Self-optimizing Feature Generation via Categorical Hashing Representation and Hierarchical Reinforcement Crossing

Wangyang Ying[1*], Dongjie Wang[2*], Kunpeng Liu[3], Leilei Sun[4], Yanjie Fu[1†]
[1] *School of Computing and Augmented Intelligence, Arizona State University, Tempe, USA*
[2] *Department of Computer Science, University of Central Florida, Orlando, USA*
[3] *Department of Computer Science, Portland State University, Portland, USA*
[4] *Department of Computer Science, Beihang University, Beijing, China*
{wangyang.ying, yanjie.fu}@asu.edu, dongjie.wang@ucf.edu, kunpeng@pdx.edu, leileisun@buaa.edu.cn

***Abstract*—Feature generation aims to generate new and meaningful features to create a discriminative representation space. A generated feature is meaningful when the generated feature is from a feature pair with inherent feature interaction. In the real world, experienced data scientists can identify potentially useful feature-feature interactions, and generate meaningful dimensions from an exponentially large search space, in an optimal crossing form over an optimal generation path. But, machines have limited human-like abilities. We generalize such learning tasks as self-optimizing feature generation. Self-optimizing feature generation imposes several under-addressed challenges on existing systems: meaningful, robust, and efficient generation. To tackle these challenges, we propose a principled and generic representation-crossing framework to solve self-optimizing feature generation. To achieve hashing representation, we propose a three-step approach: feature discretization, feature hashing, and descriptive summarization. To achieve reinforcement crossing, we develop a hierarchical reinforcement feature crossing approach. We present extensive experimental results to demonstrate the effectiveness and efficiency of the proposed method. The code is available at https://github.com/yingwangyang/HRC_feature_cross.git.**

***Index Terms*—Feature Generation, Hierarchical Reinforcement Crossing, Self-optimizing**

I. Introduction

Feature generation (FG) is a technique used in machine learning to combine two or more input features in order to create new, more complex features. The new features are created by taking combinations of the original features, such as their concatenation, products, ratios, or differences, and can help capture interactions or nonlinear relationships between the original features. For example, if we have two input features, "age" and "income", we could create a new feature called "age times income" by taking the product of these two variables. This new feature may be more informative than either of the original features alone, as it captures the joint effect of age and income on the outcome variable. Feature generation can be a powerful tool for improving the predictive performance of machine learning models, particularly in cases where the relationships between the input features and the outcome variable are complex or nonlinear.

There are two major challenges in self-optimizing FG: 1) meaningful and robust generation to improve prediction and fight data bias, 2) automated and efficient generation to mimic human decision behaviors and reduce search space. First, except for a given original feature set, all the principles and mechanisms of FG are unknown: how the optimal target feature set looks like? Which generation path is the best to create a representation space where data patterns are preserved and discriminative? Is the feature-feature crossing form robust enough to fight against outlier or extreme data values? Under such an open and uncertain environment, meaningful and robust generation seeks to answer: how can we automatically perceive implicit feature interaction, generate effective new features, and fight against outlier feature values? Second, the space of FG can exponentially grow. The automated generation suffers from large search space and long calculation time. Greedy strategies are faster but result in local optimal; exhaustive strategies are globally optimized but unrealistically slow. Therefore, automated and efficient generation aims to answer: how can we navigate generation paths with a balance between time costs and global optimum?

As **Figure 1** shows, prior literature can only partially solve the aforementioned challenges. First, this study is related to exhaustive FG, for example, Factorization Machines (FMs) [1]–[3], which capture interactions between features from low orders to high orders in order to generate new features. However, FMs enumerate all feature-feature crossings, and, thus, are inefficient, particularly when a given feature set is large. Second, this study is related to greedy FG, for instance, AutoCross [4], which iterates self-crossing in a previously generated feature set, and always scores and selects the best cross feature based on downstream task performance in each iteration. However, while greedy crossing is efficient, its greedy nature exhibits two limitations: 1) only considers the top-1 cross-feature and ignores other cross features; 2) only considers the reward of the current iteration, instead of the long-term reward. Third, this study is related to reinforcement FG, for instance, GRFG [5], which leverages reinforcement policy to conduct feature-feature crossing. However, existing

*Both author contributed equally to this research.
†Corresponding author

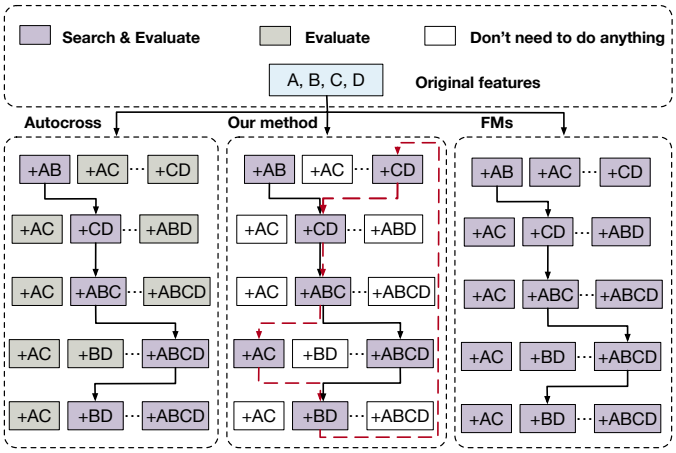


Fig. 1: The comparison of search space. Autocross is efficient, but is prone to local optima because it employs a greedy algorithm in each iteration. FMs enumerate all combinations, can find the global optima but are inefficient. Our method does not require exhaustive enumeration as the crossing policies will trim and limit the search space.

reinforcement RG is not robust to outlier feature values (e.g., extremely big or small values) when crossing two numeric features. Existing studies demonstrate the inability to jointly address meaningful, automated, and efficient generation. As a result, we need a new perspective to derive a novel formulation and solver for self-optimizing FG.

Our Contribution: An Integrated Hashing Representation and Reinforcement Crossing Perspective. We observe that a meaningful new feature is usually generated by a feature pair with statistically significant feature-feature interaction. Detection, localization, and measurement of the latent feature interaction are difficult, not to mention designing an explicit crossing form to augment feature space. We show that reinforcement agents are a great fit for sensing the mechanism-unknown feature-feature interaction. We formulate FG as a self-optimizing framework to achieve meaningful, robust, and efficient generation. We highlight three contributions in our framework: 1) hierarchical reinforcement intelligence can learn generation policies to perceive feature-feature interactions, drive the selections of meta feature and crossed feature, and navigate optimal feature generation path, in a way that we reduce search space and balance between global optimal and greedy efficiency. 2) discretization of feature values before generation is useful because categorical feature crossing can robustilize FG to fight against outlier numeric or continuous feature values to avoid generating anomaly features and introducing bias to future feature generation. 3) the three steps: feature categorization, feature hashing, and descriptive summarization are an efficient strategy to achieve fixed length state representation of a dynamically varying feature space.

