training.

Algorithm 1: The GMM-Based Generative Rectified Sampling Algorithm

Input : Memory dataset $\boldsymbol{T}$.
Output: A mini-batch of samples $\boldsymbol{B}$.
$p \leftarrow$ high-quality sample proportion of $\boldsymbol{T}$.
Stratify $\boldsymbol{T}$ into two groups. Samples with $a = 0$ are assigned to group $\boldsymbol{T_0}$ and samples with $a = 1$ are assigned to group $\boldsymbol{T_1}$.
for $i = 0$ *to* 1 **do**
    $N_i \leftarrow$ sample number of $\boldsymbol{T_i}$.
    $K_i \leftarrow$ component number of GMM model $\mathcal{G}^i$.
    Rank samples in $\boldsymbol{T_i}$ by their reward r, then select top $N_i * p$ samples from $\boldsymbol{T_i}$ to form the high-quality dataset $\boldsymbol{H_i}$.
    Use $\boldsymbol{H_i}$ to train the GMM $\mathcal{G}^i = \sum_1^{K_i} \phi_i \mathcal{N}(\mu_i, \Sigma_i)$ via EM algorithm.
    Generate $N_i * (1-p)$ samples from $\boldsymbol{G}^i$ to form the generated dataset $\boldsymbol{G_i}$.
    Join $\boldsymbol{H_i}$ and $\boldsymbol{G_i}$ to create high-quality dataset of action i, $\boldsymbol{T_i'}$.
end
Join $\boldsymbol{T_0'}$ and $\boldsymbol{T_1'}$ to get high-quality dataset $\boldsymbol{T'}$.
Sample a mini-batch of samples $\boldsymbol{B}$ from $\boldsymbol{T'}$.

Accelerating Feature Subspace Exploration via Rank-Based Softmax Sampling

The GMM-based generative rectified sampling strategy can make full use of the samples in experience replay, thus accelerate the training process of reinforcement learning policy. However, it naturally has three potential drawbacks: 1) It is based on an assumption that the sample are generated by a Gaussian Mixture Model, while their actual distribution could

be different; 2) The fitting of GMM model is computationally expensive. To make things worse, it needs to fit the GMM model every time it samples; 3) There could exist noise in the samples, which affects the fitting accuracy of GMM model. Here we have one question: can we propose a more simple sampling strategy yet effective than the sampling strategy in 2, which could have lower computational burden and higher robustness?

To tackle these problems, we introduce a rank-based softmax sampling strategy. In this sampling strategy, we measure the importance of each samples by their ranks in the experience replay memory. The sampling probability for each sample is then designed based on their ranks:

$$P(i) = \frac{exp(p_i)}{\sum_{n=1}^{N_E} exp(p_n)} \tag{2.7}$$

where N_E is the size of experience replay memory, p_i is the priority of i-th sample, and we make $p_i = \frac{1}{rank(i)}$, where $rank(i)$ is the rank of sample i based on its reward. The softmax operation promises the sum of all probabilities equals 1. Since $rank(i)$ is a relative value, it has high tolerance of noise and could be very robust. There is no assumption of GMM distribution, thus there is no need of fitting, making the computational burden very low. Specifically, as algorithm 2 shows, we firstly derive the rank for each sample, and then obtain their sampling probabilities by Equation 2.7. The agent will take a mini-batch of samples based on the rank-based sampling probabilities. Compared with the GMM-based generative rectified sampling strategy, it is efficient due to the low computational burden, and effective due to the robustness on noise.

Accelerating Feature Subspace Exploration via Interactive Reinforcement Learning

In the classic reinforcement learning framework, the agent repeatedly explores the state space and gets reward, after which it gets more and more experience and behaves better

Algorithm 2: The Softmax Sampling Algorithm

Input : Memory dataset $\boldsymbol{T}$.
Output: A mini-batch of samples $\boldsymbol{B}$.
$N_E \leftarrow$ size of experience replay memory $\boldsymbol{T}$.
Rank samples in $\boldsymbol{T}$ by their reward, and let $p_i = \frac{1}{rank(i)}$ be their priority.
Derive the sampling probability for each data sample by Equation 2.7.
Sample a mini-batch of samples $\boldsymbol{B}$ from $\boldsymbol{T}$.

and better. This exploration strategy is general and universal, meaning that it can be applied to any formulated reinforcement learning problems. However, in our case, the state space is extremely large, and if we simply adapt the conventional exploration strategy, the exploration efficiency would be rather low. Here we have one question: Can we propose a more advanced exploration strategy, which could explore along a more promising direction, so that the feature space exploration process would be accelerated?

The proposed multi-agent reinforcement learning framework improves itself step by step, and it has a apprenticeship period in the beginning when its performance is very bad. To reduce its exploration space, we introduce interactive reinforcement learning (IRL) [17, 18]. In IRL, a naive feature selection method, i.e., K-Best Selection [105], works as the 'advisor' to guide reinforcement learning to explore along a relatively good direction. After pre-defined steps, the reinforcement learning abandons the advisor and explore the state space independently.

Specifically, as algorithm 3 shows, we firstly derive a feature subset $\mathcal{S}^K$ via K-Best Selection. In the apprenticeship steps, we randomly choose half of the features in $\mathcal{S}^K$ to add them to the selected feature subset. Through this addition, the state representation is changed and thus guides the reinforcement learning to a better exploration direction. The reason we don't use all of the features in $\mathcal{S}^K$ every step is to avoid over-fitting, and to keep the feature selection process different from the K-Best Selection. After the apprenticeship period, the multi-agent reinforcement learning would do feature selection independently.

Algorithm 3: The Interactive Reinforcement Learning Enhanced Exploration Strategy

Input : Feature number K, apprenticeship step N_A, overall step N_O, feature set $\mathcal{S}$.
Output: Optimal feature subset $\mathcal{S}'$.
Derive a feature subset $\mathcal{S}^K$ via K-Best Selection.
Randomly initialize selected feature subset $\mathcal{S}'_0$.
for $i = 1$ *to* N_A **do**
    Derive a selected feature subset $\mathcal{S}'_i$ by multi-agent reinforcement learning feature selection. (*Note: This step relies on the selected feature subset* $\mathcal{S}'_{i-1}$ *from the last step, as* $\mathcal{S}'_{i-1}$ *decides the state representation.*)
    Randomly choose $K/2$ features from $\mathcal{S}^K$, denoted as $\mathcal{S}_i^{K/2}$.
    Let $\mathcal{S}'_i = \mathcal{S}'_i + \mathcal{S}_i^{K/2}$.
end
for $i = N_A + 1$ *to* N_O **do**
    Derive a selected feature subset $\mathcal{S}'_i$ by multi-agent reinforcement learning feature selection.
end

Experimental Results

We evaluate the proposed method in feature selection with real-world data.

Data Description

The experiments are conducted on a publicly available cartographic dataset from Kaggle [8]. There are 15120 samples with 54 features that describe the characteristics of different wilderness areas. The class labels are categorical values that range from 1 to 7, and represent seven forest cover types (the predominant kind of tree cover) in the areas. Among the 54 features, 10 of them are continuous, and the remaining 44 are categorical.

Evaluation Metrics

To show the effectiveness of the proposed method, we use the following metrics for evaluation.

Overall Accuracy is the ratio of number of correct predictions to number of all predictions. Formally, the overall accuracy is given by $\frac{TP+TN}{TP+TN+FP+FN}$, where TP, TN, FP, FN are true positive, true negative, false positive and false negative for all classes. We use this metric to measure the accuracy of a classifier on test dataset for all cover types. The latter three metrics measure classification performance of each cover type from different aspects.

Precision is given by $\frac{TP_k}{TP_k+FP_k}$ which represents the ratio of true positive to true positive plus false positive with respect to the k-th ($k \in [1, 7]$) type of cover.

Recall is given by $\frac{TP_k}{TP_k+FN_k}$ which represents the ratio of true positive to true positive plus false negative with respect to the k-th ($k \in [1, 7]$) type of cover.

F-measure considers both precision and recall in a single metric by taking their harmonic mean. Formally, F-measure is given by $2 * P * R/(P + R)$, where P and R are precision and recall respectively.

Baseline Algorithms

We compare performance of our proposed Multi-Agent Reinforcement Learning Feature Selection (MARLFS) against the following six baseline algorithms, where K-Best Selection and mRMR belong to filter methods; LASSO and Recursive Feature Elimination (RFE) belong to embedded methods; Genetic Feature Selection (GFS) and Single-Agent Reinforcement Learning Feature Selection (SARLFS) belong to wrapper methods.

(1) K-Best Selection. The K-Best Selection [105] firstly ranks features by their χ^2 scores with the target vector (label vector), and then selects the K highest scoring features. In the experiments, we make K equal to the number of selected features in MARLFS.

(2) mRMR. The mRMR [81] firstly ranks features by minimizing feature's redundancy, while maximizing their relevance with the target vector (label vector), and then selects the K highest ranking features. In the experiments, we make K equal to the number of selected features in MARLFS.

(3) LASSO. LASSO [96] conducts feature selection and shrinkage via l_1 penalty, which drops the feature variables whose coefficients are 0. The hyper parameter in LASSO is its regularization weight λ, which is set to 1.0 in the experiments.

(4) Recursive Feature Elimination (RFE). RFE [37] selects features by recursively selecting smaller and smaller feature subsets. Firstly, the predictor is trained by all features and the importance of each feature are scored by the predictor. After that, the least important features are deselected. This procedure process recursively until the desired number of features are selected. In the experiments, we set the selected feature number half of the feature space.

(5) Genetic Feature Selection (GFS). Genetic Feature Selection [61] selects features by firstly calculating the fitness level for each feature and then generates better feature subsets via crossover and mutation. In the experiments, we set crossover probability to 0.5, mutation probability to 0.2, crossover independent probability to 0.5 and mutation independent probability to 0.05.

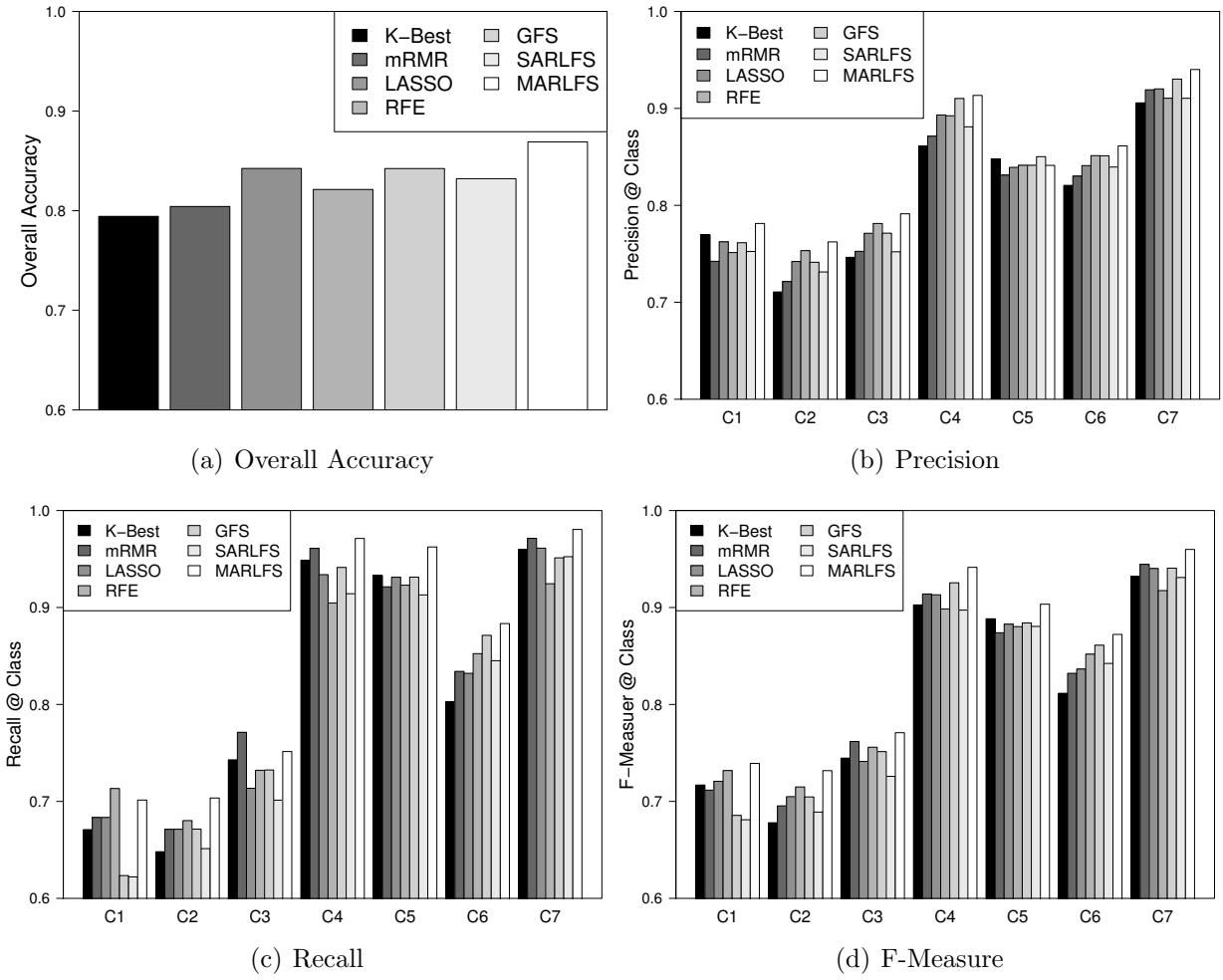
(6) Single-Agent Reinforcement Learning Feature Selection (SARLFS). The agent learns a KWIK (Knows What It Knows) model in SARLFS [60], which is represented by a dynamic Bayesian network, deduces a minimal feature set from this network, and computes a policy on this feature subset using dynamic programming methods. In the experiments, the two accuracy thresholds in the KWIK are set to $\epsilon = 0.15$, $\delta = 0.10$.

Overall Performance

We compare our method MARLFS with baseline methods in terms of overall accuracy as well as precision, recall and F-measure of the seven classes on the real-world data. In the experiments, for all deep networks, we set mini-batch size to 32 and use AdamOptimizer with a learning rate of 0.01. For all experience replays, we set memory size to 2000. 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 high-quality proportion in GMM sampling is 0.20. Unless specified, we use GCN method as the representation learning algorithm in the experiments, whose network is a two-layer ReLU with 128 and 32 nodes in the first and second layer. The predictor we use is a random forest with 100 decision trees. Figure 2.7 shows that our method exceeds all of the baseline methods in the task of exploring a qualified feature subset.

Robustness Check

The predictive accuracy relies on not just feature selection, but also predictors. We apply our method to different predictors in order to investigate whether our explored feature subset are consistently stable and can consistently outperform other baseline methods on various predictors. In this way, we can examine the robustness of our methods. Aside from the random forest (RF) predictor, we use (i) LASSO; (ii) Decision Tree (DT); (iii) SVM with a rbf kernel, and (iv) XGBoost as predictors for this experiment. Table 2.1 shows that our MARLFS outperforms the baselines methods over almost all of the predictors. However, when we use LASSO to perform both feature selection and target prediction, the accuracy of our method is slightly lower than LASSO. This might be explained by the reason that both feature section and prediction optimization are integrated and unified in a single model framework. However, when we use LASSO to perform feature selection, and use other



(a) Overall Accuracy (b) Precision

(c) Recall (d) F-Measure

Figure 2.7: Performance comparison of different feature selection algorithms.

classification models for prediction, our method outperform such type of baselines.

Study of Reward Function

We study the impacts of the reward function design in our method. We consider four cases: (i) **Acc** that only considers accuracy in the reward function; (ii) **Rv** that only considers relevance in the reward function; (iii) **Rd** that only considers redundancy in the reward function; (iv) **Acc+Rv+Rd** that considers accuracy, relevance and redundancy in the reward function.

