Automated Feature Engineering Using Deep Reinforcement Learning

Ziyu Xiang [1]

Abstract

Automated Feature Engineering(AFM) is an emerging technique to automatically extract useful information of structured data from a dataset and gives an interpretable analytical function. However, typically an AFM method will require to construct the whole features space, which might be very huge and sparse. In this paper, we transform such a feature construction process into a feature generation tree exploration problem and use Deep Q-learning to find the optimal exploration policy. This Algorithm is proved to have effects in optimizing the learning efficiency and reducing the computation scalability compared with the traditional AFM method.

1. Introduction

Feature engineering (FE) is considered as the process of using domain knowledge of data to create features that help machine learning algorithms work better. It also involves lots of intuition and often takes a long process of trial and error. 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 rather than let the algorithms learn these features all by themselves. Besides, sometimes we not only want a black-box system telling us the prediction but also a model with good interpretability.

However, such a traditional Manual Feature Engineering (MFE) faced problems of either being problem-specific, a code for one problem may not be able to reapply 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). It will use automated process to find the data structure and relationships in datasets. Compared with MFE, AFE showed the advantages of having interpretable features with real-world significance, preventing invalid data to mislead the predictive model, having general applied code for various datasets and being able to save more human and computing resources.

In our project, we mainly focus on using reinforcement learning in AFE. We will use the datasets and results for evaluation from (Ouyang, 2018), which is a classification problem and we aim to find an analytical function of primary features as a descriptor to predict materials' metal/non-metal property. We firstly define a feature generation tree and transform the exhausted feature construction process into feature generation tree exploration problem. Then we adopt Deep Q-Learning (Mnih, 2018) with experience replay and target network to learn the optimal policy to generate such an idea descriptor. Finally we will compare with our algorithm with SISSO to analyse its effects in improving the learning efficiency and computation scalability.

2. Related Work

Several methods of AFE research have been done. For example, Deep Feature Synthesis (Kanter & Veeramachaneni., 2015) presents an algorithm to automatically generate features related to datasets. SISSO (Ouyang, 2018) firstly constructed the overall features space and then used the sure independent screening to generate a rather small features space for sparse operations(e.g. Lasso) to implement on. (Khurana & Turaga, 2018) introduced the transformation Graph to combine reinforcement learning with AFE using linear functional approximation.

3. Problem Formulation

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.

To construct the whole features space, we need to construct set of generated features set F_i and assess it with its classification or regression accuracy $A_L\{F_i, y\}$. L defines the classification or regression algorithm and F_i denotes the gen-

[1]Department of Electronic and Computer Engineering, Texas A&M University, College Station, US. Correspondence to: Ziyu Xiang <ziyu.xiang@tamu.edu>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

erated set of features $\{func_0(F_0, c_1), func_1(F_0, c_2), ...\}$, where functions $func_i()$ consist of set of operations ϕ in the operation set H:

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

and coefficients c_i represents the maximum number of 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 features space $F = \{F_0, F_1, ...\}$ consists of the primary features set F_0 and the generated features set F_i. Then our goal is to find such a generated features set F^* to maximize the accuracy $A_L\{F_i, y\}$:

$$F^* = \arg \max_{F_i \in F, c_i < c_{max}} A_L\{F_i, y\} \tag{2}$$

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.

4. Research Approach

4.1. Feature Generation Tree

Different from SISSO(Ouyang, 2018) which constructs the whole features space first and then implement the sure independent screening to produce a rather small sub-features space for sparse operation(e.g. Lasso), here we transform the features construction process as a tree search problem.

Iterations start from the root node n_0 which represents the primary features 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 features 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 features space F consists of the primary features set F_0 and all the generated features setF_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 features set F_i with highest efficiency and lowest computation scalability. So we will then use reinforcement learning to find such a policy π^*

4.2. Learning Algorithm

As our problem is a model-free, finite Markov Decision Process(MDP) problem, so we consider using Q-learning

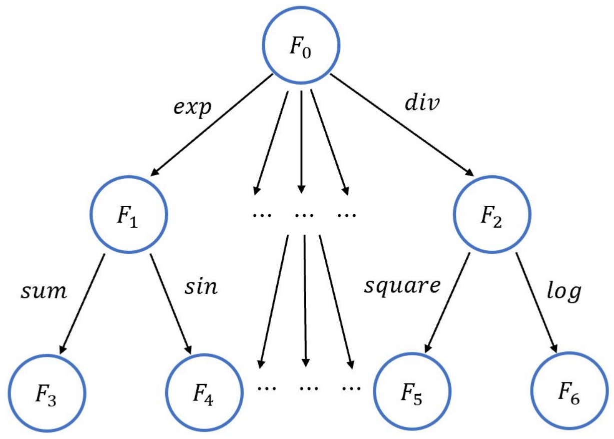


Figure 1. Example of a feature generation tree

as the learning algorithm. Besides, considering the huge features space and the large available operations 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(Mnih, 2018) 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 features set
- **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

So the peusocode of the Deep Q-learning Algorithm for feature generation tree is given as the following. We will update the NN weights as: w = w + $\alpha(A_L\{F_i, y\}) + \gamma \max_b Q(s_i, b, w^-) - Q(s_i, a_i, w))\nabla Q(s_i, a_i, w)$

In the algorithm, dim denotes the dimension of optimal generated features set F^*. So we always look for a F^* which can meet our threshold of accuracy with the lowest dimension. To piratically implement Deep Q-learning, we need a consistent features 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.