Summary of Proposed Approach. Inspired by these findings, this paper develops a principled and generic representation-crossing framework for the FG task by iterating hashing-based categorical feature space state representation and hierarchical reinforcement feature crossing (HRC). The framework has two goals: 1) categorizing, hashing, and describing feature space to achieve outlier feature value robustness and fast state extraction in the representation step; 2) hierarchical reinforcement crossing of meta feature and crossed feature to generate meaningful dimensions in the crossing step. To achieve Goal 1, we propose a three-step approach: feature discretization, feature hashing, and descriptive summarization. In particular, we first discretize all the data table values feature by feature, then hash categorical features into a small feature table, and finally exploit dual (feature-wise and instance-wise) descriptive statistics to extract a fixed-length feature space representation. Feature discretization can eliminate extreme and outlier values, which could later participate in feature crossing and introduce bias into future generations. Unlike one-hot encoding that creates a large feature table, feature hashing can reduce the size of the data table, and help to complete the next step of descriptive summarization in a short time. Descriptive summarization can extract a fixed-length state representation of a large dynamically-varying feature space, because DQN usually only takes fixed-length state representation as inputs. To achieve Goal 2, we develop a hierarchical reinforcement feature crossing approach. This approach has two agents: meta controller and controller. In each iteration, the meta controller selects a meta feature and the controller selects a feature to cross with the meta feature. The two DQNs of the meta controller and controller learn policies to sense and select two categorical features to cross. We leverage mutual information to assess feature-label relevance and feature-feature redundancy, combined with the accuracy of a downstream task as reward quantification to incentivize policy training. Finally, we design extensive experiments to verify the effectiveness and efficiency of the proposed methods.

II. Problem Statement

Cartesian Feature Crossing. Cartesian crossing is to use the Cartesian join operation to cross categorical features and generate new features. For example, if the two original features are marriage={married, single} and salary = {high, low}, then Cartesian join will generate a new feature with four categorical values: {married and high, married and low, single and high, single and low}.

Feature Generation and Key Challenge. The feature generation problem is to generate new features from original features in order to improve feature space and help the downstream model obtain better predictive performance. The key challenge is that the number of potential feature sets is extremely large. For example, given N original features, if the highest order of generated feature is k, then the overall number of original features together with generated features is $C_N^1+\sum_{k=2}^{N} C_N^k = 2^N-1$. With so many features, the potential combination of them, i.e., the number of potential feature sets is $2^{(2^N-1)}$. Aside from the large number of candidate-generated feature sets, we need to find the best feature set from all of the $2^{(2^N-1)}$ potential candidates, which is a time-consuming and computationally intensive process.

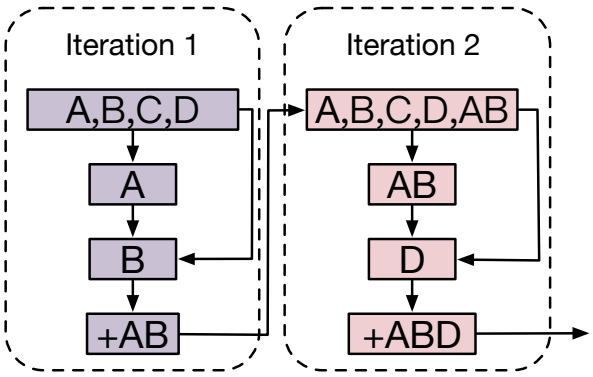


Fig. 2: An example of the iterative HRC process.

Iterative Generation to Quickly Approach An Acceptable Optimal. Iterative approaches can efficiently generate cross-features to reduce search space and computational costs. **Figure 2** shows why iterative generation can allow us to select the most important features and drop the others, and control the number of candidates generated feature sets. Specifically, we initialize the optimal generated feature set by the original feature set. In the first iteration, the strategy selects one feature as a meta feature. The strategy then selects another feature that has the strongest interaction with the selected meta feature in order to generate a new feature, which will be added to the generated feature set. After multiple iterations, we can approach an acceptable optimal generated feature set. AutoCross [4] is also an example study.

Hierarchical Control Structure. To put this strategy into practice, it's critical to carefully control and manage the selection process. Obviously, there are two crucial stages in each iteration: (1) the initial stage selects a meta feature, (2) and the subsequent stage selects another feature to cross with the meta feature. It is important to note that the two stages are not independent and are not parallel. The first stage needs to smartly sense which meta feature can potentially lead to a new and effective dimension based on the existing feature space; the second stage relies on the first stage and aims to identify another feature that exhibits the strongest interaction with the meta feature.

The Self-optimizing Feature Generation Formulation. Formally, given an original feature set and a downstream predictive task, we aim to automate the derivation of a generated feature set from the original feature set to maximize the performance improvement of the downstream predictive task. We formulate this problem as a hierarchical reinforcement crossing task. This framework includes a set of elements: agents, actions, environments, states, and rewards. These elements collaboratively iterate two major steps: 1) feature space representation; 2) crossing policy learning to automate feature generation. Such an interaction-feedback-learning-long-term reward targeting framework is a great fit for solving the joint challenges: 1) the mechanism about how to select two optimal features for crossing is unclear; 2) the generation path needs to trade-off between efficiency and global optimum.

III. Self-optimizing Feature Generation

A. Framework Overview

Figure 3 shows our framework iterates two iteratively-interact components: 1) categorical hashing representation, and 2) hierarchical reinforcement crossing.

The goal of categorical hashing representation is to extract a fixed-length state representation to describe a dynamically varying feature space. We propose a three-step approach for feature space state representation by integrating discretization, hashing, and summarization. In particular, in Step 1 (feature discretization), we first leverage discretization to convert all the feature values into categorical. This step reformulates FG into pairwise categorical feature crossing that is robust against biased feature generation caused by outlier or extreme continuous feature values crossing. Since feature values are categorical, if we want to describe the state of feature space, an intuitive solution is to apply one-hot encoding to categorical features in order to ease the statistics extraction of feature space. However, one-hot encoding can significantly increase the dimensionality of a feature space to describe, making state extraction inefficient. Therefore, in Step 2 (feature hashing), we propose a faster feature hashing-based approach to encode the categorical feature table into a hashing value table without losing accuracy. In Step 3 of descriptive summarization, we integrate both feature-wise and instance-wise descriptive statistics to extract a fixed number of statistics as a fixed-length state representation of the feature space. This is because: although the feature space varies over iterations, reinforcement policy learners (e.g., DQN) only accept a fixed-length state as inputs.