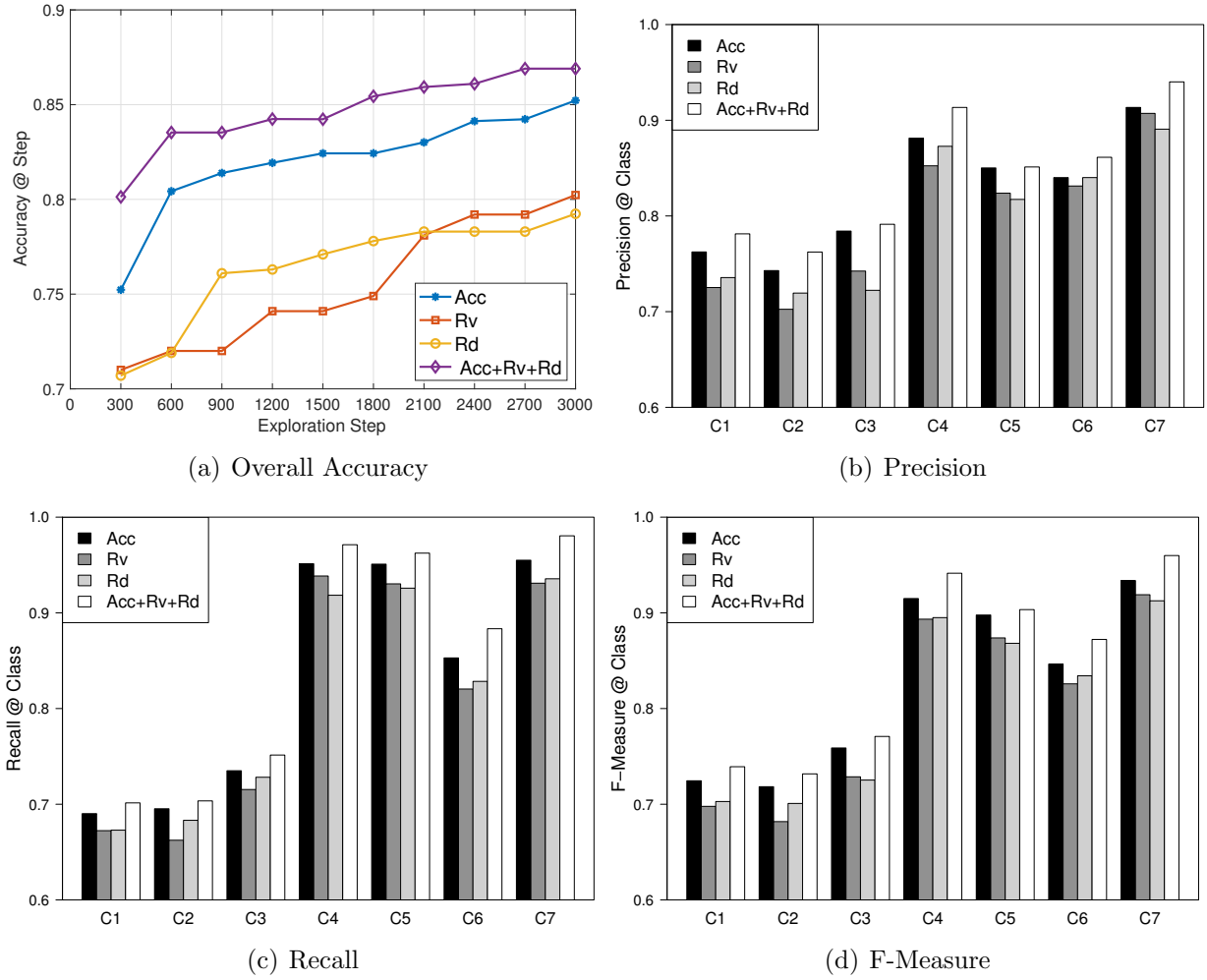
Table 2.1: Overall accuracy of feature selection algorithms on different predictors.

		Predictors				
		RF	LASSO	DT	SVM	XGBoost
Algorithms	K-Best	0.7943	0.8246	0.8125	0.8324	0.8076
	mRMR	0.8042	0.8124	0.8096	0.8175	0.8239
	LASSO	0.8426	**0.8513**	0.8241	0.8131	0.8434
	RFE	0.8213	0.8236	0.8453	0.8257	0.8348
	GFS	0.8423	0.8318	0.8350	0.8346	0.8302
	SARLFS	0.8321	0.8295	0.8401	0.8427	0.8450
	MARLFS	**0.8690**	0.8424	**0.8583**	**0.8542**	**0.8731**

Figure 2.8 shows that Acc is the second best reward function, since it leads the exploration to the direction of improving accuracy. Rv and Rd are less satisfactory. This is because both are unsupervised indicators of rewards and are not directly relevant to prediction accuracy. Acc+Rv+Rd achieve the best performances since it considers both supervised indicator and unsupervised indicator into account. Specifically, Figure 2.8(a) shows the comparisons of overall accuracy over exploration steps. Figure 2.8(b), 2.8(c) and 2.8(d) show the comparisons of precision, recall and F-measure over different classes with 3000 exploration steps.

Study of State Representation Learning

We compare the performances different representation learning methods. We consider five cases, i.e., (i) **MDS**: meta descriptive statistics, which uses the meta data of descriptive statistics of feature subspace to represent the state; (ii) **AE**: auto-encoder based deep representation, which uses deep auto-encoder to encode feature subspace twice to obtain state representation; (iii) **GCN**: uses Dynamic-Graph Based GCN; (iv) **MDS+AE**: combines the variables of (i) and (ii) to represent the state; (v) **MDS+AE+GCN**: combines the variables of (i), (ii), and (iii) to represent the state.

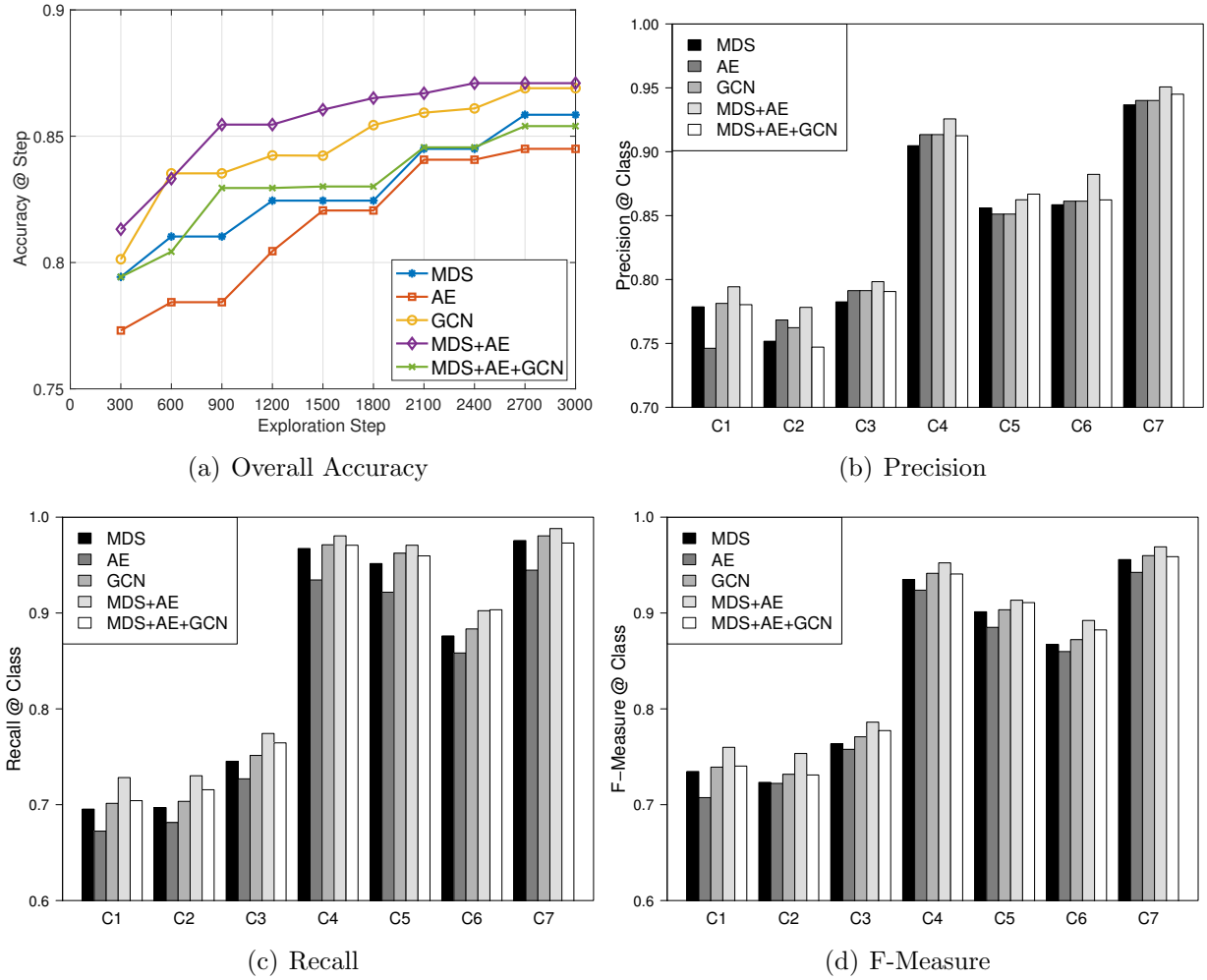


(a) Overall Accuracy (b) Precision

(c) Recall (d) F-Measure

Figure 2.8: Performance comparison of different reward functions.

Figure 2.9 shows GCN outperform MDS and AE. This is because GCN could better capture the relationship between features in the feature subspace. After taking the two combined methods into account, MDS+AE achieves the best performance, since it considers both explicit and implicit information from the selected features. An interesting observation is that MDS+AE+GCN doesn't have better performance than MDS+AE+GCN. This might be explained by the fact that there is potential training loss in the training phrase of AE and GCN. In other words, integrating AE and GCN might possibly introduce more model biases. Specifically, Figure 2.9(a) shows the comparisons of overall accuracy over exploration steps.



(a) Overall Accuracy

(b) Precision

(c) Recall

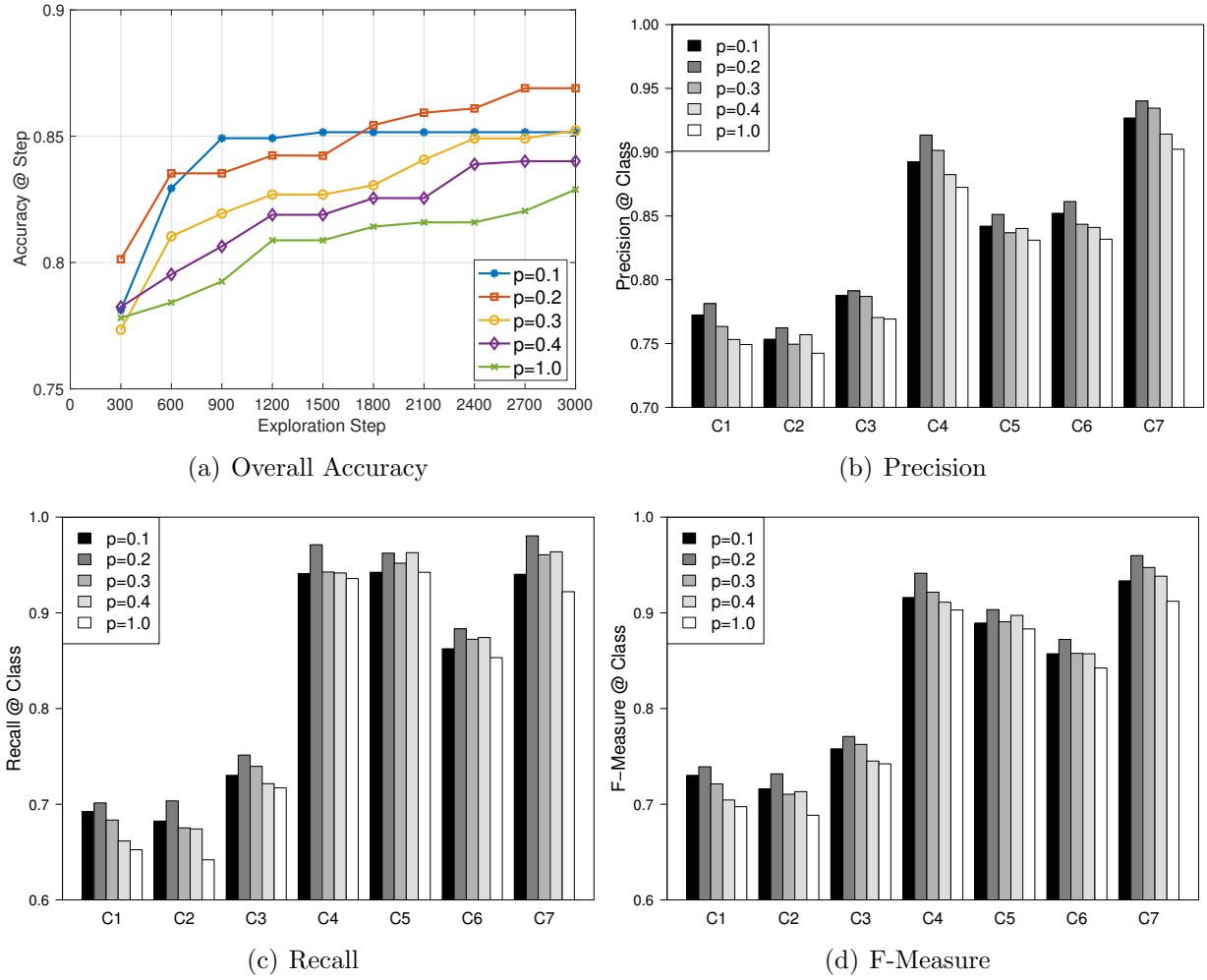
(d) F-Measure

Figure 2.9: Performance comparison of different representation learning methods.

Figure 2.9(b), 2.9(c) and 2.9(d) show the comparisons of precision, recall and F-measure over different classes with 3000 exploration steps.

Study of GMM based Generative Rectified Sampling

We study the impacts of GMM-based generative rectified sampling, where the high-quality proportion $p \in [0.1, 0.2, 0.3, 0.4, 1]$ respectively. Here, when $p = 1$, our GMM based method will be reduced into the traditional sampling strategy, where samples are considered as high-



(a) Overall Accuracy

(b) Precision

(c) Recall

(d) F-Measure

Figure 2.10: Performance comparison of different GMM sampling strategies.

quality. We call the method with $p = 1$ as the non GMM method.

Figure 2.10 all GMM-based sampling methods ($p < 1$) outperform the non-GMM method. Among the all these methods, $p = 0.2$ shows the best performances and can quickly explore a quality feature space. Specifically, 2.10(a) shows the comparisons of overall accuracy over exploration steps. Figure 2.10(b), 2.10(c) and 2.10(d) show the comparisons of precision, recall and F-measure over different classes with 3000 exploration steps.

Table 2.2: Overall accuracy comparison of sampling strategies.

		Predictors				
		RF	LASSO	DT	SVM	XGBoost
Sampling	Uniform	0.8585	0.8381	0.8392	0.8406	0.8627
	GMM	0.8690	0.8424	0.8583	0.8542	0.8731
	Softmax	0.8633	0.8400	0.8583	0.8496	0.8659

Study of Softmax Sampling

We study the impacts of softmax sampling and compare it with vanilla experience replay (uniformly sampling) and GMM-based sampling. We use meta descriptive statistics as the state representation method, and we use MARLFS as feature selection method. We run each setting 5 times to compare their averaged execution time per step. The exploration step is set to 3000. Our experiments were conducted on a machine with the following specification:

- Processor: 64-bit Intel I9-9920X @ 4.40GHz with 12 core(s)
- Memory: 128GiB DIMM DDR4 2666MHz
- SSD: 2TB PCIe NVMe - M.2

Table 2.2 shows that the GMM-based sampling method achieves the best accuracy, and the softmax sampling method has a parellel accuracy with GMM-based sampling, which are both higher than the uniform sampling. Table 2.3 shows that the softmax sampling method costs a bit more time than uniform sampling, and both of them require less execution time than the GMM-based sampling method.

