AUTOMATED FEATURE ENGINEERING WITH REINFORCEMENT LEARNING

A Thesis

by

ZIYU XIANG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2019

Major Subject: Electrical & Computer Engineering

ABSTRACT

Automated Feature Engineering (AFE) is one of emerging machine learning topics to automatically extract useful information from given data whose dependency relationships can be interpreted with analytical functions. However, it is often computationally prohibitive to exhaust all the potential relationships to construct and search the whole feature space. We explore new AFE strategies by feature generation tree exploration with Deep Q-learning to find the optimal exploration policy. This thesis will develop, evaluate and benchmark such reinforcement learning based AFE methods and compare with the existing AFE methods.

ACKNOWLEDGMENTS

I would like to thank Dr. Xiaoning Qian's and Dr. Dileep Kalathil's valuable guidance for this project. I would also like to thank Dr. Tie Liu and Dr. Xia Hu for carefully reviewing this material. Special thanks to Dr. Xiaofeng Qian for providing datasets for the experiments. We acknowledge the support of TAMU's HPRC platform for implementing the experiments and obtaining the results.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor Xiaoning Qian, Professor Dileep Kalathil and Professor Tie Liu of the Department of Electrical & Computer Engineering and Professor Xia Hu of the Department of Computer Science and Engineering.

The data analyzed for Chapter 3 was provided by Professor Xiaofeng Qian of the Department of Materials Science and Engineering and another published artical[1].

All other work conducted for the thesis was completed by the student independently.

NOMENCLATURE

FE	Feature engineering
MFE	Manual Feature Engineering
AFE	Automated Feature Engineering
SISSO	Sure Independence Screening and Sparse Operation
LASSO	Least Absolute Shrinkage and Selection Operator
MDP	Markov Decision Process
DQN	Deep Q-learning
UCB	Upper Confidence Bounds
UCT	Upper Confidence Tree
NN	Neural Network

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION AND LITERATURE REVIEW

1.1 Introduction

Feature engineering (FE) is the process of creating features that captures hidden dependency relationships in data and helps machine learning algorithms work better. It involves lots of intuition, experience, and often takes a long trial-and-error process. People strive to use feature engineering to create new features and remove redundant features because we want to provide more information about the underlying structure of data to the machine learning systems to aid them learn efficiently and understand underlying systems with good interpretability instead of simply relying on black-box predictions

However, traditional Manual Feature Engineering (MFE) faces problems of either being problem-specific—a code for one problem may not be able to be applied to another problem, or being error-prone—the final results will depend on humans' patience and creativity. Recently, a new technology has arisen as Automated Feature Engineering (AFE) to find the dependency structure and relationships automatically from given datasets. Typically, an AFE method will construct new feature spaces by operating algebraic operations on primary features in the original dataset, and extract features from it to build a better predictive model. Compared with MFE, AFE showed the advantages of having interpretable features with real-world significance, preventing invalid data to mislead the predictive model, having generally applied code for various datasets and being able to save more human and computing resources.

In this thesis, we mainly focus on exploring a new AFE strategy by using reinforcement learning in AFE. The proposal will develop as the following. The first chapter will introduce the background of AFE and the problem we are studying. In the second chapter, We first define a feature generation tree and transform the exhaustive feature construction process into a feature generation tree exploration problem. Then we adopt Deep Q-Learning (DQL) [2] with experience replay and target network to learn the optimal policy to generate such an ideal feature set. In the third chapter, We will use the datasets and results from Ouyang's artical [1] for evaluation, where we aim to find an analytical function of primary features as a descriptor to predict materials' metal/non-metal property. Then we will discuss the results by comparing our algorithm with other existing AFE methods to analyse its effects in improving the learning efficiency and computation scalability. Finally in the forth chapter, we will summary the existing and expected results, as well as introduce the challenges and work plan for this thesis.

1.2 Related Work

Several AFE methods have been proposed recently. For example, Deep Feature Synthesis [3] presents an algorithm to automatically generate features related to datasets. The Sure Independent Screening and Sparse Operation (SISSO) [1] was introduced to construct the overall feature space and then use the sure independent screening to generate a rather small feature space using sparse operators (e.g. Least Absolute Shrinkage and Selection Operator (Lasso)) to implement on. Khurana [4] introduced a transformation Graph to combine reinforcement learning with AFE using linear functional approximation. Gaudel [5] introduced a feature selection method based on reinforcement learning, relying on one-player game approach and Upper Confidence Tree (UCT).