The goal of hierarchical reinforcement crossing is to learn a feature crossing-based operation path to achieve the automated, robust, and fast generation of new meaningful features. For this purpose, we develop a hierarchical reinforcement crossing approach. The reinforcement intelligence is designed for automation: automatically decide how many cross-features to generate in each iteration. The hierarchical agent design includes a meta controller agent and a controller agent to sense feature-feature interaction to select a meta feature and a crossed feature that are the most appropriate for crossing new meaningful dimensions. The policy-driven generation can improve itself from the feedback of previous generations, find the most optimized direction for future generations, and the tradeoff between greedy search-caused local optimal issues and exhaustive search-caused low-efficiency issues.

Finally, we iterate both categorical hashing representation and hierarchical reinforcement crossing to conduct self-optimizing feature generation.

B. Categorical Hashing Representation

Why Categorical Hashing Representation Matters. When feature values are continuous data, crossing two continuous features can be easily compromised to generate biased new features. For example, if a feature includes extremely large or small outlier values, these outlier values can propagate

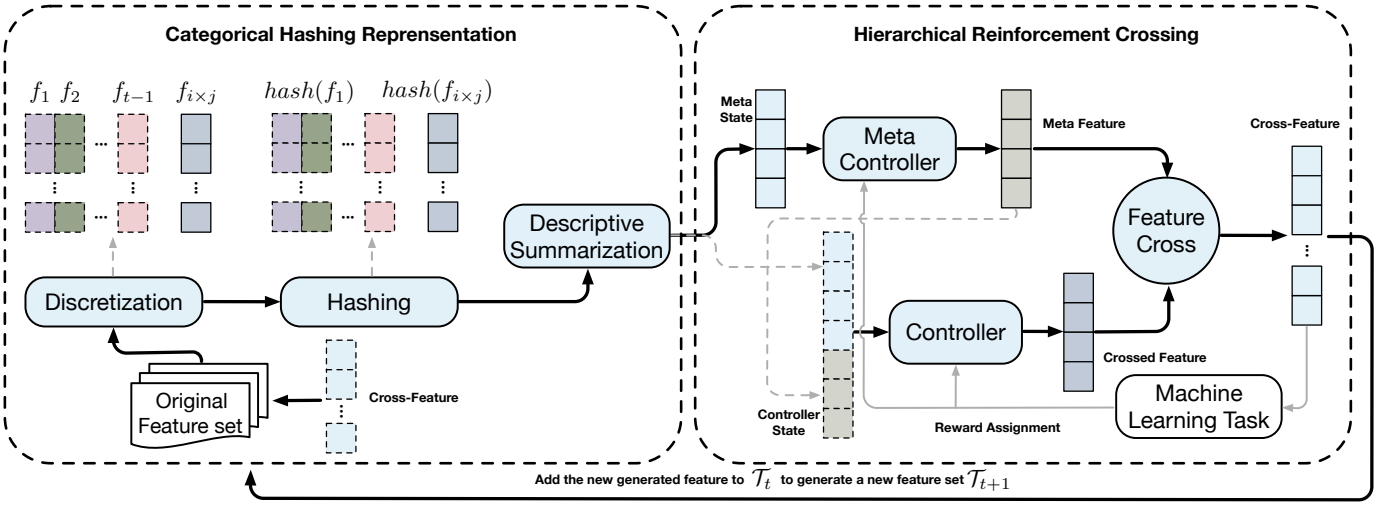


Fig. 3: Overview of the framework. In the first part, we convert a dynamically varying feature space into a fixed-length state using the category hashing representation technique. Based on the state, we create a globally optimal sub-feature space using hierarchical reinforcement crossing in the second part.

bias into generated features. We highlight that feature discretization (binning) can aggregate continuous values into category levels, and reduce the bias propagation of extreme values in feature crossing. However, if we adopt feature discretization and bin all the data into categorical values, we need to equip reinforcement agents with the ability to describe the state of a categorical feature space. Traditional methods, such as neural representation learning or descriptive statistics-based approaches, usually require us to apply one-hot encoding to transform categorical data and ease state representation extraction. Unfortunately, one-hot encoding will significantly increase the dimensionality that we will describe as a fixed-length feature space state vector, thereafter, increase the computation burden of state representation extraction. So, a dilemma is: how can we fight against outlier and extreme values via discretization, meanwhile improving state representation efficiency?

Leveraging the Integrated Power of Discretization, Hashing, and Descriptive Summarization. We found that integrating feature discretization, feature hashing, and descriptive statistics summarization can achieve both robustness against outlier data, and fast state representation of categorical data. Based on our unique insight, we propose a step-by-step testable method that includes three steps. **Figure 3** illustrates the three-step approach of categorical hashing representation.

Step 1: Binning Outliers and Noises. Outliers and noises can introduce and propagate bias in the next and future feature crossing. Binning smooths a continuous data value by consulting its neighborhood, transforming the value into a discrete category, and ensuring that data in the same bin is similar and data in different bins are more distinguishable. Therefore, binning can limit noises or anomaly by creating categorical values with distributions comprising fewer unique values, and robustize the crossing of categorical features. To this end, we integrate hierarchical bottom-up clustering and $\mathcal{X}^2$-distribution to develop an automated binning approach for feature crossing. Since the feature value distribution after binning should be similar to the feature value distribution before binning, we leverage a hierarchical bottom-up clustering idea to create small bins first and then combine small bins with similar distributions into large bins. Our approach includes two stages: 1) the initialization stage; 2) the bottom-up merging stage. In particular, in the initialization stage, we sort the values of a feature according to their numerical values, and then each value of the feature column is treated as a separate bin. In the merging stage, we iterate the following steps: i) we first calculate the chi-square of each pair of adjacent bins; ii) based on the chi-squares, we merge any neighbor bin pair with the minimum chi-square; iii) we repeat i) and ii) until all the chi-squares of the bins are larger than a certain threshold, or the number of bins reaches a minimum number of bins.

Step 2: Feature Hashing. After Step 1, all the features in the feature space are categorical (nominal). To extract the feature space state representation, we need a numeric representation of a categorical feature table that not only preserves feature space patterns, but also eases the statistical summarization of a feature space. An intuitive way is to use the existing one-hot encoding to map the categorical features into a one-hot data table for statistical summarization without losing too much information. However, one-hot encoding will convert a single categorical feature into multiple features, and dimensionality will increase. Moreover, since iterative feature crossing will keep generating more feature value categories, the dimensionality under one-hot encoding and the time costs of computing the feature space state will explode. Our idea is to develop a hashing representation for the categorical features. Since a hashing function can map all categorical data into a smaller fixed set, the use of hashing can greatly reduce the number of categories and prevent a dimensional explosion. Specifically, we use a hash function [6] to convert each feature categorical value to an integer value, and then we reduce the integer value modulo a pre-defined number to get a new smaller number.