Table 2.3: Execution time comparison of sampling strategies.

		Predictors				
		RF	LASSO	DT	SVM	XGBoost
Sampling	Uniform	1.04s	0.46s	0.50s	17.34s	4.97s
	GMM	1.44s	0.93s	0.94s	17.86s	5.42s
	Softmax	1.06s	0.51s	0.63s	17.46s	5.05s

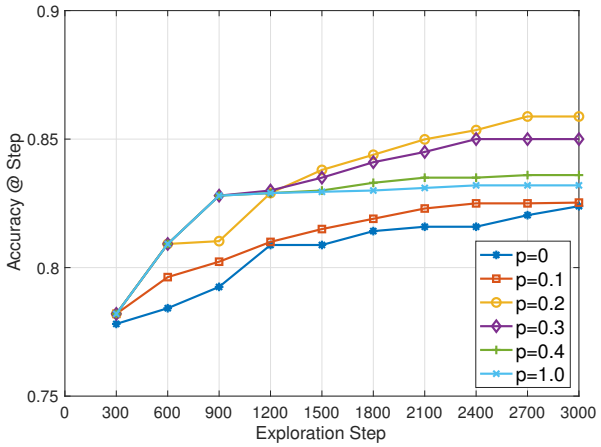


Figure 2.11: Performance comparison of different IRL sampling strategies.

Study of Interactive Reinforcement Learning

We study the impacts of IRL, where its apprenticeship steps $N_A \in [0, 0.1, 0.2, 0.3, 0.4, 1]*3000$ respectively. We set $K = 38$. Here, when $N_A = 0$, the IRL strategy will be reduced into the traditional exploration strategy, and when $N_A = 3000$, the IRL spends all its exploration steps being advised by K-Best Selection method. Figure 2.11 shows that the IRL sampling method ($p = 0.0$) outperforms the non-GMM method ($p > 0.0$). When $p = 0.2$, it can achieve the best performances and can quickly explore a quality feature space.

Feature Selection. Feature selection can be categorized into three types, based on how the feature selection algorithm combines with the machine learning tasks, i.e., filter methods, wrapper methods and embedded methods [11, 85]. Filter methods rank the features merely by relevance scores and only top-ranking features are selected. The representative filter methods are univariate feature selection [105, 29] and correlation based feature selection [43, 106]. With very simply computation complexity, filter methods are very fast and thus they're efficient on high-dimensional datasets. However, they ignore the feature dependencies, as well as interactions between feature selection and the subsequent predictors. Unlike filter methods, wrapper methods take advantage of the predictors and consider the prediction performance as the objective function [38]. The representative wrapper methods are branch and bound algorithms [79, 58]. Wrapper methods are supposed to achieve better performance than filter methods since they search on the whole feature subset space. However, the feature subset space exponentially increases with the number of features, making traversing the feature subset space a NP-hard problem. Evolutionary algorithms [103, 55, 30] low down the computational cost but could only promise local optimum results. Embedded methods combine feature selection with predictors more closely than wrapper methods, and actually they incorporate feature selection as part of predictors. The most widely used embedded methods are LASSO [96], decision tree [93] and SVM-RFE [42]. Embedded methods could have supreme performance on the incorporated predictors, but normally not very compatible with other predictors.

Multi-Agent Reinforcement Learning. Our work is related to multi-agent reinforcement learning, where multiple agents share a complex environment and interact with each other [95]. *Stankovic et al.* proposed new algorithms for multi-agent distributed iterative value

function approximation where the agents are allowed to have different behavior policies while evaluating the response to a single target policy [90]. *Liao et al.* proposed Multi-objective Optimization by Reinforcement Learning (MORL) to solve the optimal power system dispatch and voltage stability problem, which is undertaken on individual dimension in a high-dimensional space via a path selected by an estimated path value which represents the potential of finding a better solution [65]. *Yang et al.* developed deep reinforcement learning algorithms which could handle large scale agents with effective communication protocol [104, 82]. *Lin et al.* proposed to tackle the large-scale fleet management problem using reinforcement learning, and proposed a contextual multi-agent reinforcement learning framework which successfully tackled the taxi fleet management problem [66]. However, these methods define their states by handcraft rules instead of by representation learning, which may leave out important information provided by the environment. And also, as we know the training speed of multi-agent reinforcement learning is low due to the large action space, but these methods rarely study how to improve the training efficiency.

Reinforcement Learning in Feature Selection. Existing studies [27, 60] create a single agent to make decisions. However, this agent has to determine the selection or disselection of all N features. In other words, the action space of this agent is 2^N. Such formulation is similar to the evolutionary algorithms [103, 55, 30], which are NP-hard problems and can merely obtain local optima.

Conclusion Remarks

In this chapter, we study the problem of automated feature subspace exploration. Through this method, we can reduce dimensionality, shorten training times, enhance generalization, avoid overfitting, and improve predictive accuracy in order to support downstream predictive

tasks. We formulate the problem of automated feature subspace exploration as a multi-agent reinforcement learning framework, in which each feature is associated to a feature agent, a feature agent can decide to select or drop a feature, the reward function is a combination of accuracy, redundancy, and relevance, and the environment is the characteristics of the selected feature subspace. To better represent the environment, we propose three different representation learning methods. To accelerating feature exploration, we develop a GMM-based generative rectified sampling strategy a softmax sampling strategy and an IRL-based exploration strategy. Finally, we present extensive experiments on two real world datasets to demonstrate the effectiveness of the proposed method. The proposed feature selection method can be widely used in the data mining and machine learning areas, and it is more helpful when users are not experts in feature selection but need this technology in their work and research.

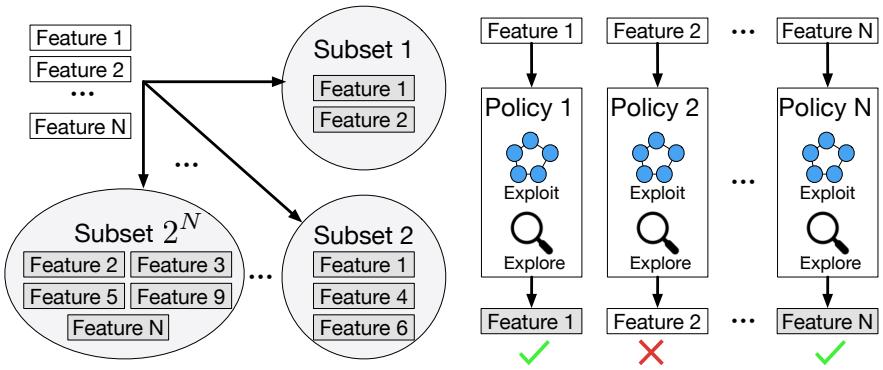
CHAPTER 3: SINGLE_AGENT REINFORCEMENT FEATURE SELECTION

In this chapter, I propose a single-agent Monte Carlo based reinforced feature selection (MCRFS) method, as well as two efficiency improvement strategies, i.e., early stopping (ES) strategy and reward-level interactive (RI) strategy.

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 technique, 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 [105, 29], correlation based feature selection [43, 106]); (ii) wrapper methods, in which optimal feature subset is identified by a search strategy that collaborates with predictive tasks (e.g., evolutionary algorithms [103, 55], branch and bound algorithms [79, 58]); (iii) embedded methods, in which feature selection is part of the optimization objective of predictive tasks (e.g., LASSO [96], decision tree [93]). 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 non-zero

weighted features are considered to be important.



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

Figure 3.1: 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.

Recently, reinforcement learning has been incorporated with feature selection and produces an emerging feature selection method, called reinforced feature selection [70, 26]. 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: Can we design a more practical and efficient method to address the feature selection problem while preserving the effectiveness of reinforced fearure 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. 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 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 generates 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 [70], 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 [94]. 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 chapter, 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 skew, the following samples are skew. We propose a stopping criteria based on the importance sampling weight, and re-calculate a more appropriate weight to make the samples from the behavior policy more close to the target policy.

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 chapter, 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 chapter, we propose a state-based reward integration strategy, which leads to a more inspiring integrated reward as well as preserving the optimal policy.

To summarize, the contributions of this chapter 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 a reward-level interactive strategy to improve the training efficiency. (5) We design extensive experiments to reveal the superiority of the proposed method.

Table 3.1: 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 $\{s^i \| i < inf\}$
$\mathcal{A}$	action space defined as $\{a^j \| j \in [1, N]\}$
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 $\{\mathbf{f}^k \| k \in [0, M]\}$

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.

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 $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a mapping function from state-action pair to a scalar, $\gamma \in [0, 1]$ is a discount factor and $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition probability from state-action pair to the next state. In this chapter, 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.

Monte Carlo for Solveing 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 Section 4), and improve it by Bellman optimality:

$$\pi^{i+1}(s) = argmax_a Q_{\pi^i}(s, a) \tag{3.1}$$

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:

$$\pi^0 \xrightarrow{E} Q_{\pi^0} \xrightarrow{I} \pi^1 \xrightarrow{E} Q_{\pi^1} \xrightarrow{I} \ldots \pi^M \xrightarrow{I} Q_{\pi^M} \tag{3.2}$$

where $\stackrel{E}{\rightarrow}$ denotes the policy evaluation and $\stackrel{I}{\rightarrow}$ denotes the policy improvement. After M iterations, we can achieve an optimal policy. As Equation 3.2 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.

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 [70] formulates the feature selection problem into a multi-agent reinforcement learning task, in order to automate the selection process. As Figure 3.2 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 $\mathcal{F}^i$:

$$s^{i+1} = represent(\mathcal{F}^i) \tag{3.3}$$

where $\mathcal{F}^i$ is the selected feature subset at time t. *represent* is a representation learning algorithm which converts the dynamically changing $\mathcal{F}_t^i$ 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 [70]. The reward r^i is an evaluation of the selected feature subset $\mathcal{F}^i$:

$$r^i = eval(\mathcal{F}^i) \tag{3.4}$$

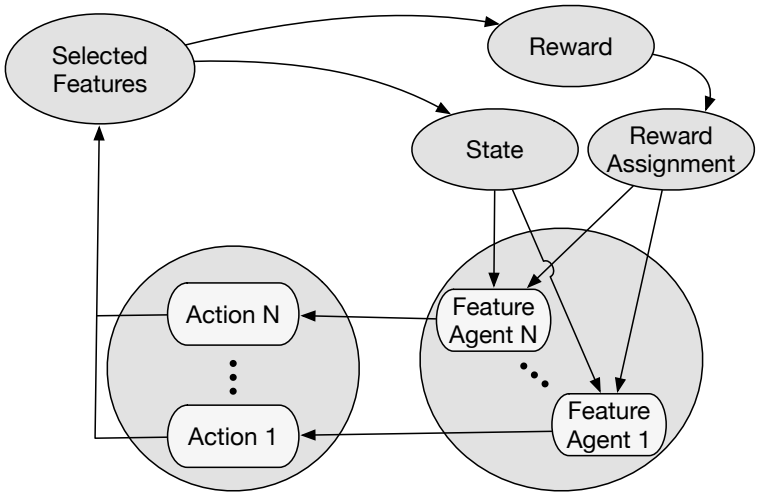


Figure 3.2: Multi-agent reinforced feature selection. Each feature is controlled by one feature agent.

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 [70]. 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.

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

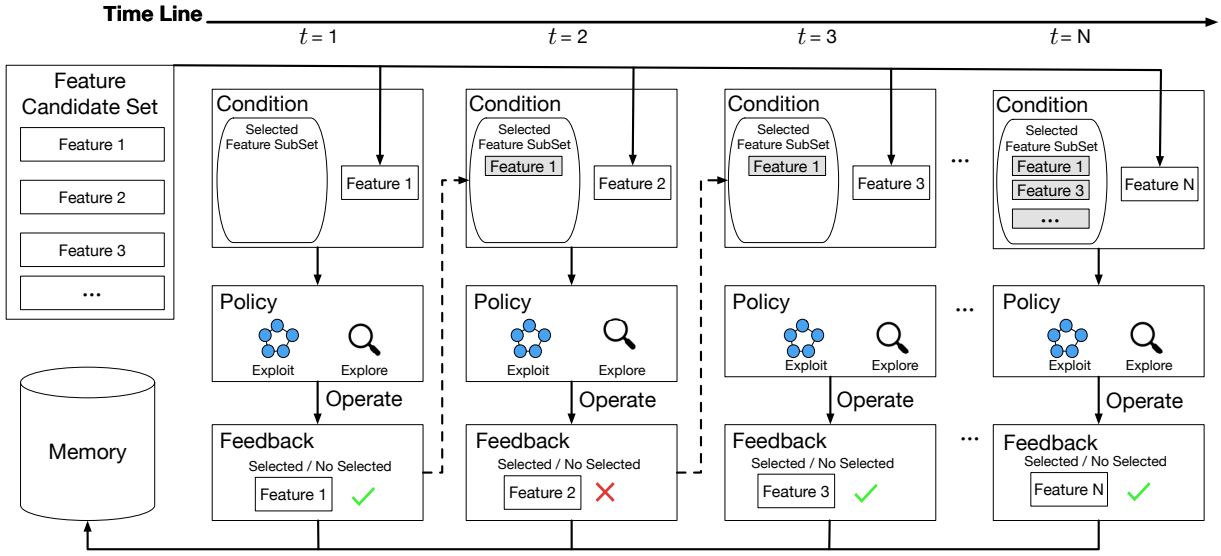


Figure 3.3: Single-agent reinforced feature selection with traverse strategy. At each step, the agent transverses
features one by one to decide their selection. The traverse data are stored in the memory to form a training episode.

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 Mote Carlo method as the reinforcement learning algorithm.

Traverse Strategy

As Figure 3.3 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 forms 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 $eval(\mathcal{F}_t^i)$.

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:

$$G^i(s_t^i, a_t^i) = \sum_{j=0}^{t} \gamma^{(t-j)} \cdot r_j^i \tag{3.5}$$

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:

$$\rho_t^i = \frac{\Pi_{j=0}^{t} \pi^i(a_j^i | s_j^i)}{\Pi_{j=0}^{t} b^i(a_j^i | s_j^i)} \tag{3.6}$$

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

$$\mathcal{L}_{\pi^i} = ||Q_{\pi^i}(s_t^i, a_t^i) - \rho_t^i * G^i(s_t^i, a_t^i)||^2 \tag{3.7}$$

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

$$\pi^{i+1}\{a|s\} = \frac{exp(Q_{\pi^i}\{a,s\})}{exp(Q_{\pi^i}\{a=0,s\}) + exp(Q_{\pi^i}\{a=1,s\})} \tag{3.8}$$

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

$$b^{i+1}\{a|s\} = \begin{cases} 1-\epsilon & a = argmax_a Q_{\pi^i}(s,a); \\ \epsilon & otherwise; \end{cases} \tag{3.9}$$

Algorithm 4 shows the process of Monte Carlo based feature selection (MCRFS) with traverse strategy.

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 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 skew [74]. The skew samples not only are a waste time 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.