Besides, each episode will start at the state which is either to be some primary features(e.g. n primary features when $dim = n$) or random selected features which haven't attained c_{max}. Methods for choosing actions can either be

Algorithm 1 Deep Q-learning Algorithm for feature generation tree

```
Input: primary features set $F_0$, target vector $y$, operations set **H**
for $dim = 1$ to $dim_{max}$ do
  Construct_NN($dim * n_{samples}, action\_space$)
  repeat
    Initialize $state = Initial()$.
    for $i = 1$ to $c_{max}$ do
      $action = greedy\_method(state)$
      $next\_state, c = step(state, action)$
      $reward = accuracy_L(next\_state, y)$
      if $c > c_max$ then
        $done = True$
      else
        $done = False$
      end if
      Append $\{state, action, reward, next_state, done\}$ to $memory$
      $state = next\_state$
      exp_replay($memory$)
      if $done == True$ then
        break
      end if
    end for
    if $max\_Reward > threshold$ then
      End
    end if
  until $episodes > e_{max}$
end for
```

epsilon greedy or UCT. Both parts will be more specifically discussed in the following section.

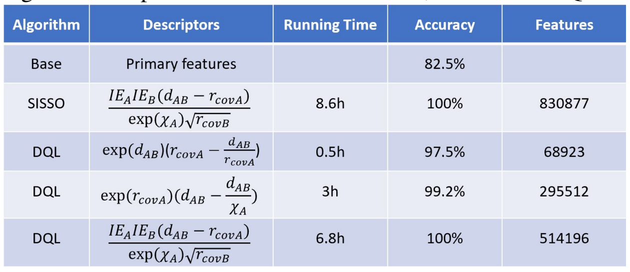
4.3. Experiments and Results

The experiments are implemented based on the NaCl prototype materials dataset from (Ouyang, 2018), which is composed of 132 samples and seven primary features $\{IE_A, IE_B, d_{AB}, r_{covA}, r_{covB}, X_A, X_B\}$. The problem is focusing on predicting the metal/non-metal property of NaCl prototype materials and so is a classification problem. We use Logistic Regression to calculate the accuracy as rewards and use a operation set including $\{exp, log,^2, \sqrt{}, -, \div\}$. In the following results, we will describe generating function $func_i()$ as descriptors.

As for the QDL coefficients setting, we set 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 table(Fig.2) we can see that

Figure 2. Comparison of results between F_0, SISSO and DQL



Algorithm	Descriptors	Running Time	Accuracy	Features
Base	Primary features		82.5%	
SISSO	$\frac{IE_A IE_B (d_{AB} - r_{covA})}{\exp(\chi_A)\sqrt{r_{covB}}}$	8.6h	100%	830877
DQL	$\exp(d_{AB})(r_{covA} - \frac{d_{AB}}{r_{covA}})$	0.5h	97.5%	68923
DQL	$\exp(r_{covA})(d_{AB} - \frac{d_{AB}}{\chi_A})$	3h	99.2%	295512
DQL	$\frac{IE_A IE_B (d_{AB} - r_{covA})}{\exp(\chi_A)\sqrt{r_{covB}}}$	6.8h	100%	514196