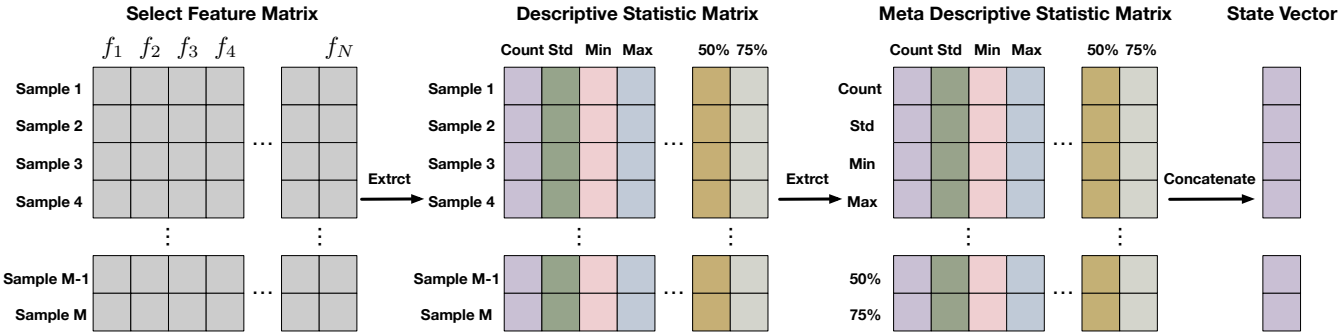


Fig. 4: Representation of generated feature set. We extract descriptive statistics twice from the feature set to obtain a fixed-length state vector. The length of the represented state vector remains fixed regardless of the change in feature numbers.

Step 3: Descriptive Summarization. The feature number (dimensionality) of the generated feature set changes at each iteration. However, the policy network and target network in DQN require the feature space stat representation to be a fixed-length vector at each step. The goal of Step 3 is to derive a fixed-length state representation of the hashed feature space whose dimension changes over time. As shown in **Figure 4** We propose a descriptive statistics-based dual summarization method that includes two steps: 1) row-wise summarization, and 2) column-wise summarization. Here, the eight descriptive statistics include: count, mean, standard, minimum, maximum, and three quartiles (25%,50%,75%). **Figure 4** shows the two-step procedure. Step 1 extracts the eight descriptive statistics of each row in $\mathcal{D}$, and thus, obtains a descriptive statistics matrix with columns as eight statistics and rows as samples. Step 2 extracts the eight descriptive statistics of each column of the descriptive statistics matrix, and obtains a meta descriptive statistics matrix with rows as eight statistics and columns as eight statistics. Finally, we concatenate each column of the meta matrix together as the fixed-length state representation vector with a length of 8x8=64.

C. Hierarchical Reinforcement Crossing

Why Hierarchical Reinforcement Crossing Matters. First, crossing categorical features can generate more explainable and meaningful new dimensions. For instance, a driver age feature with three categorical values (i.e., junior, youth, adult) crosses a driver marriage feature with three categorical values (i.e., married, single, divorced), thereafter, generates a new dimension with eight meaningful categorical values (e.g., married and adult). Such dimension is more meaningful and explainable because it describes a subpopulation (e.g, married and adult drivers) of data with unique driving safety patterns in a feature space. Second, the latent interaction between two features is critical to decide whether we should cross them. However, the feature interaction is difficult to measure. Reinforcement is a great tool of AI for decision science when the mechanism is unclear, as it self-learns neural policies from feedback. Besides, a current feature generation step depends on previously generated feature space, which can be viewed as a Markov Decision Process. Third, as we need two agents to select two features for crossing, a simplified idea is to assume two agents are independent. Our large-scale analysis shows that the state of feature space changes in a chain, when a feature is selected to cross (we call meta feature), and continues to change when another feature is selected to be crossed (we call crossed feature). The two agents form a hierarchical structure.

Leveraging Categorical Crossing, Self-optimizing Reinforcement, and Hierarchical Agent Structure. We propose to develop a hierarchical reinforcement crossing framework that includes a meta controller agent and a controller, each of which in each iteration takes a state s as input, conducts an action a, and generates a new feature to obtain an updated generated feature set $\mathcal{T}$. The key components of our proposed method are as follows:

1) State. The state s_t is defined as the representation of the generated feature set $\mathcal{T}_t$. The agent makes decisions based on the current generated feature set $\mathcal{T}_t$ to produce a better $\mathcal{T}_{t+1}$. The algorithm of state representation is in Section III-B.

2) Action. **Figure 3** shows the agent consists of two components: a meta controller and a controller. The action of the meta controller is defined as $a_{1,t} = f_i$: select the i-th feature as the meta feature from the $\mathcal{T}_t$. The action of the controller is defined as $a_{2,t} = f_j$: combined with the meta feature, the controller visits all features in the $\mathcal{T}_t$ to select the j-th feature that can be crossed with meta feature to improve the representation of the feature set.

3) Supervised and Unsupervised Signals as Rewards. We found that both supervised signals and unsupervised signals can be reward measurements.

From the unsupervised signal perspective, we propose to measure feature-feature redundancy as a utility of feature space. Specifically, when two features have very high mutual information, that means the two features are similar and overlapped with a high collinearity. Therefore, adding the two similar features into the same feature space will increase information redundancy, instead of new information. In other words, one of the objectives of FG is to minimize feature redundancy. Formally, given a feature set $\mathcal{F}$, we can derive its overall mutual information $Rd(\mathcal{F})$, which we call redundancy:

$$Rd(\mathcal{F}) = \frac{1}{|\mathcal{F}|^2} \sum_{f_i, f_j \in \mathcal{F}} I(f_i, f_j) \tag{1}$$

where f_i is the i-th feature and f_j is the j-th feature, $I(f_i, f_j)$ is the mutual information to measure the correlation between two random variables f_i and f_j.

From the supervised signal perspective, we propose two reward measurements: 1) feature relevancy; 2) downstream task accuracy. First, feature relevancy describes how predictable are labels given a feature. The higher the feature relevancy is, the higher utility the feature has. So, one of the objectives is to maximize feature relevancy. Formally, given a feature set $\mathcal{F}$ and the label y, we can derive its overall mutual information $Rv(\mathcal{F}, y)$, which we call relevance:

$$Rv(\mathcal{F}, y) = \frac{1}{|\mathcal{F}|} \sum_{f_i \in \mathcal{F}} I(f_i, y) \tag{2}$$

where f_i is the i-th feature and y is the label. Second, downstream task accuracy is clearly a signal that describes the utility of a feature space. Therefore, another objective is to maximize downstream task accuracy.

4) Reward Functions for the Meta Controller Agent and Controller Agent. Since the framework has two agents: the meta controller and the controller, we design two personalized reward functions for the meta controller and controller.