Algorithm 4: Monte Carlo Based Reinforced Feature Selection with Traverse Strategy

Input: Feature set $\mathcal{F} = \{f_1, f_2, ..., f_N\}$, downstream machine learning task $\mathcal{T}$.
Output: Optimal feature subset $\mathcal{F}'$.
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
  Update target policy π^{i+1} by Equation 3.8 and behavior policy b^{i+1} by Equation 3.9.
  if $eval(\mathcal{F}_N^i) > eval(\mathcal{F}')$ **then**
    $\mathcal{F}' = \mathcal{F}_N^i$.
  end
end
Return $\mathcal{F}'$.

Incremental Importance Sampling

Rather than calculating the importance sampling weight for each sample directly by Equation 3.6, we here decompose it into an incremental format. Specifically, in the i-th iteration, we define the weight increment:

$$w_t^i = \frac{\pi^i(a_t^i|s_t^i)}{b^i(a_t^i|s_t^i)} \tag{3.10}$$

and the importance sampling weight can be calculated by:

$$\rho_t^i = \rho_{t-1}^i \cdot w_t^i \tag{3.11}$$

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

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:

$$p_t^i = max(0, 1 - \rho_t^i / v) \tag{3.12}$$

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:

$$w_t^i = p_v^i \cdot \rho_t^i / p_t^i \tag{3.13}$$

where the p_v can be calculated by:

$$p_v^i = \int max(0, 1 - \rho_t^i / v) b^i(\mathbf{s}_t)\ d\mathbf{s}_t \tag{3.14}$$

As p_v^i is identical for all samples in the i-th episode regardless of t, the calculation of Equation 3.13 does almost no increase to the computation.

Decision history based traversing strategy. In the i-th iteration, the stopping criteria stops 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 5 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:

1. Use a random behavior policy b^0 to traverse the feature set. Stop the traverse with the probability in Equation 3.12 and get an episode $\mathbf{x}^0_{N_0}$.

2. Evaluate the policy π^0 to get the Q value Q^0 by minimizing Equation 3.7, and derive the updated policy π^1 and b^1 from Equation 3.8 and 3.9 respectively.

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.

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.

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 $\pi^*_{\mathcal{M}}(s) = argmax_a Q^*_{\mathcal{M}}(s, a)$ for an MDP $\mathcal{M}$. The optimal Q-value can be updated by Bellman equation [4]:

Algorithm 5: Monte Carlo Based Reinforced Feature Selection with Early Stopping Traverse Strategy

Input: Feature set $\mathcal{F} = \{f_1, f_2, ..., f_N\}$, downstream machine learning task $\mathcal{T}$.
Output: Optimal feature subset $\mathcal{F}'$.
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 Equation 3.12;
  end
  Update target policy π^{i+1} by Equation 3.8 and behavior policy b^{i+1} by Equation 3.9.
  if $eval(\mathcal{F}_N^i) > eval(\mathcal{F}')$ **then**
    $\mathcal{F}' = \mathcal{F}_N^i$.
  end
end
Return $\mathcal{F}'$.

$$Q^*_{\mathcal{M}}(s_t, a_t) = \mathcal{R}(s_t, a_t) + \gamma * max_{a_{t+1}} Q^*_{\mathcal{M}}(s_{t+1}, a_{t+1}) \tag{3.15}$$

Interactive reinforcement learning (IRL) is proposed to accelerate the learning process of reinforcement learning (RL) by providing external action advice to the RL agent [92]. As Figure 3.4 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 [97]. 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.

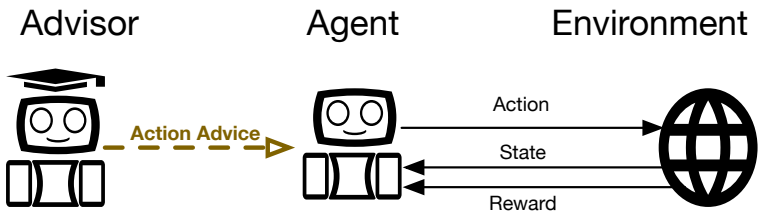


Figure 3.4: Classic interactive reinforcement learning. The advisor gives the agent advice at the action level.

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

$$\mathcal{R}'(a_t, s_t) = \mathcal{R}(a_t, s_t) + c * (\gamma * \mathcal{U}(s_{t+1}) - \mathcal{U}(s_t)) \tag{3.16}$$

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 Equation 3.16:

We firstly subtract $c * \mathcal{U}(s_t)$ from both sides of Equation 3.15, and we have:

$$\begin{aligned} Q^*_{\mathcal{M}}(s_t, a_t) - c * \mathcal{U}(s_t) = \mathcal{R}(s_t, a_t) \\ + \gamma * max_{a_{t+1}} Q^*_{\mathcal{M}}(s_{t+1}, a_{t+1}) - c * \mathcal{U}(s_t) \end{aligned} \tag{3.17}$$

Algorithm 6: Reward-Level Interactive Reinforcement Learning

Initialize replay memory $\mathcal{D}$; Initialize the Q-value function with random weights;
Initialize the advising state number N_a, stop time T;
for $t = 1$ *to* T **do**

$a_t = \begin{cases} \text{random action} & \text{with probability} \epsilon; \\ max_{a_t} Q(s_t, a_t) & \text{with probability} 1 - \epsilon; \end{cases}$

Perform a_t, obtaining reward $\mathcal{R}(a_t, s_t)$ and next state s_{t+1};

$\mathcal{R}'(s_t, a_t) = \begin{cases} \mathcal{R}(s_t, a_t) & t > N_a; \\ \mathcal{R}(s_t, a_t) + c * (\gamma * \mathcal{U}(s_{t+1}) - \mathcal{U}(s_t)) & t \leq N_a; \end{cases}$

Store transition $(s_t, a_t, \mathcal{R}'(s_t, a_t), s_{t+1})$ in $\mathcal{D}$;
Randomly sample mini-batch of data from $\mathcal{D}$;
Update $Q(s, a)$ with the sampled data;
end

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

$$\begin{aligned} & Q^*_{\mathcal{M}}(s_t, a_t) - c * \mathcal{U}(s_t) = \mathcal{R}(s_t, a_t) \\ & + \gamma * max_{a_{t+1}} Q^*_{\mathcal{M}}(s_{t+1}, a_{t+1}) - c * \mathcal{U}(s_t) \\ & + c * \gamma * \mathcal{U}(s_{t+1}) - c * \gamma * \mathcal{U}(s_{t+1}) \\ & = \mathcal{R}(s_t, a_t) + c * \gamma * \mathcal{U}(s_{t+1}) - c * \mathcal{U}(s_t) \\ & + \gamma * max_{a_{t+1}} [Q^*_{\mathcal{M}}(s_{t+1}, a_{t+1}) - c * \mathcal{U}(s_{t+1})] \end{aligned} \tag{3.18}$$

We define:

$$Q^{\partial}_{\mathcal{M}'}(s_t, a_t) = Q^*_{\mathcal{M}}(s_t, a_t) - c * \mathcal{U}(s_t) \tag{3.19}$$

Then Equation 3.18 has the new form:

$$\begin{aligned} & Q^{\partial}_{\mathcal{M}'}(s_t, a_t) = \mathcal{R}(s_t, a_t) + c * [\gamma * \mathcal{U}(s_{t+1}) - \mathcal{U}(s_t)] \\ & + \gamma * max_{a_{t+1}} Q^{\partial}_{\mathcal{M}'}(s_{t+1}, a_{t+1}) \\ & = \mathcal{R}'(s_t, a_t) + \gamma * max_{a_{t+1}} Q^{\partial}_{\mathcal{M}'}(s_{t+1}, a_{t+1}) \end{aligned} \tag{3.20}$$

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.,

$$Q^{*}_{\mathcal{M}'}(s_t, a_t) = Q^{\partial}_{\mathcal{M}'}(s_t, a_t) \tag{3.21}$$

We combine Equation 3.19 and Equation 3.21 and have:

$$Q^{*}_{\mathcal{M}'}(s_t, a_t) = Q^{*}_{\mathcal{M}}(s_t, a_t) - c * \mathcal{U}(s_t) \tag{3.22}$$

Obviously,

$$\begin{aligned} argmax_{a_t} Q^{*}_{\mathcal{M}'}(s_t, a_t) &= argmax_{a_t}[Q^{*}_{\mathcal{M}}(s_t, a_t) - c * \mathcal{U}(s_t)] \\ &= argmax_{a_t} Q^{*}_{\mathcal{M}}(s_t, a_t) \end{aligned} \tag{3.23}$$

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 $\mathcal{R}$.

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 6. Specifically, we adapt the early advising strategy [97] to select the advising states, i.e., the advisor gives advice for the first n states the IRL agent meets.

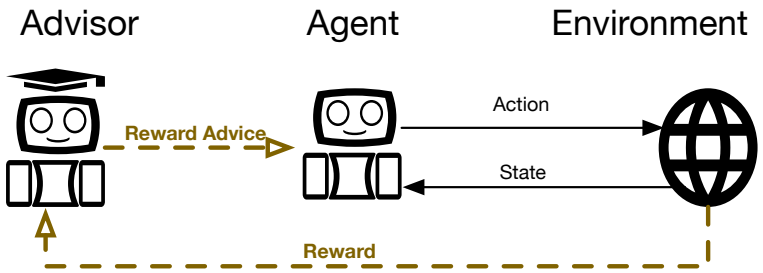


Figure 3.5: Reward-level interactive reinforcement learning. The advisor gives advice at the reward level.

Comparison with Prior Literature

Compared with filter methods, our methods capture feature interactions; Compared with wrapper methods, our methods reduce the search space; Compared with embedded methods, our methods don't rely on strong structured assumptions; Compared with multi-agent reinforcement learning feature selection, our methods achieve parallel performance with lower computational cost.

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.

Experimental Setup

Data Description

We use four publicly available datasets on classification task to validate our methods, i.e., Forest Cover (FC) dataset [8], Spambase (Spam) dataset [24], Insurance Company Benchmark (ICB) dataset [98] and Arrhythmia (Arrhy) dataset [40]. The statistics of the datasets are in Table 3.2.

Table 3.2: Statistics of datasets.

	FC	Spam	ICB	Arrhy
Features	54	57	86	274
Samples	15120	4601	5000	452

Table 3.3: Overall performance.

		FC		Spam		ICB		Arrhy	
		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
	LASSO	0.8438	0.8493	0.9143	0.8556	0.8801	0.8507	0.6293	0.6543
	GFS	0.8498	0.8350	0.9043	0.8431	0.9099	0.637	0.6406	0.6550
	mRMR	0.8157	0.8241	0.8980	0.8257	0.8998	0.8423	0.6307	0.6368
	RFE	0.8046	0.8175	0.9351	0.8480	0.9045	0.8502	0.6452	0.6592
	MARFS	0.8653	0.8404	0.9219	0.8742	0.8902	0.8604	0.7238	0.6804
	MCRFS	0.8688	0.8496	0.9256	0.8738	0.8956	0.8635	0.7259	0.7152
	ES-MCRFS	**0.8942**	**0.8750**	**0.9402**	**0.9067**	**0.9187**	**0.8803**	**0.7563**	**0.7360**

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 $Acc = \frac{TP+TN}{TP+TN+FP+FN}$, 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 $P = \frac{TP}{TP+FP}$ is precision and $R = \frac{TP}{TP+FN}$ is recall.

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 [105]. In the experiments, we set k equals to half of the number of input features.(2) **LASSO** conducts feature selection via $l1$ penalty [96]. The hyper parameter 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 [61]. (4) **mRMR** ranks features by minimizing feature's redundancy and maximizing their relevance with the label [81]. (5) **RFE** selects features by recursively selecting smaller and smaller feature subsets [37].(6)**MARFS** is a multi-agent reinforcement learning based feature selection method [70]. 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**.

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 auto-encoder method whose encoder/decoder network is a two-layer ReLU with 128 and 32 nodes in the first and second layer.

Environmental Setup

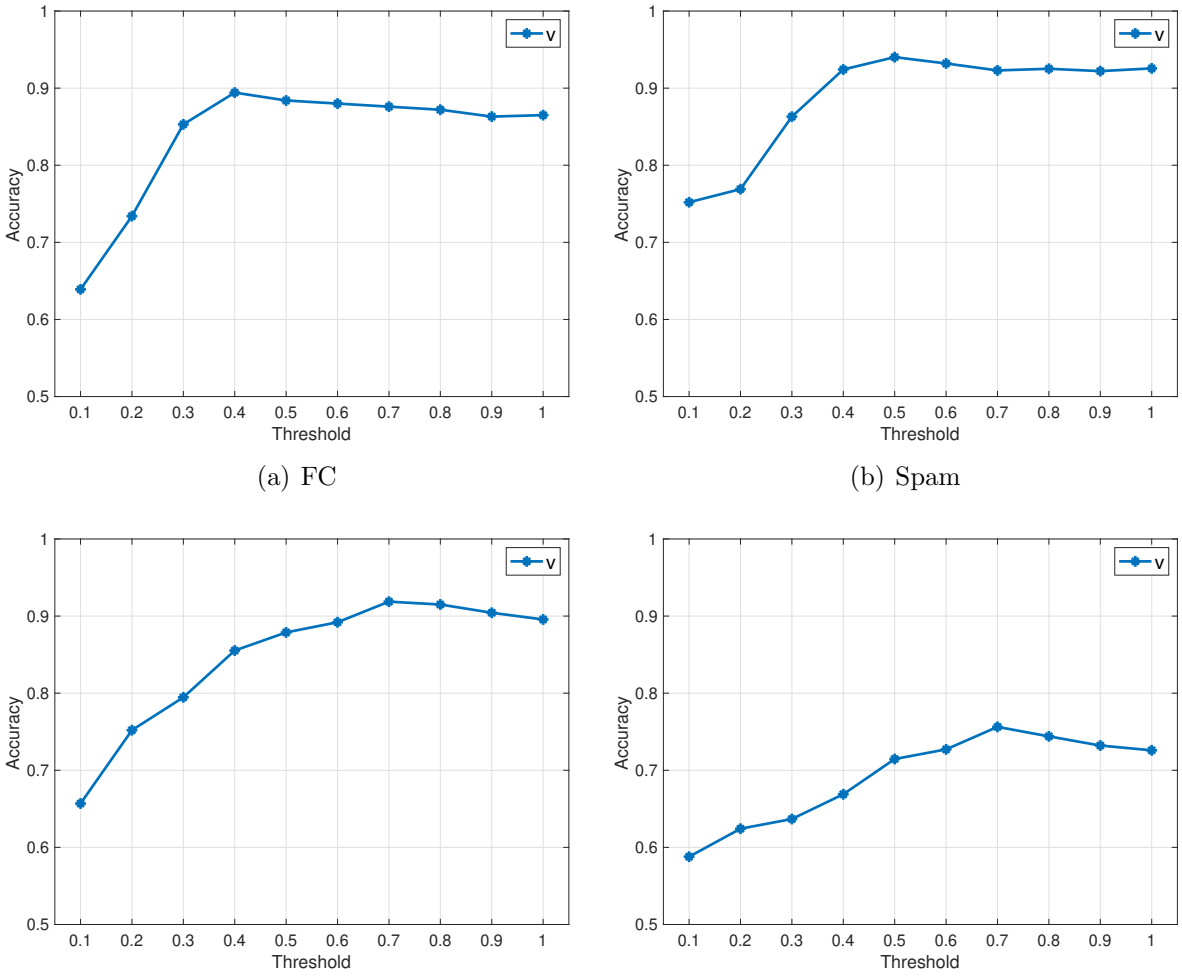
The experiments were carried on a server with an I9-9920X 3.50GHz CPU, 128GB memory and a Ubuntu 18.04 LTS operation system.

Overall Performance

We compare the proposed ES-MCRFS method with baseline methods and its variant with regard to the predictive accuracy. As Table 3.3 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.