1.3 The Statement of the Problem

Consider a primary dataset $D_0 = < F_0, y >$, where F_0 denotes the finite set of primary features $\{f_0, f_1, ...\}$ and y denotes the target vector. When y is continuous, then it describes a regression problem. Otherwise when y is categorical, then it describes a classification problem.

For AFE, we need to construct sets of generated features F_i and assess them based on their classification or regression accuracy $A_L\{F_i, y\}$. L defines the classification or regression algorithm and F_i denotes the generated set of features $\{func_0(F_0, c_1), func_1(F_0, c_2), ...\}$, where functions $func_i()$ consist of set of algebraic operations ϕ in the set H:

$$H = \{exp, log, \wedge 2, \wedge 3, \wedge 6, \wedge -1, \sqrt{\ }, \sqrt[3]{\ }, +, -, \times, \div\} \tag{1.1}$$

and coefficients c_i represents the complexity—the maximum number of algebraic operations implemented on one feature in one function. For example, the function $exp(f_0) \times f_1^2 + f_2$ has the complexity of 3.

Thus, the whole feature space $F = \{F_0, F_1, ...\}$ consists of the primary feature set F_0 and the generated feature set F_i's. Then our goal is to find such a generated feature set F^* to maximize the accuracy $A_L\{F_i, y\}$:

$$F^* = \arg \max_{F_i \in F, c_i < c_{max}} A_L\{F_i, y\}. \tag{1.2}$$

It can also gives the set of analytical function $func_i()$ generating F_i within the maximum complexity c_{max}, which might reveal the potential relationships between primary features F_0 and the target vector y.

2. METHODOLOGY

2.1 Feature Generation Tree

Different from SISSO [1] which constructs the whole feature space first and then implement the sure independent screening to produce a rather small sub-feature space for sparse operation (e.g. Lasso), here we transform the feature construction process as a tree search problem.

As shown in the fig 2.1, iterations start from the root node n_0 which represents the primary feature set. We choose an operation ϕ_0 (represented as an edge) according to some policy π and then move forward to the next node n_i and obtain the generated feature set's accuracy $A_L\{F_i, c_i\}$. Similarly, from the current node n_i, we choose another operation ϕ_i according to π and move forward to $n_j (j > i)$ and obtain $A_L\{F_j, c_j\}$. We repeat this step until we attain the maximum complexity c_{max}, which also represents the maximum height of the tree, and start a new iteration. So the total feature space F consists of the primary feature set F_0 and all the generated feature set F_i. And each node's generating function $func_i()$ is composed of each operation to generate that node.

So far, we have defined the *Feature Generation Tree* to implement the features construction process, but our goal is to find the optimal policy π^*, which aims to find the optimum generated feature set F_i with highest efficiency and lowest computation scalability. We will then use reinforcement learning to find such a policy π^*

2.2 Learning Algorithm

Since our problem is a model-free, finite Markov Decision Process(MDP) problem, we consider using Q-learning as the learning algorithm. Besides, considering the huge feature space and the large available operation set we may have, compute all the available accuracy as Q values for each $< state, action >$ pair may be expensive. So we will use Deep Q-learning[2] to approximate the Q value through a neural network. Formally, we define the states, actions and rewards as the following:

- **states**: F_i, which denotes a primary or generated feature set

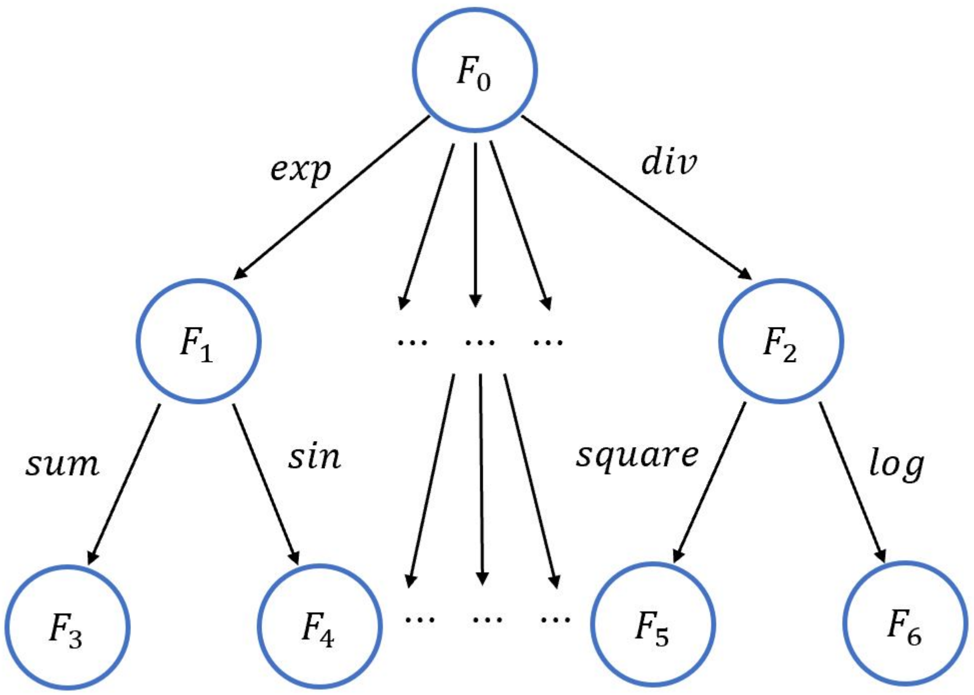

Figure 2.1: Example of a feature generation tree

- **actions**: ϕ, which denotes an operation in the operation set **H**
- **rewards**: $A_L\{F_i, c_i\}$, which denotes the accuracy of the current state with a classification or regression algorithm L

Thus, the peusocode of the Deep Q-learning Algorithm for feature generation tree is given as the algorithm 1. We will update the Neural Network (NN) weights as:

$$w = w + \alpha(A_L\{F_i, y\}) + \gamma \max_b Q(s_i, b, w^-) - Q(s_i, a_i, w))\nabla Q(s_i, a_i, w) \tag{2.1}$$

In this algorithm, dim denotes the dimension of optimal generated feature set F^*. We'll always look for a F^* which can meet our threshold of accuracy with the lowest dimension. So for each feature engineering problem, we will look for F^* with $dim = 1$ first.

To do it, we will start at the state for each episode, which is either one of primary features or random selected features from the generated feature sets which haven't attained c_{max}. From the contemporary state, we'll choose actions according to methods which can be either epsilon greedy or Upper Confidence Bounds (UCB) algorithm. Then, we will calculate next state's features and their accuracy according to the actions we have chose, store those information into the buffer for experience replay of the DQL, and finally move to the next state. Each episode will keep iterating these processes until the reward is meet our threshold and the program will be closed, or the state attained its maximum complexity and a new episode will start.

If the program finds the one-dimensional (1^d) F^* satisfying the threshold within the maximum number of episodes we can tolerate, then the solution for the problem is found. Otherwise, we will start a new process to look for F^* with higher dim like 2^d and 3^d.

There is one problem needed to be noticed when searching higher dimensional F^*. In order to implement Deep Q-learning, we need a consistent feature space. So every time with a higher dimension of F^*, we construct a new NN for Deep Q-learning with the input dimension as $dim * n_{samples}$, where $n_{samples}$ represents the number of samples in the dataset. Accordingly, all the features in F_i are able to be input to NN.

The process for exploring 2^d descriptors will go like this: from the 1^d descriptors results, we will choose several (e.g. the number of 5) 1^d descriptors of highest accuracy and store them in the 1^d selected feature set $F_s^1 = \{f_0^1, f_1^1, \ldots\}$. The 1^d selected feature set will then be stored in the selected feature set F_s, which has the following form: $F_s = \{F_s^1, F_s^2, \ldots\}$. Then similar to the 1^d exploring process, the tree will start from one initial feature f_0, and grow through the same way. But what's different is how we calculate the rewards.