- *The meta controller reward function* is quantified by a combination of the accuracy Acc on the machine learning task $\mathcal{M}$ at the current iteration, Acc_{best} on the $\mathcal{M}$ at one window time, the relevance between the generated feature set and the target label, and the redundancy of the generated feature set, given by:

$$r_1 = w_1 * (Acc - Acc_{best}) + w_2 * Rv - w_3 * Rd \tag{3}$$

  where w_i ($i \in 1, 2, 3$) is a positive weight.
- *The controller reward function* is measured by the accuracy of the machine learning task $\mathcal{M}$, relevancy, and the redundancy:

$$r_2 = w_4 * (Acc - Acc_{best}) + w_5 * Rv - w_6 * Rd \tag{4}$$

  where w_i ($i \in 4, 5, 6$) is a positive weight.

Temporal Abstraction. Figure 3 shows one iteration of HRC. The meta controller receives the state which is the representation of the current generated feature set, and selects a meta feature from the feature set. Then, the controller selects another feature based on the state and meta feature to generate the cross-feature. Finally, add the cross-feature to the feature set and start the next iteration. The objective of the meta controller is to maximize the cumulative rewards r_1:

$$O_1 = \sum\nolimits_{t'=0}^{\infty} \gamma_1^{t'} r_{1,t'} \tag{5}$$

where t' is the time step, and γ_1 is a discount factor. Similarly, the objective of the controller is to maximize the cumulative rewards r_2:

$$O_2 = \sum\nolimits_{t''=0}^{\infty} \gamma_2^{t''} r_{2,t''} \tag{6}$$

where t'' is the time step and γ_2 is a discount factor.

Solving the Model Training Problem. We adapt the DQN to derive policies for both the meta controller and the controller. The Q function for the meta controller is given by:

$$\begin{aligned} Q_1(s_t, a_{1,t}) &= \max_{a_{1,t}} \sum\nolimits_{t'=t}^{\infty} \gamma_1^{t'-t} r_{1,t'} \\ &= \max_{a_{1,t}} [r_{1,t} + \sum\nolimits_{t'=t+1}^{\infty} \gamma_1^{t'-(t+1)} r_{1,t'}] \\ &= \max_{a_{1,t}} [r_{1,t} + \gamma_1 \max_{a_{1,t+1}} Q_1(s_t, a_{1,t+1})] \end{aligned} \tag{7}$$

The Q function for the controller is given by:

$$Q_2(s_t, a_{2,t}) = \max_{a_{2,t}} [r_{2,t} + \gamma_2 \max_{a_{2,t+1}} Q_2(s_t, a_{2,t+1})] \tag{8}$$

We derive the loss function of Q_1 based the method in [7]:

$$\begin{aligned} L_1(\theta_{1,t}) = & E_{(s,a,r,s') \sim \mathcal{D}_1} \\ & [(r + \gamma_1 \max_{a'} Q_1(s', a'; \theta_{1,t}^-) - Q_1(s, a; \theta_{1,t})^2] \end{aligned} \tag{9}$$

where $\theta_{1,t}^-$ are fixed parameters of $\theta_{1,i}$ from the previous iteration. Similarly, we derive the loss function of Q_2:

$$\begin{aligned} L_2(\theta_{2,t}) = & E_{(s,a,r,s') \sim \mathcal{D}_2} \\ & [(r + \gamma_1 \max_{a'} Q_1(s', a'; \theta_{2,t}^-) - Q_1(s, a; \theta_{2,t})^2] \end{aligned} \tag{10}$$

We design two independent neural networks for Q_1 and Q_2 and train them alternately. When making decisions, we adapt a ϵ-greedy algorithm to enhance the exploration ability of reinforcement learning algorithms. The decision history of each network is stored in a memory. The new samples come into the memory and flush the old samples to update the memory. When samples are needed in training, we sample a batch of historical data from the memory and use the gradient descendent technique to update weights in the neural network. We also strategically design a memory mechanism to memorize the historically best version of the generated feature set. In this way, we can still guarantee a relatively good generated feature set, even if HRC does not converge.

IV. Experiment

A. Experiment Setup

1) Downstream Tasks: We validate our proposed feature generation method on classification tasks compared with state-of-the-art baselines. Specifically, we realize classification by three algorithms: logistic regression, decision tree, and random forest. All algorithms are implemented by the scikit-learn package [8].

2) Data Description: For classification tasks, we use the following four publicly available datasets:

- **Access**[*]: This dataset records the access to resources of Amazon employees from 2010 to 2011. Label 1 denotes allowed access and Label 0 denotes denied access.
- **Bank**[†]: This dataset is related to the direct marketing campaigns of a Portuguese banking institution. The classification goal is to predict if a client will subscribe to a term deposit.
- **Credit**[‡]: This dataset includes financial activities of bank customers. The classification goal is to predict if the customer will experience financial distress in the next two years.
- **Nomao**[§]: This dataset is a place localization dataset from UCI. The class label is categorical values that range from 1 to 2, which represents two geographical spots.

All the details about the datasets are shown in **Table I**.

[*]https://www.kaggle.com/c/amazon-employee-access-challenge/data
[†]https://www.kaggle.com/brijbhushannanda1979/bank-data
[‡]https://www.kaggle.com/c/GiveMeSomeCredit/data
[§]https://www.openml.org/d/1486

TABLE I: Details of datasets used in the experiments

Datasets				
Name	# Samples		Features	
	Training	Testing	# Num	# Cate
Access	26,216	6,553	0	9
Bank	21967	5,492	10	10
Credit	120,000	30,000	10	0
Nomao	27,572	6,893	89	30

3) Evaluation Metrics: To show the effectiveness of the proposed method, we use the accuracy metric for evaluating the classification of machine learning tasks. Additionally, we use the following metrics:

- **Accuracy** is the ratio of the sum of True Positives and True Negatives to the sum of True Positives, True Negatives, False Positives, and False Negatives. Formally, the accuracy is given by $\frac{TP+TN}{TP+TN+FP+FN}$, where TP, TN, FP and FN represent the number of True Positives, True Negatives, False Positives and False Negatives respectively. Larger values indicate better performance.
- **Precision** is given by $\frac{TP}{TP+FP}$ and presents the ratio of the number of True Positives to the number of True Positives and False Positives. Larger values indicate better results.
- **Recall** is given by $\frac{TP}{TP+FN}$ and presents the ratio of the number of True Positives to the number of True Positives and False Negatives. Larger values indicate better results.
- **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. Larger values indicate better results.

4) Baseline Methods: We compare the performances of our proposed methods: hierarchical reinforcement crossing feature generation (HRC) against the following four baseline algorithms.