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



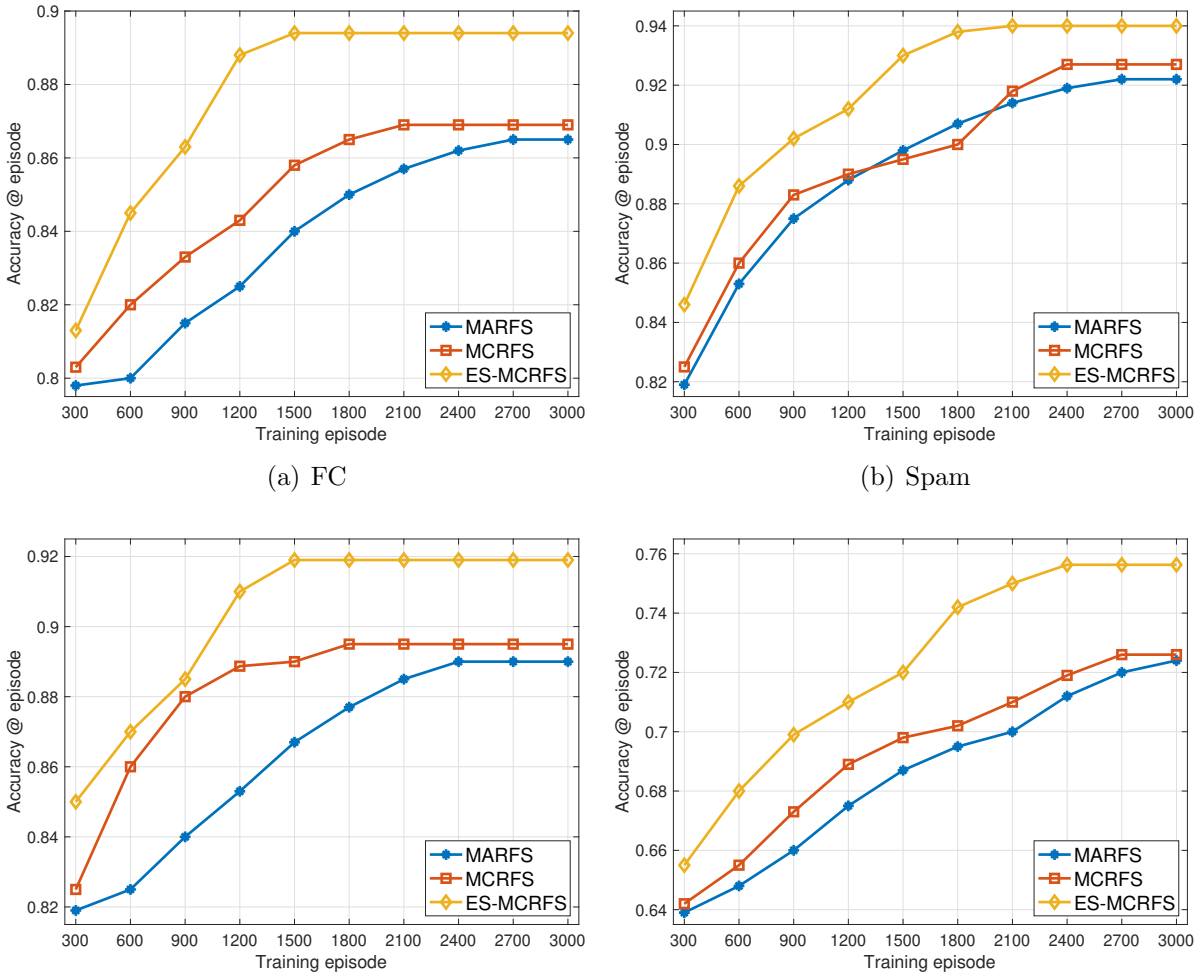
(a) FC (b) Spam

(c) ICB (d) Arrhy

Figure 3.6: Threshold sensitivity of early stopping criteria.

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 3.7 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.



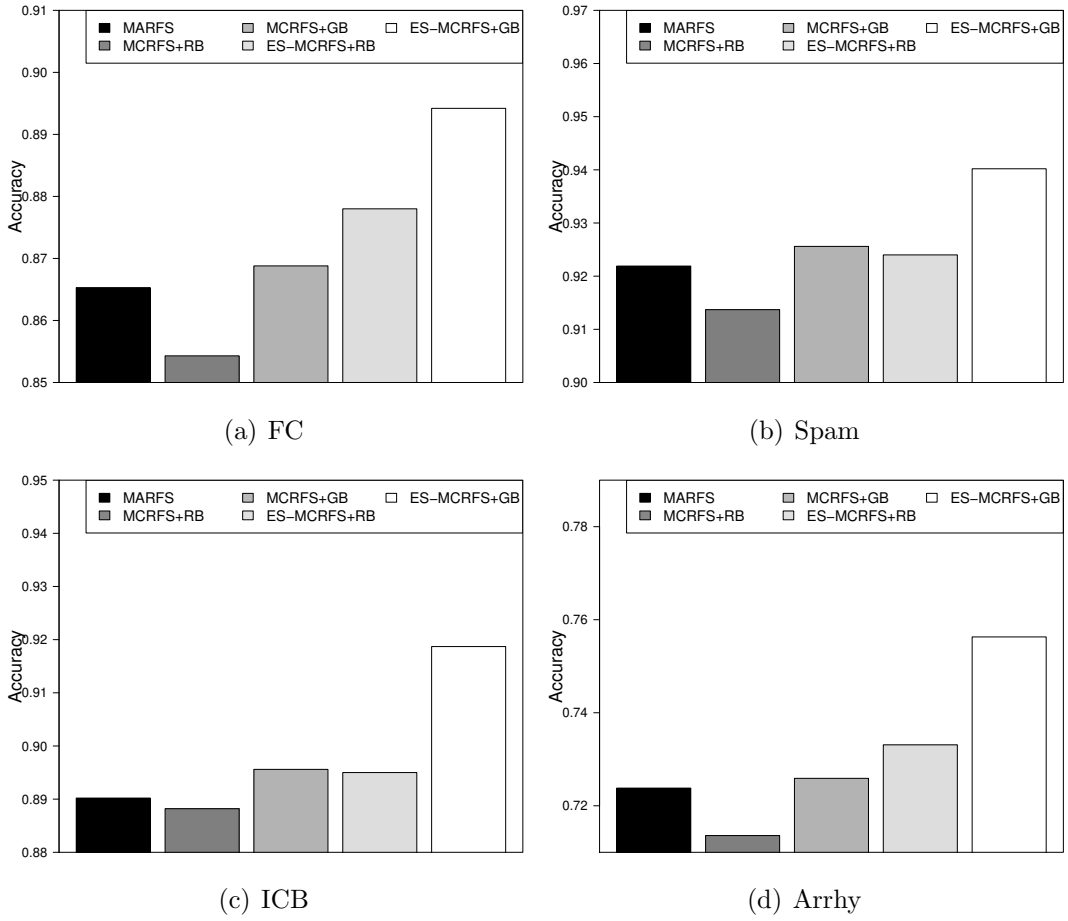
(a) FC (b) Spam

(c) ICB (d) Arrhy

Figure 3.7: Predictive accuracy on training step.

Study of the Behavior Policy

We study the difference between random behavior policy and the ϵ-greedy policy presented in Equation 3.9. We combine the two policies with MCRFS and ES-MCRFS respectively. Figure 3.8 shows that the ϵ-greedy policy outperforms the random behavior policy on all datasets.



(a) FC (b) Spam

(c) ICB (d) Arrhy

Figure 3.8: Predictive accuracy on different training strategies. RB for random behavior policy and GB for ϵ-greedy behavior policy.

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

Table 3.4: 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

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 3.5 shows that the decision history can significantly improve performance of the traverse strategy.

Table 3.5: Traverse strategy ablation. DH for decision history.

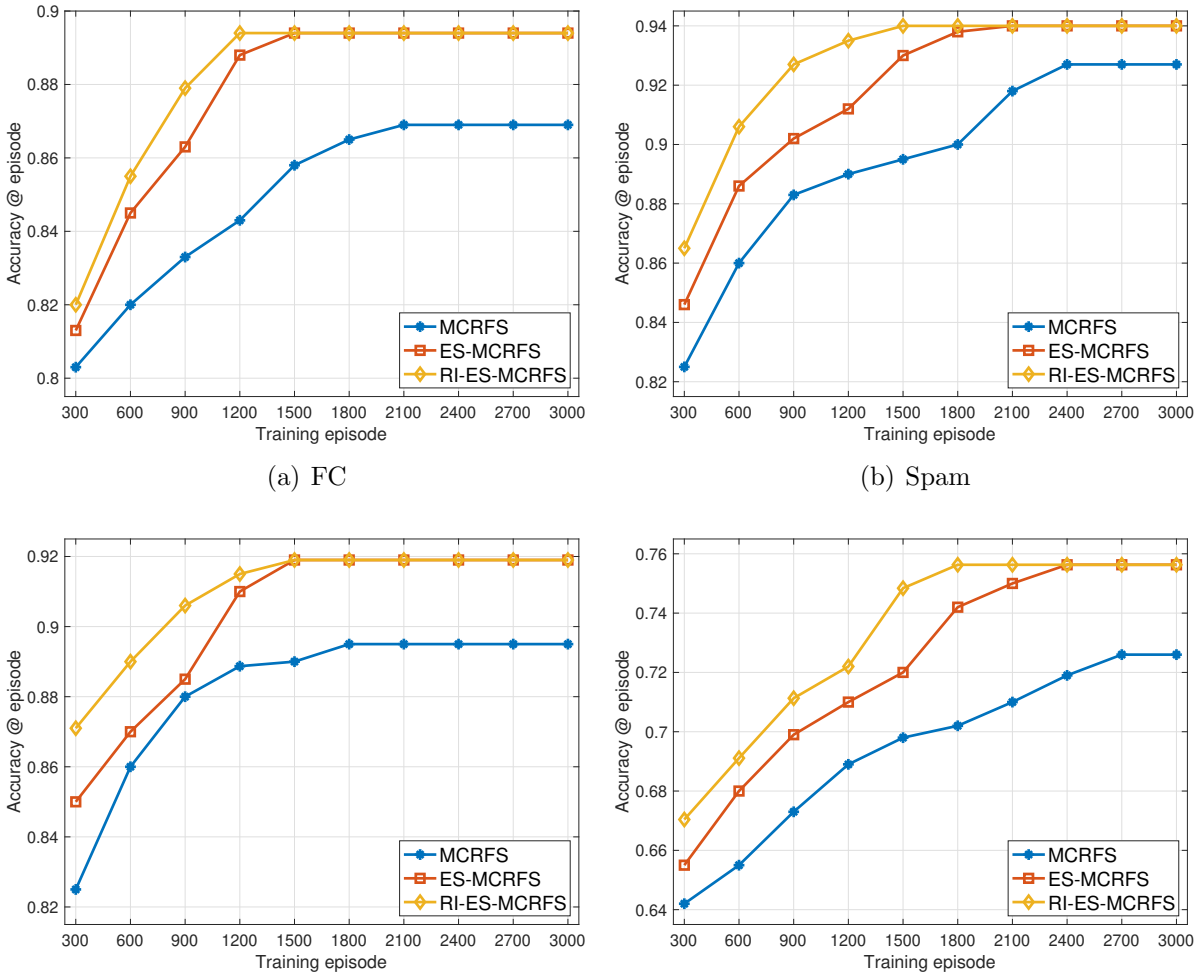
	FC		Spam		ICB		Arrhy	
	Acc	F1	Acc	F1	Acc	F1	Acc	F1
No DH	0.75	0.82	0.83	0.79	0.75	0.81	0.59	0.56
With DH	0.89	0.88	0.94	0.91	0.92	0.88	0.76	0.74

Table 3.6: 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**

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 3.9 shows that



(a) FC (b) Spam

(c) ICB (d) Arrhy

Figure 3.9: Predictive accuracy on training step.

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.

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 3.6 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, 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.

Related Work

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 [69] and marginalized importance sampling [102]. 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 have better performance on some specific tasks [31]. However, these methods suffer from slow convergence and no theoretical guarantee [107]. Besides, there are other attempts to develop sample efficient reinforcement learning, such as curiosity-driven exploration and hybrid optimization [83].

Feature Selection. Feature selection can be categorized into three types, i.e., filter methods, wrapper methods and embedded methods [71]. Filter methods rank features only by relevance scores and only top-ranking features are selected. The representative filter methods is the univariate feature selection [29] The representative wrapper methods are branch and bound algorithms [79, 58]. Wrapper methods are supposed to achieve better performance than filter methods since they search on the whole feature subset space. Evolutionary algorithms [103, 55] 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 [96] and decision tree [93].

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 [67], where the authors presents 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 [86, 46], IRL sticks to learning from the environment and the advisor is only an advice-provider in its apprenticeship [57, 100]. 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 [97], the authors study the advising state selection and propose four advising strategies, i.e., early advising, importance advising, mistake correcting and predictive advising.

Conclusion Remarks

Summary. In this chapter, 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.

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 diversify the training process, and the interactive reinforcement learning accelerates the training process without changing the optimal policy.

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.

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, the interactive reinforcement learning can obtain advice from action level and sampling 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).

CHAPTER 4: AUTOMATED ITERATIVE FEATURE GENERATION

Feature generation aims to generate new features from an original feature set, to refine feature space and improve downstream predictive tasks. Traditional hand-crafted feature generation requires human intuition and domain knowledge, therefore it is appealing to develop automated feature generation methods for the convenience of beginners and researchers outside of machine learning area. In this chapter, I propose an iterative feature generation framework.

Introduction

Feature generation aims to generate effective and relevant features from an original feature set. Feature generation can create a refined feature space, identify a better representation of data, and improve downstream predictive tasks. Hand-crafted feature generation methods require extensive human intuition and domain knowledge, therefore it is appealing to develop automated feature generation methods that are more friendly for beginners and researchers outside of machine learning and data mining areas.

Existing research on automated feature generation can be grouped into two categories: (i) deep learning based methods, e.g., deep factorization machine [39] and XDeepFM [64]. These methods can automatically generate effective features, but due to the black-box mechanism of deep learning, the generated features are latent, and, thus, difficult to interpret; (ii) transformation graph based methods, e.g., traversal transformation graph [53]. Such methods can generate explicit features by conducting a series of operations on feature sets. However, they

apply the same operation to each feature and ignore feature-feature heterogeneity, lacking personalized generation plans. Moreover, a feature set grows exponentially large with depth and includes many redundant features.

To fill these gaps, in this chapter, we focus on a new research question: can we propose a method that can simultaneously generate explicit features, take feature heterogeneity into account, and avoid feature explosion? To answer this question, we propose an innovative iterative feature generation framework. In the framework, we iteratively update a generated feature set (GFS). Specifically, in each iteration, we first choose a meta feature from the original feature set, then select an operation from an operation set to make a judgement. If the operation is unary (e.g.,'square','tanh'), then we directly conduct the operation on the meta feature and derive a generated feature. If the operation is binary (e.g., 'addition', 'difference'), then we choose a second meta feature from the original feature set, and conduct the operation on the two meta features. The generated feature is later added back to the original feature set, to develop a new GFS. In the next iteration, we set the GFS as the original feature set and repeat the iterative procedures. After each iteration, we evaluate the GFS. The GFS with the best evaluation score over all iterations is our optimal GFS.

To implement the framework, we develop a Task-free Iterative Feature Generation method (TIFG). In TIFG, we design two utility scores based on mutual information, i.e., utiltiy score 1 to quantify features and utility score 2 to quantify feature sets. When choosing meta features or operation, we rank and choose candidates with highest utilities scores. We evaluate the GFS by utility score 2. To prevent the GFS from growing exponentially large, we conduct feature selection on the GFS when the size of GFS exceeds a pre-defined threshold to reduce dimensionality. The feature selection is implemented by ranking and selecting top features evaluated by utility score 1.