In 1^d descriptor exploring process, we will take the reward as the accuracy of generated features directly. However, in 2^d situation, we will calculate the accuracy with two features. One is the generated features as the state, and the other is one from the 1^d selected feature set F_s^1. Then we will take the maximum rewards as the reward for the action. We denote $A_L(F)$ as the accuracy of a set of features F using the algorithm L. Then rewards can be denoted as:

$$reward = \max_{f_i^1 \in F_s^1} A_L(state, f_i^1) \tag{2.2}$$

Accordingly, the process is equal to we choosing one feature to add to the feature space F_s^1, which will maximize the accuracy of 2^d descriptors $\{d_0^2, d_1^2\}$ extracted from the feature space. Similarly, when we consider 3^d or higher dimensional situation, we will calculate the accuracy with three or more features, consisting of one features as the state and the others selected from the selected features set F_s. We will denote the function $func()$ generating feature set F as the descriptor D and the following algorithm 1 describes these process.

Algorithm 1 DQL Method

```
1: input: Primary features $F_0$, Action set $H$
2: for $dim = 1, 2, ...$ do
3:     Construct Neural Network $nn$ for DQL
4:     Clear $Buffer$
5:     for $episode = 1, 2, ..., N$ do
6:         $state$ = random_start()
7:         while $c < c_{max}$ do
8:             With probability $\epsilon$, random choose $action$ in $H$
9:             With probability $1 - \epsilon$, $Q(state, action) = nn.predict(state)$, choose $action = \arg\max_{action \in H} Q(state, action)$
10:            $next\_state, reward, c$ = step$\{state, action\}$
11:            $Buffer \leftarrow \{state, next\_state, reward, c\}$
12:            $Batch \leftarrow \{state, next\_state, reward, c\}_n$ in $Buffer$
13:            $Q\_values(state, action) = nn$.predict($state$ in $Batch$)
14:            $Q\_targets(state, action) = reward + \gamma(\max nn.predict(next\_state))|sign(c - c_{max})|$ for $\{reward, next\_state, c\}$ in $Batch$
15:            $Q\_values \leftarrow Q\_targets$
16:            nn.fit($state$ in $Batch$,$Q\_values$)
17:            if $reward > threshold$ then
18:                exit
19:            end if
20:            $state \leftarrow next\_state$
21:        end while
22:    end for
23:    Store Generated feature $F_s^d$ of highest $reward$ with descriptors $D_s^d$ to $F_s$
24: end for
25: Output: Generated feature $F_s$ of highest $reward$ with descriptors $D_s$
```

2.3 Evaluation Methods

In our project, we'll mainly compare our algorithm with another AFM method such as SISSO [1] in both classification problems and regression problems. To be more specifically, we'll divide the problem into four experiments:

- 1^d exploration for classification problems
- multi-dimensional exploration for classification problems
- 1^d exploration for regression problems
- multi-dimensional exploration for regression problems

For classification problems, we will use the NaCl prototype materials dataset from Ouyang's artical [1] for metal/non-metal property of NaCl prototype materials' classification. For regression problems, we will use the dataset provided by Drs. Raymundo Arróyave and Xiaofeng Qian of the Materials Science and Engineering Department.

To compare the different algorithms, we mainly evaluate the running time for observing the first descriptor to obtain the 100% accuracy as the efficiency, and the number of total generated features as the computation scalability.

3. PRELIMINARY RESULTS AND DISCUSSION

To validate the algorithm proposed in chapter 2, the experiment based on 1^d descriptor exploration for classification problems is designed. The experiment was implemented based on the NaCl prototype materials dataset from Ouyang [1], which is composed of 132 samples and seven primary features $\{IE_A, IE_B, d_{AB}, r_{covA}, r_{covB}, \chi_A, \chi_B\}$. The problem is focusing on predicting the metal/non-metal property of NaCl prototype materials and so is a classification problem. We used Logistic Regression to calculate the accuracy as rewards and use a operation set including $\{exp, log, \wedge 2, \sqrt{}, -, \div\}$.

As for the DQL coefficients setting, we set parameters as the following: Learning rate: 0.001, Batch size: 32, Gamma: 0.99, Epsilon: 1.0(decay 0.99 min 0.1) and the hidden layer for neural network: {150,120} with the activation function as $relu$. The maximum complexity c_{max} is set to be 3.