- **Raw:** Directly conduct machine learning algorithms on the original dataset without any feature generation.
- **DeepFM:** DeepFM [9] combines factorization machines and deep learning in a new neural network architecture. It derives an end-to-end feature generation model that emphasizes both low and high-order feature interactions.
- **xDeepFM:** xDeepFM [10] generates feature interactions in an explicit fashion and at the vector-wise level. It can also learn arbitrary low and high-order feature interactions implicitly.
- **AutoCross:** AutoCross [4] is the state-of-the-art method that outperforms other existing feature generation methods based on cross operation. It can automatically construct high-order cross-features and uses a beam search strategy to iteratively generate a locally optimal feature set.
- **GRFG** [5] studies the problem of generation on continuous features and generates features based on operations (e.g., +, -, *, sin, cos).

Besides, we also implement ablation studies to verify the effectiveness of our method.

- **HRC*:** which means that we replace the controller with a greedy strategy based on our hierarchical strategy.
- **HRC$^\#$:** which means that we replace the meta controller with a greedy strategy based on our hierarchical strategy.
- **HRC$^!$:** which represents that we don't take any reinforcement learning method.

5) Hyperparameters and Reproducibility: In the experiments, for all HRC, we set the batch size to 20 and used AdamOptimizer with a learning rate of 0.01. We set the Q network of meta controller and controller in our methods as a one-layer ReLU with 100 nodes, respectively. The memory is set to 40 for each DQN. We limited the episode to 15, with each episode consisting of 70 exploration steps. All datasets are split with 80% for training and 20% for testing. The weights in reward functions have equal values. We used logistic regression as the default downstream task considering the fact that logistic regression is the most popular algorithm that is in need of feature generation in real-world applications [4].

B. Experiment Results

1) Overall Performance: This experiment aims to answer: *Whether our method can generate the best cross-feature space and improve downstream tasks*. Here, we would first like to emphasize that the classification task is the focus of all of our experiments. We take the approach of discretizing continuous values for feature crossing. However, in the regression task, this method will lose data accuracy when discretizing continuous variables. And we discovered that the regression task just got minor improvement during our experimental investigation. **Table II** shows the experimental results on different classification tasks in terms of accuracy, respectively. We observe that HRC always yields the best results compared to the available solutions for different datasets and different downstream tasks. The potential reason is that the HRC approach uses a two-stage reinforcement learning task in feature-cross that takes into account the overall optimization reward of long-term steps, allowing the model to search for the globally optimal result quickly. In addition, we do not need to enumerate all combinations of feature-cross, so it is very fast and efficient to get the best performance. Thus, compared with state-of-the-art baselines, our method is more practical and accurate in real-world application scenarios.

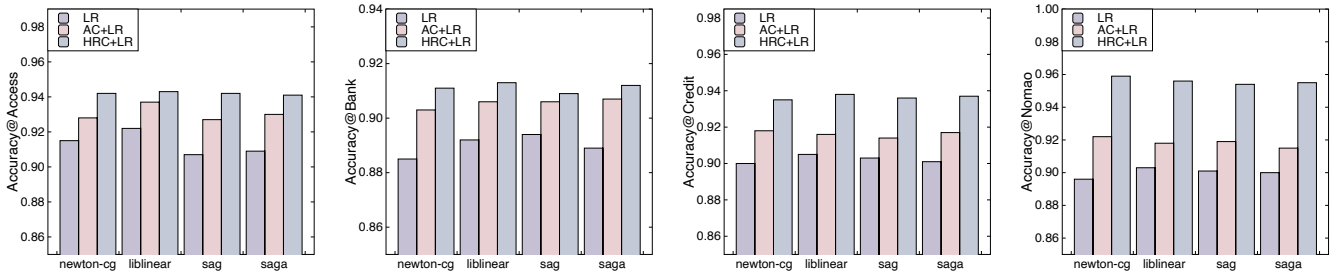
2) Ablation Study: This experiment aims to answer: *The impact of reinforcement learning component on the final result.* In our method, we design two DQNs for the meta controller and the controller, respectively. So in this set of experiments, we replace any of the reinforcement learning components with a greedy algorithm to verify the effectiveness of the HRC. **Table III** shows the results of ablation studies on different tasks evaluated by accuracy. We can discover that the results of HRC* are the worst in most cases. The HRC$^\#$ and HRC$^!$ perform slightly better than the HRC* but still perform worse than the HRC. Therefore, these experiments indicate that taking account of long-term rewards is indeed an important approach to improving feature generation.

TABLE II: Performance comparison of Hierarchical Reinforcement Crossing on different tasks evaluated by accuracy.

	Logistic Regression				Decision Tree				Random Forest			
	Access	Bank	Credit	Nomao	Access	Bank	Credit	Nomao	Access	Bank	Credit	Nomao
Raw	0.914	0.888	0.900	0.896	0.922	0.896	0.909	0.910	0.938	0.901	0.915	0.924
DeepFM	0.921	0.897	0.906	0.910	0.934	0.903	0.918	0.914	0.941	0.903	0.923	0.929
xDeepFM	0.927	0.901	0.911	0.909	0.939	0.906	0.923	0.920	0.947	0.904	0.925	0.931
AutoCross	0.930	0.903	0.919	0.921	0.941	0.908	0.931	0.928	0.951	0.910	0.929	0.934
GRFG	0.935	0.907	0.918	0.926	0.941	0.911	0.929	0.933	0.946	0.908	0.927	0.937
HRC	**0.943**	**0.913**	**0.938**	**0.959**	**0.948**	**0.915**	**0.937**	**0.953**	**0.955**	**0.919**	**0.935**	**0.969**

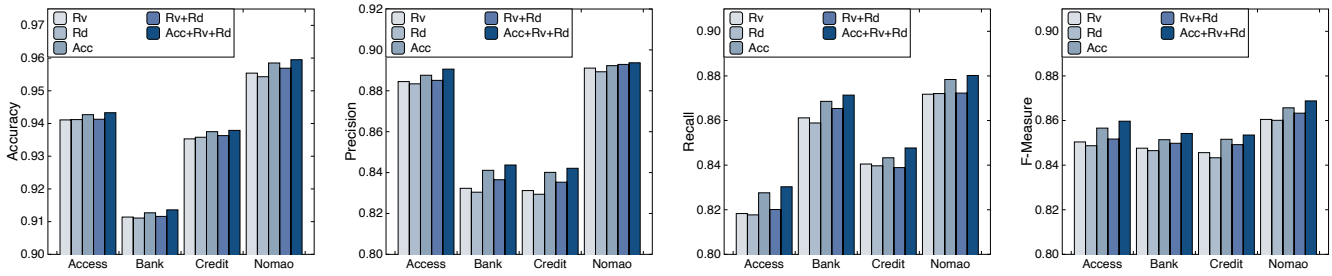
TABLE III: Performance comparison of ablation studies on different tasks evaluated by accuracy.

