Automatic Feature Engineering by Deep Reinforcement Learning

Extended Abstract

Jianyu Zhang, Jianye Hao*, Françoise Fogelman-Soulié, Zan Wang
College of Intelligence and Computing, Tianjin University
Tianjin, China
edzhang@tju.edu.cn,Jianye.hao@tju.edu.cn,francoise.soulie@outlook.com,wangzan@tju.edu.cn

ABSTRACT

We present a framework called *Learning Automatic Feature Engineering Machine* (LAFEM), which formalizes the *Feature Engineering* (FE) problem as an optimization problem over a *Heterogeneous Transformation Graph* (HTG). We propose a Deep Q-learning on HTG to support efficient learning of fine-grained and generalized FE policies that can transfer knowledge of engineering "good" features from a collection of datasets to other unseen datasets.

KEYWORDS

Innovative agents and multiagent applications; Deep learning; Feature generation

ACM Reference Format:
Jianyu Zhang, Jianye Hao*, Françoise Fogelman-Soulié, Zan Wang. 2019. Automatic Feature Engineering by Deep Reinforcement Learning. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 3 pages.

1 INTRODUCTION

Most of existing methods of automatic FE either generate a large set of possible features by predefined transformation operators followed by feature selection (Brute-force) [1, 4, 9] or apply simple Machine Learning / Reinforcement Learning (simple algorithm and/or simple meta-features derived from FE process) to recommend a potentially useful feature [2, 3, 3, 5]. The former makes the process computationally expensive, which is even worse for complex features, while the latter limits the performance boost.

A recently proposed FE approach [3] is based on RL. It treats all features in the dataset as a union, and applies traditional Q-learning [8] on FE examples to learn a strategy for automating FE under a given budget. Reinforcement Learning (RL) is a suitable way for solving the FE problem, but this work uses Q-learning with linear approximation with only 12 simple manual features, which limits the ability of automatic FE. Furthermore, it ignores the differences between features and applies a transformation operator on all of them at each step. Because of this nondiscrimination of different features, it is computation expensive, especially for large datasets and complex transformation operators.

To address the above limitations, in this work, we propose LAFEM (*Learning Automatic Feature Engineering Machine*), a novel approach for automatic FE based on Deep Reinforcement Learning (DRL). We define a *Heterogeneous Transformation Graph* (HTG), which is a heterogeneous directed acyclic graph representing relationships between different transformed versions of features and datasets, to organize the FE process.

* Corresponding author.
Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

2 METHOD

In this section, we present a new framework called LAFEM. It contains a *Heterogeneous Transformation Graph* (HTG) to represent the FE process in feature level and an off-policy RL algorithm on top of HTG to find a good FE policy. We consider a collection of datasets $\mathcal{D} = \{D^0, D^1, ..., D^N\}$ and each dataset D^i or (D for short) can be represented as $D = \langle F, y \rangle$ where $F = \{f_0, f_1, ..., f_n\}$ is a set of features and y is the corresponding target variable we want to predict. We apply a classification algorithm C (e.g. Random Forest) on dataset D and use an evaluation measure m (e.g. log-loss, F1-score) to measure the classification performance. We use $P_C^m(F, y)$ or $P(D)$ to denote the cross-validation performance of classification algorithm C and evaluation measure m on dataset D.

Since the FE process is time-consuming, especially *model evaluation*, in practice, we usually need to set a budget (e.g. time) for a particular FE problem. A budget B_{emax} here indicates the maximum count of *model evaluation* steps we can take for a dataset.

A transformation operator τ in FE is a function that is applied on a set of features to generate a new feature $f_+ = \tau(\{f_i\})$where the order of the operator is the number of features in $\{f_i\}$. We denote the set of derived features as F_+. For instance, a *product* transformation applied on two features (order 2) generates a new feature $f_+ = product(f_i, f_j)$. We use T to denote the set of operators.

2.1 Heterogeneous Transformation Graph

The Heterogeneous Transformation Graph (HTG) is a directed heterogeneous acyclic graph (Figure 1). Each node in HTG corresponds to either a feature (feature node) or a dataset (dataset node). Each feature node corresponds to either one original feature or one feature derived from original features. Each dataset node corresponds to either the original dataset D_0 or a dataset derived from it. The edges are divided into three categories: (a) Feature transformation edge, the edge between two feature nodes f_i and $f_j, i > j \geq 0$, indicates that f_i is a feature derived from f_j. If there are more than one feature nodes $\{f_j\}$ connecting to f_i, then f_i is a feature derived from all of them, i.e. $f_i = \tau(\{f_j\})$. (b) Dataset transformation edge, the edge between two dataset nodes D_i, $D_j, i > j \geq 0$ indicates that D_i is a dataset obtained from D_j by *feature generation, feature selection* or *model evaluation*. (c) The edge from feature node f_i

to dataset node D_k or in the inverse direction, indicates that f_i is added to D_k or dropped from D_k. Since we only allow one feature to be added / eliminated at one time, for any dataset node D_k, there is at most one feature node connected to one dataset node.