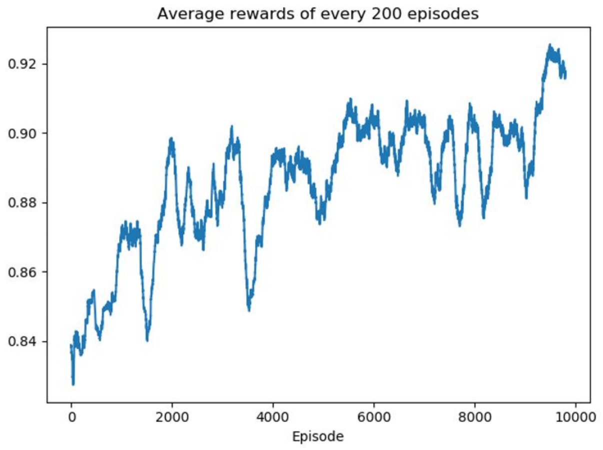


Figure 3. Average rewards of every 200 episodes for DQL learning process

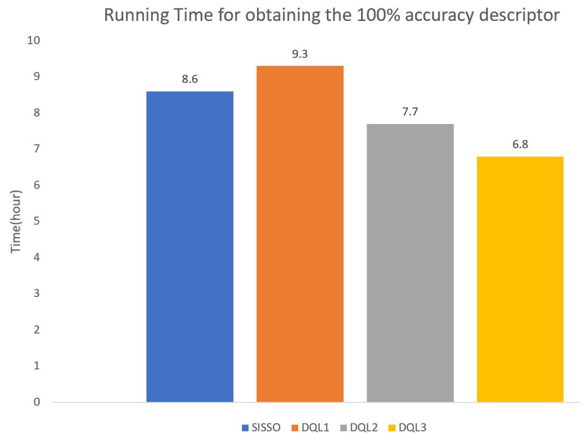


Figure 4. Comparison of running time between SISSO, DQL1 with root node of primary features, DQL2 with root node of random selected features and DQL3 with same root node of DQL2 and with UCT

compared with the primary features set $Base$, SISSO and DQL both can improve the classification accuracy(maximum of 100% v.s. 82.5%). However, SISSO has to take 8.6 hours to compute the overall features space of 830877 features to obtain the optimal descriptor of 100% accuracy. While in the first half hour, DQL can find the descriptor of 97.5% accuracy with only constructing 68,923 features. Then in the first 3 hours, DQL is able to find the descriptor of 99.2% accuracy with 295,512 features constructed. Finaly, 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 an obvious decrease in the constructed features space between SISSO's and DQL's, and also an evident saving-time of DQL. This meas by using DQL to explore the features generating tree can have effects in increasing the exploring efficiency and decreasing the computation scalability. Fig.3 shows the average rewards of every 200 episodes in the learning process of DQL.

Fig.4 shows a comparison results of running time between four kinds of algorithms: SISSO, DQL1 with root node of primary features, DQL2 with root node of random selected features and DQL3 with same root node of DQL2 and with UCT. 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, we can have a better running efficiency than SISSO. Besides, with the huge features space **F** ,the possible large operations 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-Bandit Problem (Gaudel & Sebag., 2010). To further strengthen the exploration process, we also replace the DQL2's epsilon greedy method with UCT method, resulting in the DQL3 algorithm. So we finally get the optimal results of DQL3, decreasing the total running time to obatain 100% accuracy descriptor of SISSO(8.6 Hours) to DQL(6.8 Hours).

5. Conclusions

In this paper, we presented a novel algorithm of AFM based on the feature generation tree exploration with Deep Q-learning. Unlike SISSO(Ouyang, 2018) which constructs the whole features space first, we transform the features construction process into a feature generation tree exploration problem. Also, unlike (Khurana & Turaga, 2018) which used a transformation graph for exploring the best set of generated features with variant features space and action space, we define our feature generation tree with consistent features space and action space for deep Q-learning and always aim to find the lowest dimension of the generated features set with the lowest complexity. The results showed that our algorithm have effects in decreasing the running time of SISSO by 20.9% and also decreasing the computation scalability of SISSO by 38.1% with the same dataset. The example code can be found through the link: https://github.tamu.edu/ziyu-xiang/AFM_RL

Though our algorithm optimized the running efficiency and computation scalability of SISSO and relaxed the correlation between Q values and actions of Linear Functional approximation (Khurana & Turaga, 2018) by using experience replay, our algorithm is still not perfect. In this algorithm, actions are defined as only the operation type, but not including the operating target featrues. The problems will occur for binaray operations(e.g.sum), for which we add all the existing features to the current nodes' features and select one generated features with the highest accuracy as the next state. This will introduce bias in exploration, since the accuracy may not be monotonic with more features added in, and also wastes lots of computation resources, since we only have preference on operation selection, but not on operation target selection. However, when we consider set actions as the pair of operations and targets, then the action space will constantly increase due to more and more generated features added to the target set. A large discrete action space (Dulac-Arnold, 2015) might be taken into consideration for this problem.

Acknowledgements

We thank Prof. Dileep Kalathil's and Prof. Xiaoning Qian's valuable guidance for this project and we acknowledge the support of TAMU's HPRC platform for implementing the experiments and obtaining the results.

References

Dulac-Arnold, Gabriel, e. a. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv*, (1512): 07679, 2015.

Gaudel, R. and Sebag., M. Feature selection as a one-player game. 2010.

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

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

Mnih, Volodymyr, e. a. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2018.

Ouyang, Runhai, e. a. Sisso: A compressed-sensing method for identifying the best low-dimensional descriptor in an immensity of offered candidates. *Physical Review Materials*, 2(8):083802, 2018.