	Logistic Regression				Decision Tree				Random Forest			
	Access	Bank	Credit	Nomao	Access	Bank	Credit	Nomao	Access	Bank	Credit	Nomao
HRC*	0.939	0.903	0.934	0.945	0.940	0.903	0.931	0.938	0.945	0.907	0.925	0.958
HRC$^{\#}$	0.941	0.905	0.935	0.949	0.941	0.909	0.933	0.942	0.947	0.906	0.927	0.961
HRC$^{!}$	0.941	0.907	0.934	0.951	0.943	0.906	0.931	0.937	0.951	0.911	0.929	0.961
HRC	**0.943**	**0.913**	**0.938**	**0.959**	**0.948**	**0.915**	**0.937**	**0.953**	**0.955**	**0.919**	**0.935**	**0.969**



(a) Access (b) Bank (c) Credit (d) Nomao

Fig. 5: Performance comparison of different solvers on different datasets.



(a) Accuracy (b) Precision (c) Recall (d) F-Measure

Fig. 6: Performance comparison of different rewards in HRC on different datasets.

3) Robustness Check: This experiment aims to answer: *Whether HRC can still obtain stable results for different downstream tasks*. The predictive performance relies on not just feature generation, but also downstream tasks. As a result, we apply our method to logistic regression with different kernels in the generated feature set to see if our generated feature set is consistently stable and can consistently outperform other baseline methods on various predictor settings. In this way, we can examine the robustness of our methods. We use (1) 'newton-cg' solver; (2) 'liblinear' solver; (3) 'sag' solver; (4) 'saga' solver as the solvers of logistic regression for this experiment. We compare our method with LR (short for logistic regression) and the state-of-the-art AC (short for AutoCross) +LR. **Figure 5** shows the comparison of different solvers on different datasets. We can see that in every solver, The HRC+LR method outperforms AC+LR.

4) Study of Reward in HRC: This experiment aims to answer: *The impact of reward function design on the final result*. We study the impacts of the reward function in HRC and consider five cases:

(1) **Rv** that only considers relevance in the reward function;

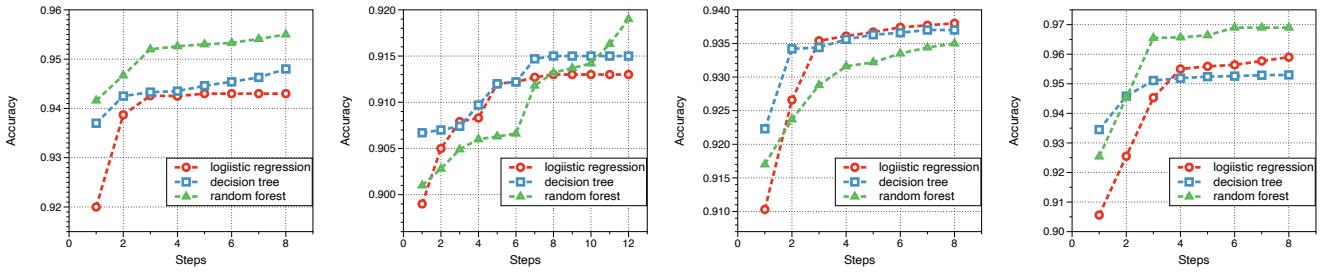
(2) **Rd** that only considers redundancy in the reward function;

(3) **Acc** that only considers accuracy in the reward function;

(4) **Rv+Rd** that only considers relevance and redundancy in the reward function;

(5) **Acc+Rv+Rd** that considers accuracy, relevance and redundancy in the reward function.

Figure 6 reveals the performance comparison of different rewards in HRC on different datasets. We can see that Acc is the second-best reward function, since it leads exploration in the direction of improving accuracy. Rv and Rd are less



(a) Access (b) Bank (c) Credit (d) Nomao

Fig. 7: Convergence steps of different downstream tasks on different datasets.

satisfactory. This is because both are unsupervised indicators of rewards and are not directly relevant to prediction accuracy. Their combination of Rv and Rd improves performance slightly but does not outperform Acc. Acc+Rv+Rd achieves the best performance since it takes both supervised and unsupervised indicators into account.

5) Convergence of HRC: This experiment aims to answer: *How many steps would the HRC take before convergent*? It is important for reinforcement learning methods to converge. We study the performances of HRC with different episodes, varying from 0 to 15 episodes over different datasets. Normally, the convergence of reinforcement learning methods is evaluated by the steadiness of accumulated rewards, but this is not the case in the feature generation problem. In feature generation, we only need to remember one optimal feature set through all of the episodes, and the HRC feature generation method is considered to converge when the remembered feature set no longer changes or falls into a predefined range. **Figure 7** shows that our HRC feature generation method converges in several episodes, which is extremely fast for a reinforcement learning method.

V. Related Work

Automated Feature Generation. There are three major categories of automated feature generation methods, i.e., factorization machine-based methods, cross-operation-based methods, and embedded methods [11]–[14]. Factorization machine methods can effectively capture the low-order interactions between features. *Blondel et al.* proposed the first generic yet efficient algorithms for training arbitrary-order HOFMs [2]. *Cheng et al.* proposed a novel Gradient Boosting Factorization Machine (GBFM) model to incorporate a feature selection algorithm with Factorization Machines into a unified framework [1]. *Juan et al.* established field-aware factorization machines as an effective method for classifying large sparse data including those from CTR prediction [3]. *Cheng et al.* presented Wide and Deep learning-jointly trained wide linear models and deep neural networks to combine the benefits of memorization and generalization for recommender systems [15]. These methods were implicit and had difficulty to capture high-order interactions. Cross operations such as [4], [16] captured high-order feature interactions. *Liu et al.* gave a definition of interpretation inconsistency in deep neural networks, and proposed a novel method called CrossGO, which selected useful cross features according to the interpretation inconsistency [17]. *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 [4]. There were also other embedded methods like gradient boost machine [18] and group lasso [19] which build useful features in the process of model training. *Dong et al.* combined feature selection and feature generation into a transformation graph and optimized the two processes jointly [20]. However, these methods usually require complex optimization procedures and thus have difficulties dealing with large-scale datasets [21], [22].

Hierarchical Reinforcement Learning. Hierarchical reinforcement Learning (HRL) [23] decomposes the target Markov decision process (MDP) [24]into a hierarchy of smaller MDPs and solves them sequentially [25]–[27]. There are numerous studies on HRL. *Dietterich et al.* defined a hierarchical Q learning algorithm, proved its convergence, and showed experimentally that it can learn much faster than ordinary "flat" Q learning [28]. *Vezhnevets et al.* employed a Manager module and a Worker module. The Manager operated at a lower temporal resolution and set abstract goals which are conveyed to and enacted by the Worker. The Worker generated primitive actions at every tick of the environment [29]. *Morimoto et al.* proposed a hierarchical reinforcement learning architecture that realized practical learning speed in real hardware control tasks [30]. *Ribas et al.* proposed that the computations supporting hierarchical behavior may relate to those in HRL, a machine-learning framework that extended reinforcement-learning mechanisms into hierarchical domains [31]. *Florensa et al.* proposed a general framework that first learns useful skills in a pre-training environment, and then leverages the acquired skills for learning faster in downstream tasks [32]. *Lin et al.* proposed to tackle the large-scale fleet management problem using reinforcement learning, and proposed a contextual multi-agent reinforcement learning framework that successfully tackled the taxi fleet management problem [33]. *Wei et al.* proposed a more effective deep reinforcement learning model for traffic light control [34]. *Nachum et al.* proposed to use off-policy experience for both higher and lower-level training [35].