However, the implementation of TIFG doesn't consider the long-term benefit. For example,

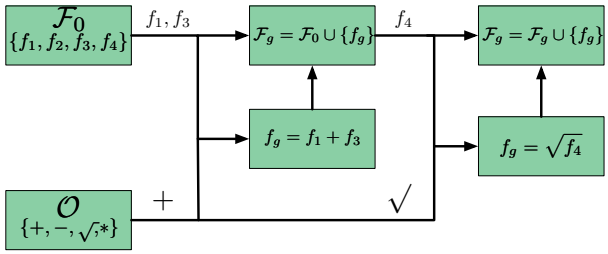


Figure 4.1: A toy example of TIFG with two iterations. $\mathcal{F}_0$, $\mathcal{O}$, f_g, and $\mathcal{F}_g$ are original feature set, operation set, generated feature and generated feature set respectively.

in Figure 4.1, the two generated features are $f_1 + f_3$ and $\sqrt{f_4}$, because each of them is the best choice in their own iteration. However, a more long-term optimized generation can be $\sqrt{f_3}$ and $f_2 * f_4$, because the utility of $\mathcal{F}_0 \cup \{\sqrt{f_3}, f_2 * f_4\}$ is higher than $\mathcal{F}_0 \cup \{f_1 + f_3, \sqrt{f_4}\}$.

To tackle this challenge, we develop a reinforced iterative feature generation method (RIFG) by incorporating reinforcement learning (RL) technology into the iterative feature generation framework. Specifically, we develop three RL agents, i.e., the meta agent 1, the operation agent, the meta agent 2, to respectively select the first meta feature, an operation, and the second meta feature. For the meta agent 1, the environment is defined as the original feature set, the action space is the original feature set, the reward is downstream task improvement (evaluated by classification accuracy or regression loss); For the operation agent, the environment is defined as the combination of original feature set and meta feature, the action space is the operation set, the reward is utility improvement (evaluated by mutual information); For the meta agent 2, the environment is the combination of original feature set, meta feature and operation, the action space is the original feature set, the reward is a combination of utility improvement and downstream task performance.

To summarize, the contributions of this chapter are: (1) We propose an iterative feature generation framework; (2) We implement the proposed framework and propose two algorithms, i.e., task-free iterative feature generation (TIFG) and reinforced iterative feature generation

(RIFG); (3) We design a prioritized experience replay method and a descriptive representation method to better train the RIFG algorithm; (4) We design extensive experiments to reveal the effectiveness and of the proposed feature generation algorithms.

Preliminaries

We firstly introduce some relevant preliminaries of feature generation, and then present the problem formulation.

Definition and Problem Formulation

We generate new features by conducting operations on original features. Formally, we define some conceptions:

Definition 1: Original feature set, denoted by $\mathcal{F}_0 = \{f_1, f_2, \ldots, f_N\}$, refers to the set which consists of original features. The features, selected from the original feature set, to generate new features, are called 'meta feature'. The corresponding samples matrix, $\mathcal{X}_0 = \{X_1, X_2, \ldots, X_N\}$, is depicted by samples as rows and original features as columns. Each sample vector X_i corresponds to one feature f_i. In this chapter, we use 'feature', 'feature set' for interpretation and 'sample vector', 'sample matrix' for calculation.

Definition 2: Operation set, denoted by $\mathcal{O} = \{o\}$, refers to the set which consists of operations to be implemented on features. Each time we choose one operation to conduct on original features. Operations in the operation set can be categorized into two categories, i.e., unary operations and binary operations. In this chapter, we use $o(f_1, f_2)$ to denote the conduction of operation o on feature f_1 and f_2. When o is unary, f_2 is set to NULL by

default.

Definition 3: Generated feature set, denoted by $\mathcal{F}_g = \{f_g\}$, refers to the set which consists of generated features. In this chapter, the features are generated by combinations of operations and original features. Formally, $f(g) = o(f_1, f_2)$, where o is the operation and f_1 and f_2 are two original features. The corresponding generated sample matrix is $\mathcal{X}_g$.

Definition 4: Downstream task, denoted by $\mathcal{T}$, refers to the machine learning task following the feature generation. In this chapter, we choose either classification or regression as the downstream task.

Definition 5: Performance metric, denoted by E, refers to the evaluation standard of the downstream task. In this chapter, we use the predictive accuracy as the performance metric for classification tasks, and 1- relative absolute error for regression tasks.

Problem Formulation. Given an original feature set $\mathcal{F}_0$, a target label y, an operation set $\mathcal{O}$, a downstream task $\mathcal{T}$, a performance metric E, the goal is to generate a feature set $\mathcal{F}_g$ by implementing operations in $\mathcal{O}$ on $\mathcal{F}_0$, such that the performance metric E can be maximized. Formally:

$$\mathcal{F}_g = \text{argmax}_{\mathcal{O}(\mathcal{F}_0)} E_{\mathcal{T}}(\mathcal{O}(\mathcal{F}_0), y) \tag{4.1}$$

where $\mathcal{O}(\mathcal{F}_0)$ represents the generated feature set by conducting operations on original features.

Table 4.1: Commonly used notations.

Notations	Definition
$\mathcal{F}_0$	original feature set
$\mathcal{F}_{0,t}$	original feature set at iteration t
$\mathcal{F}_g$	generated feature set
$\mathcal{F}_{g,t}$	generated feature set at iteration t
$\mathcal{X}_0$	original sample matrix
$\mathcal{S}$	state space defined as $\{s\}$
$\mathcal{A}$	action space defined as $\{a\}$
γ	discount factor in range [0,1]
$\mathcal{P}(s_t, a_t, s_{t+1})$	transition probability
$\mathcal{M}$	Markov decision process (MDP) defined as $\{\mathcal{S}, \mathcal{A}, \mathcal{R}, \gamma, \mathcal{P}\}$
$\mathcal{T}$	downstream machine learning task
E	performance metric of downstream task
$\mathcal{U}$	utility function

Framework Overview

Figure 4.2 shows how we generate new features and update the GFS in an iterative way. The original feature set is defined as the GFS of the previous iteration. Starting from the original feature set $\mathcal{F}_0$, we select a meta feature f_{meta}, and then we select an operation from the operation set. If the operation is binary, then we select another meta feature f'_{meta} from $\mathcal{F}_0$; If the operation is unary, then we set the second meta feature f'_{meta} as NULL. We derive a generate feature by conducting $f_g = o(f_{meta}, f'_{meta})$. Adding the generated feature to the origin feature set will produce a GFS $\mathcal{F}_g = \mathcal{F}_0 \cup \{f_g\}$. This $\mathcal{F}_g$ is considered as a candidate of the optimal GFS. After all iterations, we can get the optimal GFS which has the best evaluation score. Besides, in each iteration, we apply the feature selection technique to control the size of generated feature set.

In this chapter, we design two implementations of the proposed framework, i.e., task-free

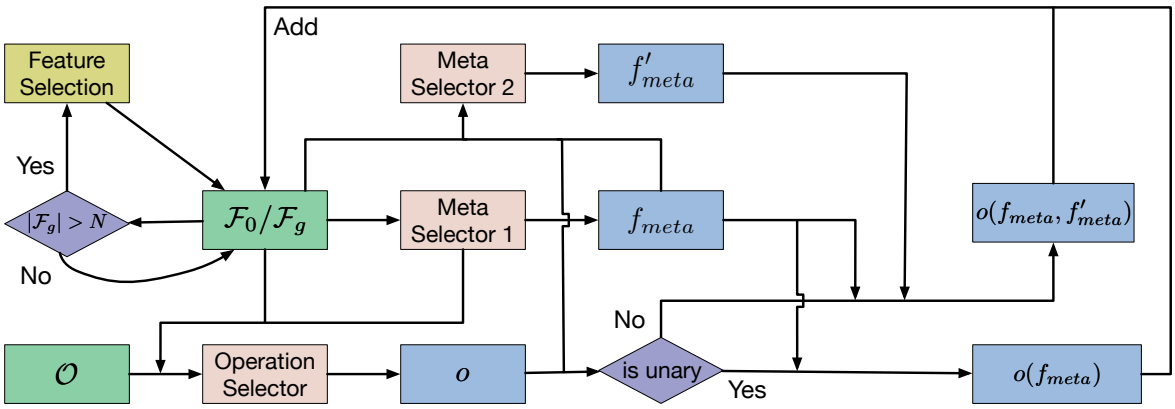


Figure 4.2: The framework of iterative feature generation. In each iteration, we select two meta features and one operation to generate a feature f_g. $\mathcal{F}_0$ is initialized by $\mathcal{F}_g$ of the last iteration, and $\mathcal{F}_g$ is updated by $\mathcal{F}_g = \mathcal{F}_0 \cup \{f_g\}$.

iterative feature generation (TIFG) and reinforced iterative feature generation (RIFG).

Task-Free Iterative Feature Generation

We first design two task-free utility scores, and then leverage the two scores to implement the iterative feature generation framework in a task-free way.

Utility Score

The two vital challenges in the proposed framework are 1) how to select meta features and operation and 2) how to evaluate the generated feature set. To tackle these two challenges, we need to quantify the usefulness of feature, operation, and feature set. In the task-free scenario, we propose two utility scores based on mutual information [16]. Formally, for two

features f_1 and f_2, the mutual information can be calculated by:

$$I(f_1, f_2) = \sum_{x_1 \in X_1} \sum_{x_2 \in X_2} p(x_1, x_2) log(\frac{p(x_1, x_2)}{p(x_1)p(x_2)}) \tag{4.2}$$

where X_1, X_2 are corresponding sample vectors of f_1 and f_2, $p(x_1, x_2)$ is the joint distribution of X_1 and X_2, and $p(x_1)$ and $p(x_2)$ are the marginal distributions of X_1 and X_2 respectively.

Mutual information quantifies how much information about one variable we can infer from the other variable. For a feature, if its mutual information with other features is small, it means its information is difficult to infer from other features, and thus it is of high usefulness for downstream tasks. Similarly, if its mutual information with the target label is large, it means the label can be easily inferred by this feature, and thus this feature is useful. To conclude, a high utility score for a feature requires small mutual information with other features and large mutual information with the target label.

Formally, given a feature set $\mathcal{F}$ and a target label y, we define the utility score of a feature f which belongs to $\mathcal{F}$ as:

$$U_1(f|\mathcal{F}, y) = -\frac{1}{|\mathcal{F}|} \sum_{f' \in \mathcal{F}} I(f, f') + I(f, y) \tag{4.3}$$

For a feature set, if the mutual information between features is large, it means features can infer each other's information, and thus the feature set is redundant. If its features' mutual information with the target label is small, it means these features are useless to predict the label. To conclude, a high utility score requires small mutual information of features and large mutual information with the target label.

Formally, given a feature set $\mathcal{F}$ and a target label y, we define the utility of the $\mathcal{F}$ is as:

$$U_2(\mathcal{F}|y) = -\frac{1}{|\mathcal{F}|^2}\sum_{f_1,f_2\in\mathcal{F}} I(f_1,f_2) + \frac{1}{|\mathcal{F}|}\sum_{f\in\mathcal{F}} I(f,y) \tag{4.4}$$

Task-Free Iterative Feature Generation Algorithm

Based on the proposed feature generation framework and utility scores, we implement a task-free iterative feature generation (TIFG) algorithm. In each iteration, given the original feature set $\mathcal{F}_0$, operation set $\mathcal{O}$ and target label y, we first select the meta feature with the largest utility:

$$f_{meta} = \text{argmax}_{f\in\mathcal{F}_g} U_1(f|\mathcal{F}_0, y) \tag{4.5}$$

And then, we select the operation o and the second meta feature f'_{meta} by traversing $\mathcal{O}$ and $\mathcal{F}_0$:

$$o, f'_{meta} = \text{argmax}_{o\in\mathcal{O}, f\in\mathcal{F}_0} U_2(\mathcal{F}_0 \cup \{o(f_{meta}, f)\}|y) \tag{4.6}$$

Finally, we derive the generated feature by: $f_g = o(f_{meta}, f'_{meta})$ and the generated feature set $\mathcal{F}_g = \mathcal{F}_0 \cup f_g$. We evaluate the utility of $\mathcal{F}_g$ by $U_2(\mathcal{F}_g|y)$. The $\mathcal{F}_g$ with the largest utility score over all iterations is our optimal generated feature set.

Additionally, in each iteration, to control the size of $\mathcal{F}_g$, we apply the feature selection technology. We define two thresholds, i.e., maximized size threshold M_{max} and minimized size threshold M_{min}. When the size of $\mathcal{F}_g$ is larger than $|\mathcal{F}_0| * M_{max}$, we reduce its size by ranking and selecting the top K ($K = |\mathcal{F}_0| * M_{min}$) features by utility score:

$$\mathcal{F}_g = \{\text{TopK}_{f\in\mathcal{F}_g} U_1(f|\mathcal{F}_g, y)\} \tag{4.7}$$

The pseudocode as well as more details about the TIFG algorithm can be found in the Appendix.

Reinforcement Learning based Iterative Feature Generation

Though the task-free iterative method is easy to implement, it doesn't consider the long-term benefit. To tackle this problem, we formulate the feature generation process with Markov decision process (MDP) and solve it by reinforcement learning.

Markov Decision Process

Given a feature set $\mathcal{F}_0$, the feature generation consists of three decision making processes, corresponding to the selection of meta feature, operation and the second meta feature.

MDP for Meta Feature Selection

The problem of selecting the meta feature from the original feature set can be formulated by using the Markov decision process (MDP), which is defined by a tuple $\mathbf{M}^0 = \{\mathcal{S}^0, \mathcal{A}^0, \mathcal{R}^0, \mathcal{P}^0\}$. Here, $\mathcal{S}^0$ and $\mathcal{A}^0$ are state space and action space, reward function $\mathcal{R}^0 : \mathcal{S}^0 \times \mathcal{A}^0 \to \mathbb{R}$ maps a state-action pair to a scalar,and $\mathcal{P}^0$: $\mathcal{S}^0 \times \mathcal{A}^0 \times \mathcal{S}^0 \to \mathbb{R}$ is the probability to transit from a state-action pair to the next state. More specifically,

1. **State** reflects the status of the meta feature selection. The initial state $s_0^0 = Rep(\mathcal{F}_0)$, as all meta selection starts from the original feature set. Here, *Rep* is a representation method that compress a matrix to a vector. We will detail the *Rep* method in the next

section. At the t-th iteration, $s_t^0 = Rep(\mathcal{F}_{0,t})$, where $\mathcal{F}_{0,t}$ is the original feature set at iteration t.

2. **Action** is defined as the selected meta feature $a_t^0 = f_{meta,t}$.

3. **Transition** describes the probability of the next state s_{t+1}^0 based on the current s_{t+1}^0 and the current action a_t^0. Since our MDP is deterministic, the transition probability $\mathcal{P}^0(s_t^0, a_t^0, s_{t+1}^0) \equiv 1$.

4. **Reward** is the incentive to inspire the meta feature selection. Here, we adapt the utility score from Equation 4.3:

$$\mathcal{R}^0(s_t^0, a_t^0) = U_1(f_{meta,t}|\mathcal{F}_{0,t}, y) \tag{4.8}$$

where $s_t^0 = Rep(\mathcal{F}_{0,t})$, $a_t^0 = f_{meta,t}$.

MDP for Operation Selection

The problem of selecting an operation from the operation set can be formulated by using the Markov decision process (MDP), which is defined by a tuple $\mathbf{M}^1 = \{\mathcal{S}^1, \mathcal{A}^1, \mathcal{R}^1, \mathcal{P}^1\}$. More specifically,

1. **State** reflects the status of the operation selection. At the t-th iteration, $s_t^1 = Rep(\mathcal{F}_{0,t}) \bigoplus Rep(f_{meta,t})$, where $\mathcal{F}_{0,t}$ is the original feature set, $_{meta,t}$ is the selected meta feature at iteration t, $\bigoplus$ is the concatenation of vectors.