From the experimental result in the table 3.1 we can see that compared with the primary feature set, SISSO and DQL both can improve the classification accuracy(maximum of 82.5% v.s. 100%). However, SISSO has to take 8.6 hours to compute the overall feature space of 830,877 features to obtain the optimal descriptor of 100% accuracy. While in the first hour, DQL can find the descriptor of 97.5% accuracy with only constructing 123,623 features. Then, DQL took 6.8 hours to find the same optimal descriptor as SISSO's of 100% accuracy with 514,196 features constructed. We can see a decrease in the constructed feature space between SISSO's and DQL's, and also a time saving of DQL. This means by using DQL to explore the features generating tree can increase the

Table 3.1: Comparison of Results between Primary Features, SISSO and DQL

Algorithm	Descriptors	Running time	Accuracy	Features Generated
	Primary Features		82.5%	
DQL	$\sqrt[4]{IE_B} - IE_A + \frac{IE_A}{rcov_A}$	1 hour	97.5%	123,623
DQL	$\frac{IE_A IE_B (d_{AB} - r_{covA})}{exp(\chi_A)\sqrt{r_{covB}}}$	6.8 hour	100%	514,196
SISSO		8.6 hour	100%	830,877

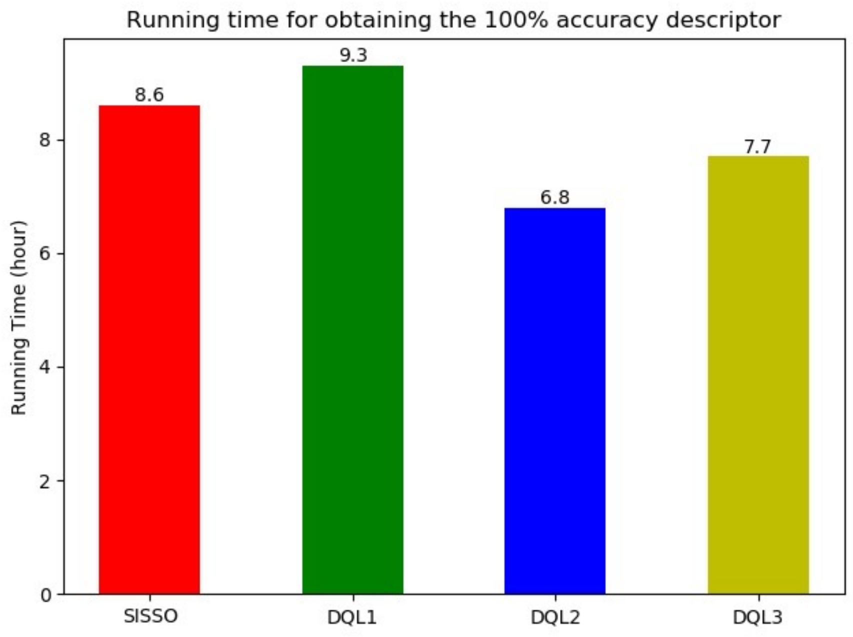

Figure 3.1: Comparison of running time between SISSO, DQL1 with root node of primary features, DQL2 and DQL3 with root node of random selected features

exploring efficiency and decrease the computation scalability.

Fig 3.1 shows a comparison results of running time between three kinds of algorithms: SISSO, DQL1 with root node of primary features, DQL2 and DQL3 with root node of random selected features. It is worth noticing that DQL1 even behaves worse than SISSO. This is because if we always start with the root node of primary features, DQL will turn to exploitation too soon and be easily trapped in the local optimal. If we increase the exploration rate in the epsilon greedy method, then it will converge too slowly. So one solution is to replace the root node with the random selected features which haven't attained the maximum complexity c_{max} after some periods of exploring. This can increase the chance of exploring more possible combinations of features and won't affect the exploitation process of choosing actions obviously. Then we can see that with such an adapted algorithm DQL2 and DQL3, we can have a better running efficiency than SISSO. Besides, with the huge feature space **F** ,the possible large operation set **H** and limited number of running episodes, the feature generation tree will grow very large but also very flat, which means the problem is a typically multi-armed Bandit problem [5]. To further strengthen the exploration process, we can also replace the DQL2's epsilon greedy method with Upper Confidence Bounds (UCB) algorithm. This result shows that the proposed algorithm can decrease the total running time to obatain 100% accuracy descriptor of SISSO(8.6 Hours) to DQL(6.8 Hours) for 1^d exploration.