VI. Conclusion

In this paper, we study the problem of automated feature generation. We propose the hierarchical reinforcement crossing (HRC) method to generate the cross-feature set, which can obtain the best performance on all datasets. In the HRC method, we fuse predictive accuracy and information gain to design an inspiring reward function; we use the categorical hashing representation to reduce the dimension of an increased feature set; and we design a descriptive state representation method to derive a fixed-length state vector as the input for reinforcement learning policies. The challenges of the feature generation problem are that the number of generated features can be explosively large and the generated feature set extremely falls into local optima. Our method doesn't need to enumerate all potential combinations of features, and it can explore a globally optimal cross-feature set quickly considering the long-term rewards. The performance superiority in experiments demonstrates the effectiveness of our approach.

References

[1] C. Cheng, F. Xia, T. Zhang, I. King, and M. R. Lyu, "Gradient boosting factorization machines," in *Proceedings of the 8th ACM Conference on Recommender systems*, 2014, pp. 265–272.

[2] M. Blondel, A. Fujino, N. Ueda, and M. Ishihata, "Higher-order factorization machines," in *Advances in Neural Information Processing Systems*, 2016, pp. 3351–3359.

[3] Y. Juan, Y. Zhuang, W.-S. Chin, and C.-J. Lin, "Field-aware factorization machines for ctr prediction," in *Proceedings of the 10th ACM Conference on Recommender Systems*, 2016, pp. 43–50.

[4] Y. Luo, M. Wang, H. Zhou, Q. Yao, W.-W. Tu, Y. Chen, W. Dai, and Q. 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*, 2019, pp. 1936–1945.

[5] D. Wang, Y. Fu, K. Liu, X. Li, and Y. Solihin, "Group-wise reinforcement feature generation for optimal and explainable representation space reconstruction," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 1826–1834.

[6] J.-P. Aumasson and D. J. Bernstein, "Siphash: a fast short-input prf," Cryptology ePrint Archive, Paper 2012/351, 2012, https://eprint.iacr.org/2012/351.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[9] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: a factorization-machine based neural network for ctr prediction," *arXiv preprint arXiv:1703.04247*, 2017.

[10] J. Lian, X. Zhou, F. Zhang, Z. Chen, X. Xie, and G. 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*, 2018, pp. 1754–1763.

[11] M. Xiao, D. Wang, M. Wu, K. Liu, H. Xiong, Y. Zhou, and Y. Fu, "Traceable group-wise self-optimizing feature transformation learning: A dual optimization perspective," *arXiv preprint arXiv:2306.16893*, 2023.

[12] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.

[13] Y. Saeys, I. Inza, and P. Larrañaga, "A review of feature selection techniques in bioinformatics," *bioinformatics*, vol. 23, no. 19, pp. 2507–2517, 2007.

[14] M. Xiao, D. Wang, M. Wu, Z. Qiao, P. Wang, K. Liu, Y. Zhou, and Y. Fu, "Traceable automatic feature transformation via cascading actor-critic agents," in *Proceedings of the 2023 SIAM International Conference on Data Mining (SDM)*. SIAM, 2023, pp. 775–783.

[15] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir *et al.*, "Wide & deep learning for recommender systems," in *Proceedings of the 1st workshop on deep learning for recommender systems*, 2016, pp. 7–10.

[16] Y. Shan, T. R. Hoens, J. Jiao, H. Wang, D. Yu, and J. 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*, 2016, pp. 255–262.

[17] Z. Liu, Q. Liu, and H. Zhang, "Automatically learning feature crossing from model interpretation for tabular data," 2019.

[18] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[19] L. Meier, S. Van De Geer, and P. Bühlmann, "The group lasso for logistic regression," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 70, no. 1, pp. 53–71, 2008.

[20] G. Dong and H. Liu, *Feature engineering for machine learning and data analytics*. CRC Press, 2018.

[21] G. Katz, E. C. R. Shin, and D. Song, "Explorekit: Automatic feature generation and selection," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, 2016, pp. 979–984.

[22] H. Leather, E. Bonilla, and M. O'boyle, "Automatic feature generation for machine learning–based optimising compilation," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 1, feb 2014. [Online]. Available: https://doi.org/10.1145/2536688

[23] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek, "Hierarchical reinforcement learning: A comprehensive survey," *ACM Comput. Surv.*, vol. 54, no. 5, jun 2021. [Online]. Available: https://doi.org/10.1145/3453160

[24] M. L. Puterman, "Chapter 8 markov decision processes," in *Stochastic Models*, ser. Handbooks in Operations Research and Management Science. Elsevier, 1990, vol. 2, pp. 331–434. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0927050705801720

[25] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of artificial intelligence research*, vol. 13, pp. 227–303, 2000.

[26] M. M. Botvinick, "Hierarchical reinforcement learning and decision making," *Current opinion in neurobiology*, vol. 22, no. 6, pp. 956–962, 2012.

[27] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete event dynamic systems*, vol. 13, no. 1-2, pp. 41–77, 2003.

[28] T. G. Dietterich, "The maxq method for hierarchical reinforcement learning." in *ICML*, vol. 98. Citeseer, 1998, pp. 118–126.

[29] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 3540–3549.

[30] J. Morimoto and K. Doya, "Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning," *Robotics and Autonomous Systems*, vol. 36, no. 1, pp. 37–51, 2001.

[31] J. J. Ribas-Fernandes, A. Solway, C. Diuk, J. T. McGuire, A. G. Barto, Y. Niv, and M. M. Botvinick, "A neural signature of hierarchical reinforcement learning," *Neuron*, vol. 71, no. 2, pp. 370–379, 2011.

[32] C. Florensa, Y. Duan, and P. Abbeel, "Stochastic neural networks for hierarchical reinforcement learning," *arXiv preprint arXiv:1704.03012*, 2017.

[33] K. Lin, R. Zhao, Z. Xu, and J. Zhou, "Efficient large-scale fleet management via multi-agent deep reinforcement learning," *arXiv preprint arXiv:1802.06444*, 2018.

[34] H. Wei, G. Zheng, H. Yao, and Z. 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*, 2018, pp. 2496–2505.

[35] O. Nachum, S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," *arXiv preprint arXiv:1805.08296*, 2018.