2. **Action** is defined as the selected operation $a_t^1 = o_t$.

3. **Transition** describes the probability of the next state s_{t+1}^1 based on the current s_{t+1}^1

and the current action a_t^0. Since our MDP is deterministic, the transition probability $\mathcal{P}^1(s_t^1, a_t^1, s_{t+1}^1) \equiv 1$.

4. **Reward** is the incentive to inspire the operation selection. We calculate the reward for the operation selection at the end of iteration t, after we get the generated feature $f'_{g,t}$. Moreover, we use the increment of utility to evaluate the state-action pair:

$$\mathcal{R}^1(s_t^1, a_t^1) = U_2(\mathcal{F}_{g,t}|y) - U_2(\mathcal{F}_{0,t}|y) \tag{4.9}$$

where $\mathcal{F}_{g,t} = \mathcal{F}_{0,t} \cup \{f_{g,t}\}$, $s_t^1 = Rep(\mathcal{F}_{0,t} \cup \{f_{meta,t}\})$, $a_t^1 = o_t$, and $f_{g,t} = o_t(f_{meta,t}, f'_{meta,t})$.

MDP for Meta Feature Selection 2

The problem of selecting the second meta feature from the original feature set can be formulated by using the Markov decision process (MDP), which is defined by a tuple $\mathbf{M}^2 = \{\mathcal{S}^2, \mathcal{A}^2, \mathcal{R}^2, \mathcal{P}^2\}$. More specifically,

1. **State** reflects the status of the second meta feature selection. At the t-th iteration, $s_t^2 = Rep(\mathcal{F}_{0,t}) \bigoplus Rep(f_{meta,t}) \bigoplus Rep(o_t)$, where $\mathcal{F}_{0,t}$ is the original feature set, $f_{meta,t}$ is the selected meta feature, and o_t is the selected operation at iteration t.

2. **Action** is defined as the second selected meta feature $a_t^2 = f'_{meta,t}$.

3. **Transition** describes the probability of the next state s_{t+1}^2 based on the current s_{t+1}^2 and the current action a_t^2. Since our MDP is deterministic, the transition probability $\mathcal{P}^2(s_t^2, a_t^2, s_{t+1}^2) \equiv 1$.

4. **Reward** is the incentive to inspire the selection of second meta feature. Here, we incorporate both the utility increment and downstream task evaluation into the reward:

$$\mathcal{R}^2(s_t^2, a_t^2) = U_2(\mathcal{F}_{g,t}|y) - U_2(\mathcal{F}_{0,t}|y) + E_{\mathcal{T}}(\mathcal{F}_g, y) \tag{4.10}$$

where $s_t^2 = Rep(\mathcal{F}_{0,t} \cup \{f_{meta,t}, o_t\})$, $a_t^2 = f'_{meta,t}$, $f_{g,t} = o_t(f_{meta,t}, f'_{meta,t})$, $\mathcal{F}_{g,t} = \mathcal{F}_{0,t} \cup \{f_{g,t}\}$, and $E_{\mathcal{T}}$ is the performance metric of the downstream task $\mathcal{T}$.

Reinforcement Learning Policy Training

We use the Deep Q Network (DQN) [78] to solve the three MDPs. Given state s_t and action a_t, the Q-value is updated by:

$$Q(s_t, a_t) = \mathcal{R}(s_t, a_t) + \gamma * \max_{a_{t+1}} Q_(s_{t+1}, a_{t+1}) \tag{4.11}$$

where Q is a deep neural network whose inputs are state and action and output is a scalar. The optimal policy of DQN is defined as:

$$\pi^*(a_t|s_t) = \text{argmax}_a Q(s_t, a) \tag{4.12}$$

Prioritized Experience Replay

At time t, after making the decision by Q network, we can obtain a tuple (s_t, a_t, $R(s_t, a_t)$, s_{t+1}) consisting state, action, reward, and the next reward. In the experience replay technique [87], we store these tuples into a memory. The memory is updated by taking the latest tuples and dropping the earliest tuples. In training the Q network, we sample a batch of tuples from the replay memory and apply gradient descendent optimization by Equation 4.11. The traditional experience replay technique treats all tuples equal, and thus the training batch is sampled randomly.

However, in the feature generation problem, a higher reward means more utility of the state, and we should pay more attention to it. In this chapter, we propose a ranked base prioritized experience replay (PER) method. In the PER, we rank each tuple in the experience replay memory by their reward $R(s_t, a_t)$, and calculate their sampling probability by:

$$P(i) = \frac{exp(p_i)}{\sum_{n=1}^{N_e} exp(p_n)} \tag{4.13}$$

where N_e is the size of experience replay memory, p_i is the priority of i-th sample, and $p_i = \frac{1}{rank(i)}$, where $rank(i)$ is the rank of tuple i based on its reward.

Representation of State

In the pre-defined MDPs, the state, which is the input of the Q network, requires a representation method *Rep* that compresses a matrix to a fixed-length vector. In this chapter, we design a descriptive representation method. We use 7 descriptive statistics, i.e., standard deviation, minimum, maximum, Q1 (the first quartile), Q2 (the second quartile), and Q3 (the third quartile). For a matrix $\mathcal{X} \in M \times N$, which has M samples and N features, we first derive its descriptive statistics column-wisely, and get a descriptive matrix $\mathcal{X}' \in M \times 7$; then, we derive the descriptive statistics of $\mathcal{X}'$ row-wisely, and get a descriptive matrix $\mathcal{X}'' \in 7 \times 7$. We flat $\mathcal{X}''$ and get a vector $Rep(\mathcal{X}) \in 49 \times 1$. Similarly, for a sample vector $x \in M \times 1$, we only need to do the description one time and get $Rep(x) \in 7 \times 1$. For an operation, we use one-hot encoding as its *Rep* method.

Reinforced Iterative Feature Generation Algorithm

Based on the formulated MDPs and DQN policy, we propose a reinforced iterative feature generation (RIFG) algorithm. In RIGF, we have N_e episodes, and N_i iterations per episode. In each episode, the original feature set is reset to $\mathcal{F}_0$.We design three DQNs, corresponding to the selection of meta feature, operation and the second meta feature. We derive the state by *Rep*, make decisions based on state by Equation 4.12, and get f_{meta}, o, f'_{meta} sequentially. We train DQNs by Equation 4.11 with prioritized experience replay. We derive the generated feature by $f_g = o(f_{meta}, f'_{meta})$ and the generated feature set $\mathcal{F}_g = \mathcal{F}_0 \cup f_g$. We evaluate the utility of $\mathcal{F}_g$ by $E_{\mathcal{T}}(\mathcal{F}_g, y)$. The $\mathcal{F}_g$ with the highest performance over all episodes is our optimal generated feature set. To control the size of generated feature set, we apply feature selection by Equation 4.7. The pseudocode as well as more details about the RIFG algorithm can be found in the Appendix.

Experimental Results

Experimental Setup

Data Description

We use publicly available datasets from UCI [25], Kaggle [1], LibSVM [12], and OpenML [2]. The statistics of each dataset can be found in Table 4.2. There are two types of datasets, i.e., classification and regression. We use random forest classifier as the downstream task for classification, and random forest regressor as the downstream task for regression.

Evaluation Metrics

In the experiments, we use $F1$-score and 1-RAE (Relative Absolute Error) as evaluation metrics for classification and regression respectively. Formally, the $F1$-score is defined as:

$$F1\text{-score} = 2 * P * R/(P + R) \tag{4.14}$$

where P and R are precision and recall respectively.

The relative absolute error is defined as:

$$RAE = \frac{\sum_{i=1}^{n} |y_i - \check{y}_i|}{\sum_{i=1}^{n} |y_i - \bar{y}_i|} \tag{4.15}$$

where n is the number of data points, $y_i, \check{y}_i, \bar{y}_i$ respectively denote golden standard target values, predicted target values, and the mean of golden standard targets.

Baseline Algorithms

We compare our proposed algorithms, **TIFG** and **RIFG** with the following baselines: (1) **RDG** randomly selects feature-operation-feature pairs for feature generation; (2) **ERT** is a expansion-reduction method, that applies operations to each feature to expand the feature space and selects significant features from the larger space as new features. (3) **AFT** [47] is an enhanced ERT implementation that iteratively explores feature space and adopts a multi-step feature selection relying on L1-regularized linear models. (4) **NFS** [13] mimics feature transformation trajectory for each feature and optimizes the entire feature generation process through reinforcement learning. (5) **TTG** [53] records the feature generation process using a transformation graph, then uses reinforcement learning to explore the graph to determine

the best feature set.

Implementation

The operation set consists of *square root, square, cosine, sine, tangent, exp, cube, log, reciprocal, sigmoid, plus, subtract, multiply, divide*. When the number of generated features reaches twice of the original feature set size, we performed feature selection to control feature size. In TIFG, for a original feature set with N features, we set the iteration number as $5 * N$, the threshold for applying feature selection as $2 * N$, and the feature size after feature selection as $1.5 * N$. In RIFG, for a original feature set with N features, we set the episode number as 200, the iteration number per episode as $2 * N$, the threshold for applying feature selection as $1.5 * N$, and the feature size after feature selection as N. For the DQNs, we design a two-layer ReLU network with 64 and 16 nodes in the first and second layer. In the training of DQN, we use ϵ-greedy with $\epsilon = 0.95$, AdamOptimizer with a learning rate of 0.01, and we set the experience replay memory as 32, the batch size as 8.

Overall Performance

We apply the proposed algorithms and baseline algorithms to 24 datasets and get the performance comparison in Table 4.2. As we can see from the table, our algorithm RIFG outperform all baselines on most datasets. On "Credit Default" and "SpamBase", the performance of our method is relative worse but still comparable with the best performance. This may be due to the personalized feature crossing strategy in RIFG, as well as its consideration of feature-feature distinctions in the generation process. The significant advantage of RIRG over RDG reveals that the superiority of our method is not due to randomness. Moreover, the other proposed algorithm TIFG achieves comparable performance with the

Table 4.2: Overall performance comparison. 'C' for classification and 'R' for regression.

Dataset	Source	C/R	Samples	Features	RDG	ERG	AFT	NFS	TTG	TIFG	RIFG
Higgs Boson	UCIrvine	C	50000	28	0.683	0.674	0.711	0.715	0.705	0.711	**0.718**
Amazon Employee	Kaggle	C	32769	9	0.744	0.740	0.943	0.935	0.806	0.887	**0.945**
PimaIndian	UCIrvine	C	768	8	0.693	0.703	0.736	0.762	0.747	0.752	**0.772**
SpectF	UCIrvine	C	267	44	0.790	0.748	0.775	0.876	0.788	0.848	**0.878**
SVMGuide3	LibSVM	C	1243	21	0.703	0.747	0.829	0.831	0.766	0.796	**0.843**
German Credit	UCIrvine	C	1001	24	0.695	0.661	0.751	0.765	0.731	0.762	**0.770**
Credit Default	UCIrvine	C	30000	25	0.798	0.752	0.799	0.799	**0.809**	0.795	0.804
Messidor_features	UCIrvine	C	1150	19	0.673	0.635	0.678	0.746	0.726	0.747	**0.751**
Wine Quality Red	UCIrvine	C	999	12	0.599	0.611	0.658	0.666	0.647	0.668	**0.679**
Wine Quality White	UCIrvine	C	4900	12	0.552	0.587	0.673	0.679	0.638	0.676	**0.684**
SpamBase	UCIrvine	C	4601	57	0.951	0.931	0.951	0.955	**0.961**	0.954	0.956
AP-omentum-ovary	OpenML	C	275	10936	0.711	0.705	0.783	0.804	0.795	0.804	**0.816**
Lymphography	UCIrvine	C	148	18	0.654	0.638	0.833	0.859	0.846	0.855	**0.867**
Ionosphere	UCIrvine	C	351	34	0.919	0.926	0.827	0.949	0.938	0.950	**0.958**
Bikeshare DC	Kaggle	R	10886	11	0.483	0.571	0.670	0.675	0.659	0.673	**0.680**
Housing Boston	UCIrvine	R	506	13	0.605	0.617	0.641	0.665	0.658	0.660	**0.679**
Airfoil	UCIrvine	R	1503	5	0.737	0.732	0.774	0.771	0.783	0.785	**0.795**
Openml_618	OpenML	R	1000	50	0.415	0.427	0.665	0.640	0.587	0.666	**0.672**
Openml_589	OpenML	R	1000	25	0.638	0.560	0.672	0.711	0.682	0.717	**0.741**
Openml_616	OpenML	R	500	50	0.448	0.372	0.585	0.593	0.559	0.589	**0.605**
Openml_607	OpenML	R	1000	50	0.579	0.406	0.658	0.675	0.639	0.668	**0.677**
Openml_620	OpenML	R	1000	25	0.575	0.584	0.663	0.698	0.656	0.703	**0.709**
Openml_637	OpenML	R	500	50	0.561	0.497	0.564	0.581	0.575	0.574	**0.586**
Openml_586	OpenML	R	1000	25	0.595	0.546	0.687	0.748	0.704	0.750	**0.769**

state-of-the-art algorithm TTG, indicating that even generating features in a task-free way can still derive satisfactory features with careful design of utility scores.

Related Work

There are two major categories of automated feature generation methods, i.e., deep learning based methods and transformation graph based methods [52]. Specifically, the deep learning based methods can be further grouped into three sub-categories [20, 89], i.e., factorization machine based methods, cross operation based methods and embedded methods. Factorization machine methods could effectively capture the low-order interactions between features

[14, 9, 49]. These methods were implicit and had difficulty to capture high-order interactions [15]. Cross operation such as [73, 88] captured high-order feature interactions. *Liu et al.* gave a definition of interpretation inconsistency in deep neural network, and proposed a novel method called CrossGO, which selected useful cross features according to the interpretation inconsistency [72]. *Luo et al.* proposed successive mini-batch gradient descent and multi-granularity discretization to further improve efficiency and effectiveness, while ensuring simplicity so that no machine learning expertise or tedious hyper-parameter tuning was required [73]. There were also other embedded methods like gradient boost machine [32] and group lasso [76] which built useful features in the process of model training. *Dong et al.* combined feature selection and generation into a transformation graph and optimize the them jointly [21]. However, these methods usually required complex optimization procedures and thus had difficulties to deal with large scale datasets. The transformation graph based methods could explore the feature space effectively [108, 53, 7]. One strategy to search the transformation graph is to greedily and randomly generate and select feature [23, 51]. However, these methods suffered from inefficiency due to the feature explosion with searching depth [50]. Additionally, there was a series of heuristic methods that constructed and searched useful features in the early state, e.g., interactive generation [35] and FICUS [75]. These methods were easy to implement but could hardly achieve satisfactory performance. Moreover, some dimension reduction methods such as SVM [99, 54] and PCA [28, 91] can also be viewed as feature generation methods.

Conclusion Remarks

In this chapter, to automate the feature generation process, we propose an iterative feature generation framework. We design two implementations of the framework, i.e., task-free

iterative feature generation(TIFG) and reinforced iterative feature generation (RIFG). We We design extensive experiments and we find that TIFG can achieve comparable performance with state-of-the-art algorithms while RIFG can outperform almost all baselines on most datasets. One direction to improve the research is to introduce more advanced reinforcement technique, e.g., TRPO and PPO to solve MDPs in the framework. Another potential research direction is to adapt the framework to streaming data or online scenarios to solve real-world challenging problems.

CHAPTER 5: CONCLUSION AND FUTURE RESEARCH DIRECTIONS

Conclusion

In this dissertation, I studied how to optimize the feature space via feature selection and feature generation. Specifically, I discussed this topic from three perspectives:

1. Multi-Agent Reinforcement Feature Selection. I reformulated the feature selection problem with a multi-agent reinforcement learning framework. One agent was assigned to each feature, the actions of these feature agents were to select or deselect their corresponding features, and the environment was defined as the characteristics of the selected feature subspace. I developed three different methods to do state representation. I also proposed three methods to accelerate the feature subspace exploration.