4. SUMMERY AND FUTURE WORK

4.1 Summery

In this thesis, we will present a novel AFE algorithm based on the feature generation tree exploration with Deep Q-learning. Unlike SISSO [1], which constructs the whole feature space first, we transform the feature construction process into a feature generation tree exploration problem. Also, unlike transformation graph [4], which explores the best set of generated features with variant feature space and action space, we define our feature generation tree with consistent feature space and action space for deep Q-learning. The proposed algorithm will always aim to find the lowest dimension of the generated feature set with the lowest complexity. The existing results showed that our algorithm reduces the running time of SISSO by 20.9% and also increases the computation scalability of SISSO by 38.1% with the same dataset for 1^d exploration. However, this is only the partial results for the whole problem. Higher dimensional exploration for F^* in both classification and regression problems remains to be explored.

4.2 Challenges

The challenges of the project mainly come from two aspects—the large and sparse action space and the potential local optimality due to the nature of multi-armed Bandit problems.

As for the first challenge, in the already proposed algorithm we define the actions based on the type of algebraic operations for the consideration of computation complexity. However, this is not the ideal definition of the action space for AFE because it misses the information of which features those operations will be implemented on (e.g. the addends for addition). Accordingly, problems may occur for binary operations (e.g. addition). For such an operation, we have taken the strategy to add each of all the existing features to the current states' features and select one generated feature with the highest accuracy as the next state. This may introduce bias in exploration because the accuracy may not be monotonic: even when we choose the next state with the highest accuracy at this step, the expectation of future rewards of this combination may not be the maximum. It can also consume large computation resources to compute all possible combinations.

But if we define the action space to be the pair of types of operations and the target features to be implemented on, then the action space will become inconsistent during the exploration with more and more generated features added to the feature space. And the action space will also become large and sparse. These will cause troubles to DQL-based reinformance learning. To solve such a problem, Dulac-Arnold [6] proposed an algorithm to embed the action space into continuous space and use policy gradient methods in an actor-critic framework to train it. If such a way works, then it is possible to combine transformation graph [4] to our proposed algorithm in Chapter 2, which will be one of our research directions.

As for the second challenge, as mentioned in Chapter 3, with the huge and increasing feature space **F** and operation set **H**, as well as the limited number of running episodes, the feature generation tree can grow very large but also very flat, which means that the problem is a typically multi-armed Bandit problem. It is very important to keep best exploration-exploitation trade off to solve it efficiently. Otherwise it is very easily to be trapped in local optima. In Chapter 3, we replace the root of the feature generation tree with random selected features. But it's not enough. When using Upper Confidence Tree (UCT) algorithm, with a large number of actions compared with allowed iterations, it will be biased towards exploration. Modifications of the UCT algorithm [5] can be used in this situation as the other research direction that we will pursue.

4.3 Work Plan and Time Table

- Already done: 1^d exploration for AFE in classification.
- $1\backslash 15 \sim 2\backslash 7$: Finish the rest coding and experiments for multi-dimensional exploration for classification problems and extend the algorithms for regression problems.
- $2\backslash 8 \sim 2\backslash 18$: Combine the refined UCT algorithm [6] and compare it with UCB algorithm and epsilon greedy method.
- $2\backslash 19 \sim 3\backslash 2$: try to use the policy gradient method with actor-critic framework to deal with large and sparse action space [5] and combine transformation graph [4] to our developed

algorithms.

REFERENCES

[1] e. a. Ouyang, Runhai, “Sisso: A compressed-sensing method for identifying the best low-dimensional descriptor in an immensity of offered candidates,” *Physical Review Materials*, vol. 2, no. 8, p. 083802, 2018.

[2] e. a. Mnih, Volodymyr, “Human-level control through deep reinforcement learning.,” *Nature*, vol. 518, no. 7540, p. 529, 2018.

[3] J. M. Kanter and K. Veeramachaneni., “Deep feature synthesis: Towards automating data science endeavors,” *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2015.

[4] H. S. Khurana, Udayan and D. Turaga, “Feature engineering for predictive modeling using reinforcement learning,” *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[5] R. Gaudel and M. Sebag., “Feature selection as a one-player game,” 2010.

[6] e. a. Dulac-Arnold, Gabriel, “Deep reinforcement learning in large discrete action spaces,” *arXiv preprint arXiv*, no. 1512, p. 07679, 2015.