2. Single-Agent Reinforcement Feature Selection. I reformulated the feature selection problem with a single agent reinforcement learning framework. I designed a traverse strategy, where one single agent visits each feature one by one to decide its selection. I designed two policies, i.e., one behavior policy and one target policy to implement the framework. I decomposed the sampling weight in the off-policy Monte Carlo method into an incremental format, and proposed an early stopping criteria to assure the quality of training samples as well as stopping the meaningless traverse by behavior policy.

3. Automated Iterative Feature Generation. I proposed an iterative feature generation framework to generate effective and relevant features from the original feature set. I implemented the proposed framework and propose two algorithms, i.e., task-free

iterative feature generation (TIFG) and reinforced iterative feature generation (RIFG). I designed a prioritized experience replay method and a descriptive representation method to better train the RIFG algorithm.

Future Research Directions

D1: Federated Feature Engineering for Distributed Data. In real world, data may be stored in different repositories located in different regions, and gathering these distributed data into a central server for feature engineering is not always practical because of the limitations on data confidentiality, transportation expense and network bandwidth. To tackle these challenges, there exist three core tasks: (1) developing a distributed learning framework to efficiently organize and learn from data in different repositories; (2) designing privacy-preserving calculation strategies for the distributed learning framework; (3) proposing parallel learning schemes to accelerate the feature engineering process.

D2: Real-Time Feature Engineering for Streaming Data. While the research on automated feature engineering shows the effectiveness and efficiency on static data, streaming data is another crucial scenario for real-world deployment. The potential challenges of automated feature engineering on streaming data include: (1) there may exist distribution shift in the streaming data; (2) the training speed of feature engineering may be slower than the streaming speed; (3) new features may be introduced and old features may be eliminated. To tackle these challenges, there exist three core tasks: (1) developing incremental updating strategies to balance the impact of old data and new data, and to capture the distribution changes; (2) designing adaptive sampling windows to synchronize the training of feature engineering and the data streaming; (3) proposing new feature engineer pipelines which can handle cold-start feature space changes.

D3: Causality in Feature Engineering. Traditional feature engineering algorithms select or generate features based on the relevance between features and the target label, ignoring causal relationships. However, causal features imply the underlying mechanism of the dataset, and thus are more reliable, interpretable, and robust in predicting the target label. The integration of causal analysis with feature engineering frameworks may be studied from the following perspectives: (1) causal feature identification which discovers features that are causal factors of the target label; (2) new feature engineering algorithms that take advantage of the discovered causality; (3) casual feature engineering across different datasets with different distributions.

LIST OF REFERENCES

[1] Kaggle. `https://www.kaggle.com`.

[2] Openml. `https://www.openml.org`.

[3] Nathan Baker, Frank Alexander, Timo Bremer, Aric Hagberg, Yannis Kevrekidis, Habib Najm, Manish Parashar, Abani Patra, James Sethian, Stefan Wild, et al. Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence. Technical report, USDOE Office of Science (SC), Washington, DC (United States), 2019.

[4] Richard Bellman and Robert Kalaba. Dynamic programming and statistical communication theory. *Proceedings of the National Academy of Sciences of the United States of America*, 43(8):749, 1957.

[5] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[6] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[7] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[8] Jock A. Blackard. Kaggle forest cover type prediction. [EB/OL], 2015. `https://www.kaggle.com/c/forest-cover-type-prediction/data`.

[9] Mathieu Blondel, Akinori Fujino, Naonori Ueda, and Masakazu Ishihata. Higher-order factorization machines. In *Advances in Neural Information Processing Systems*, pages 3351–3359, 2016.

[10] Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):1–37, 2011.

[11] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[12] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

[13] Xiangning Chen, Qingwei Lin, Chuan Luo, Xudong Li, Hongyu Zhang, Yong Xu, Yingnong Dang, Kaixin Sui, Xu Zhang, Bo Qiao, et al. Neural feature search: A neural architecture for automated feature engineering. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 71–80. IEEE, 2019.

[14] Chen Cheng, Fen Xia, Tong Zhang, Irwin King, and Michael R Lyu. Gradient boosting factorization machines. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 265–272, 2014.

[15] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[16] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[17] Francisco Cruz, Sven Magg, Yukie Nagai, and Stefan Wermter. Improving interactive reinforcement learning: What makes a good teacher? *Connection Science*, 30(3):306–325, 2018.

[18] Francisco Cruz, Johannes Twiefel, Sven Magg, Cornelius Weber, and Stefan Wermter. Interactive reinforcement learning through speech guidance in a domestic scenario. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2015.

[19] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[20] Shifei Ding, Hong Zhu, Weikuan Jia, and Chunyang Su. A survey on feature extraction for pattern recognition. *Artificial Intelligence Review*, 37(3):169–180, 2012.

[21] Guozhu Dong and Huan Liu. *Feature engineering for machine learning and data analytics*. CRC Press, 2018.

[22] David L Donoho. For most large underdetermined systems of linear equations the minimal 1-norm solution is also the sparsest solution. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 59(6):797–829, 2006.

[23] Ofer Dor and Yoram Reich. Strengthening learning algorithms by feature discovery. *Information Sciences*, 189:176–190, 2012.

[24] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[25] Dheeru Dua and Casey Graff. Uci machine learning repository [http://archive. ics. uci. edu/ml]. irvine, ca: University of california. *School of Information and Computer Science*, 25:27, 2019.

[26] Wei Fan, Kunpeng Liu, Hao Liu, Pengyang Wang, Yong Ge, and Yanjie Fu. Autofs: Automated feature selection via diversity-aware interactive reinforcement learning. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1008–1013. IEEE, 2020.

[27] Seyed Mehdi Hazrati Fard, Ali Hamzeh, and Sattar Hashemi. Using reinforcement learning to find an optimal set of features. *Computers & Mathematics with Applications*, 66(10):1892–1904, 2013.

[28] Imola K Fodor. A survey of dimension reduction techniques. Technical report, Lawrence Livermore National Lab., CA (US), 2002.

[29] George Forman. An extensive empirical study of feature selection metrics for text classification. *Journal of machine learning research*, 3(Mar):1289–1305, 2003.

[30] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. Deap: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13(Jul):2171–2175, 2012.

[31] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

[32] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[33] Jerome H Friedman. Fast sparse regression and classification. *International Journal of Forecasting*, 28(3):722–738, 2012.

[34] Nicolo Fusi, Rishit Sheth, and Melih Elibol. Probabilistic matrix factorization for automated machine learning. *Advances in neural information processing systems*, 31:3348–3357, 2018.

[35] Mark Lee Gillenson. *The interactive generation of facial images on a CRT using a heuristic strategy.* The Ohio State University, 1974.

[36] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[37] Pablo M Granitto, Cesare Furlanello, Franco Biasioli, and Flavia Gasperi. Recursive feature elimination with random forest for ptr-ms analysis of agroindustrial products. *Chemometrics and Intelligent Laboratory Systems*, 83(2):83–90, 2006.

[38] Dihua Guo, Hui Xiong, Vijay Atluri, and Nabil Adam. Semantic feature selection for object discovery in high-resolution remote sensing imagery. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 71–83. Springer, 2007.

[39] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

[40] H. A. Guvenir, B. Acar, G. Demiroz, and A. Cekin. A supervised machine learning algorithm for arrhythmia analysis. In *Computers in Cardiology 1997*, pages 433–436, 1997.

[41] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[42] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422, 2002.

[43] Mark A Hall. Feature selection for discrete and numeric class machine learning. 1999.

[44] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[45] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical learning with sparsity: the lasso and generalizations*. Chapman and Hall/CRC, 2019.

[46] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.

[47] Franziska Horn, Robert Pack, and Michael Rieger. The autofeat python library for automated feature engineering and selection. *arXiv preprint arXiv:1901.07329*, 2019.

[48] Ian T Jolliffe. Principal component analysis: a beginner's guide—ii. pitfalls, myths and extensions. *Weather*, 48(8):246–253, 1993.

[49] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 43–50, 2016.

[50] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*, pages 1–10. IEEE, 2015.

[51] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Explorekit: Automatic feature generation and selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 979–984. IEEE, 2016.

[52] Samina Khalid, Tehmina Khalil, and Shamila Nasreen. A survey of feature selection and feature extraction techniques in machine learning. In *2014 science and information conference*, pages 372–378. IEEE, 2014.

[53] Udayan Khurana, Horst Samulowitz, and Deepak Turaga. Feature engineering for predictive modeling using reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[54] Sang-Ki Kim, Youn Jung Park, Kar-Ann Toh, and Sangyoun Lee. Svm-based feature extraction for face recognition. *Pattern Recognition*, 43(8):2871–2881, 2010.

[55] YeongSeog Kim, W Nick Street, and Filippo Menczer. Feature selection in unsupervised learning via evolutionary search. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 365–369. ACM, 2000.

[56] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[57] W Bradley Knox, Peter Stone, and Cynthia Breazeal. 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*, pages 65–66, 2013.

[58] Ron Kohavi and George H John. Wrappers for feature subset selection. *Artificial intelligence*, 97(1-2):273–324, 1997.

[59] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[60] Mark Kroon and Shimon Whiteson. Automatic feature selection for model-based reinforcement learning in factored mdps. In *Machine Learning and Applications, 2009. ICMLA'09. International Conference on*, pages 324–330. IEEE, 2009.

[61] Riccardo Leardi. Genetic algorithms in feature selection. In *Genetic algorithms in molecular modeling*, pages 67–86. Elsevier, 1996.

[62] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. *ACM Computing Surveys (CSUR)*, 50(6):1–45, 2017.

[63] Yitan Li, Linli Xu, Fei Tian, Liang Jiang, Xiaowei Zhong, and Enhong Chen. Word embedding revisited: A new representation learning and explicit matrix factorization perspective. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[64] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1754–1763, 2018.

[65] HL Liao, QH Wu, and L Jiang. Multi-objective optimization by reinforcement learning for power system dispatch and voltage stability. In *Innovative Smart Grid Technologies Conference Europe (ISGT Europe), 2010 IEEE PES*, pages 1–8. IEEE, 2010.

[66] Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. Efficient large-scale fleet management via multi-agent deep reinforcement learning. *arXiv preprint arXiv:1802.06444*, 2018.

[67] Long Ji Lin. Programming robots using reinforcement learning and teaching. In *AAAI*, pages 781–786, 1991.

[68] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[69] Jun S Liu, Rong Chen, and Wing Hung Wong. Rejection control and sequential importance sampling. *Journal of the American Statistical Association*, 93(443):1022–1031, 1998.

[70] Kunpeng Liu, Yanjie Fu, Pengfei Wang, Le Wu, Rui Bo, and Xiaolin Li. Automating feature subspace exploration via multi-agent reinforcement learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 207–215, 2019.

[71] Kunpeng Liu, Yanjie Fu, Le Wu, Xiaolin Li, Charu Aggarwal, and Hui Xiong. Automated feature selection: A reinforcement learning perspective. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[72] Zhaocheng Liu, Qiang Liu, and Haoli Zhang. Automatically learning feature crossing from model interpretation for tabular data. 2019.

[73] Yuanfei Luo, Mengshuo Wang, Hao Zhou, Quanming Yao, Wei-Wei Tu, Yuqiang Chen, Wenyuan Dai, and Qiang Yang. Autocross: Automatic feature crossing for tabular data in real-world applications. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1936–1945, 2019.

[74] Steven N MacEachern, Merlise Clyde, and Jun S Liu. Sequential importance sampling for nonparametric bayes models: The next generation. *Canadian Journal of Statistics*, 27(2):251–267, 1999.

[75] Shaul Markovitch and Dan Rosenstein. Feature generation using general constructor functions. *Machine Learning*, 49(1):59–98, 2002.

[76] Lukas Meier, Sara Van De Geer, and Peter Bühlmann. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(1):53–71, 2008.

[77] Andriy Mnih and Russ R Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2008.

[78] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[79] Patrenahalli M. Narendra and Keinosuke Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on computers*, (9):917–922, 1977.

[80] Chong Peng, Zhao Kang, Yunhong Hu, Jie Cheng, and Qiang Cheng. Nonnegative matrix factorization with integrated graph and feature learning. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(3):1–29, 2017.

[81] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence*, 27(8):1226–1238, 2005.

[82] Peng Peng, Ying Wen, Yaodong Yang, Quan Yuan, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets: Emergence of human-

level coordination in learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2017.

[83] Maxim Raginsky, Alexander Rakhlin, and Matus Telgarsky. Non-convex learning via stochastic gradient langevin dynamics: a nonasymptotic analysis. *arXiv preprint arXiv:1702.03849*, 2017.

[84] Steffen Rendle. Factorization machines. In *2010 IEEE International conference on data mining*, pages 995–1000. IEEE, 2010.

[85] Yvan Saeys, Iñaki Inza, and Pedro Larrañaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

[86] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

[87] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[88] Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 255–262, 2016.

[89] Parikshit Sondhi. Feature construction methods: a survey. *sifaka. cs. uiuc. edu*, 69:70–71, 2009.

[90] Milos Stankovic. Multi-agent reinforcement learning. In *Neural Networks and Applications (NEUREL), 2016 13th Symposium on*, pages 1–1. IEEE, 2016.

[91] Dmitry Storcheus, Afshin Rostamizadeh, and Sanjiv Kumar. A survey of modern questions and challenges in feature extraction. In *Feature Extraction: Modern Questions and Challenges*, pages 1–18. PMLR, 2015.

[92] Halit Bener Suay and Sonia Chernova. Effect of human guidance and state space size on interactive reinforcement learning. In *2011 Ro-Man*, pages 1–6. IEEE, 2011.

[93] V Sugumaran, V Muralidharan, and KI Ramachandran. Feature selection using decision tree and classification through proximal support vector machine for fault diagnostics of roller bearing. *Mechanical systems and signal processing*, 21(2):930–942, 2007.

[94] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[95] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

[96] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[97] Lisa Torrey and Matthew Taylor. Teaching on a budget: Agents advising agents in reinforcement learning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1053–1060, 2013.

[98] Peter Van Der Putten and Maarten van Someren. Coil challenge 2000: The insurance company case. Technical report, Technical Report 2000–09, Leiden Institute of Advanced Computer Science . . . , 2000.

[99] Vladimir N Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

[100] Dongjie Wang, Pengyang Wang, Kunpeng Liu, Yuanchun Zhou, Charles E Hughes, and Yanjie Fu. Reinforced imitative graph representation learning for mobile user profiling: An adversarial training perspective. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4410–4417, 2021.

[101] Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. Intellilight: A reinforcement learning approach for intelligent traffic light control. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2496–2505, 2018.

[102] Tengyang Xie, Yifei Ma, and Yu-Xiang Wang. Towards optimal off-policy evaluation for reinforcement learning with marginalized importance sampling. In *Advances in Neural Information Processing Systems*, pages 9668–9678, 2019.

[103] Jihoon Yang and Vasant Honavar. Feature subset selection using a genetic algorithm. In *Feature extraction, construction and selection*, pages 117–136. Springer, 1998.

[104] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. *arXiv preprint arXiv:1802.05438*, 2018.

[105] Yiming Yang and Jan O Pedersen. A comparative study on feature selection in text categorization. In *Icml*, volume 97, pages 412–420, 1997.

[106] Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 856–863, 2003.

[107] Yang Yu. Towards sample efficient reinforcement learning. In *IJCAI*, pages 5739–5743, 2018.

[108] Jianyu Zhang, Jianye Hao, Françoise Fogelman-Soulié, and Zan Wang. Automatic feature engineering by deep reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2312–2314, 2019.

[109] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. Deep reinforcement learning for page-wise recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 95–103, 2018.