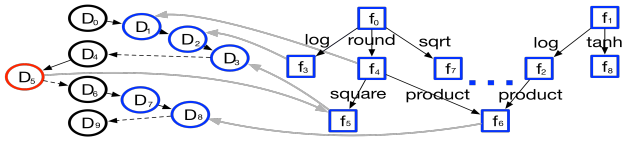

Figure 1: *Example of HTG.*

2.2 MDP Formulation

Consider the FE process with HTG on one dataset D as a Markov decision process (MDP) problem. At each time step t, a state $s_t \in \mathcal{S}$ consists of the HTG $\mathcal{G}_t$ and the *model evaluation* budget B_e available. $a_t \in \mathcal{A} = \mathcal{A}_G \bigcup \mathcal{A}_S \bigcup \mathcal{A}_E$ is an action from one of the following three groups of actions:

$\mathcal{A}_G$ is a set of actions for *feature generation*, which apply a transformation $\tau \in T$ on one or more features $\{f\}$ to derive one new feature and add it to dataset D_t. $\forall a \in \mathcal{A}_G, a = \langle \{f\}, \tau \rangle$, where $\{f\}$ is one or more features nodes in HTG. τ is a transformation operator from T.

$\mathcal{A}_S$ is a set of actions for *feature selection* by RL, which either drop one feature f from dataset $D_t = \langle F_t, y \rangle$ or add back one feature f that exists in HTG but not in current dataset D_t.

$\mathcal{A}_E$ contains one *model evaluation* action, which applies classification algorithm C and measures m on dataset D_t to get the performance of D_t as $P_C^m(D_t)$. Because we can only get the performance after the model evaluation action, we call the state after action in $\mathcal{A}_E$ the model evaluated state.

The reward r_t of this FE problem in HTG at time step t is:

$$r_t = \max_{i \in [0, t+1]} (P_C^m(D_i)) - \max_{j \in [0, t]} (P_C^m(D_j)) \tag{1}$$

2.3 LAFEM Algorithm

So far, we have introduced the organization of FE process and the MDP formulation of FE problem. The most critical part is the algorithm to find the optimal strategy of FE. We introduce LAFEM framework, an algorithm that can apply an *off-policy* DRL algorithm $\mathbb{A}$ (such as DQN, Double DQN [6], Dueling DQN [7]) on HTG to perform automatic FE.

To increase the generalization ability of the strategy learned from the LAFEM framework, we train the agent on many datasets to learn a good strategy. To prevent the algorithm from always applying *feature generation* or *feature selection* on the dataset and never evaluating the performance, we set a constraint B_{amax} on the maximum number of steps between two model evaluation states.

To train a generalization agent on many datasets, each time we apply ϵ-greedy strategy on one dataset D sampled from $\mathcal{D}$ and store transactions into *replay buffer* then use mini-batch gradient descent to update $\mathbb{A}$. We perform this process until convergence. The details of the algorithm are shown in Algorithm 1.

The reward function in Eq. (1) is the original reward for automatic FE problem in HTG. From Eq. (1), rewards only exist when we apply

Algorithm 1 LAFEM

input: a set of datasets $\mathcal{D} = \{D\}$, replay buffer R, Budget B_{emax}, and B_{amax}, an off-policy DRL algorithm $\mathbb{A}$

while not done **do**
  Bootstrap sample a dataset D from $\mathcal{D}$
  Initialize budgets: $B_e \leftarrow B_{emax}, B_a \leftarrow B_{amax}$
  while $B_e > 0$ **do**
    if $B_a > 0$ **then**
      Get an action a_t by ϵ-greedy and execute a_t
    else
      Choose action $a_t \in \mathcal{A}_E$ and execuate a_t
    end if
    Store the transition in R and set $B_a \leftarrow B_a - 1$
    if $a_t \in \mathcal{A}_E$ **then**
      Set $B_e \leftarrow B_e - 1, B_a \leftarrow B_{amax}$
    end if
  end while
  while not done **do**
    Sample a mini-batch from replay buffer R
    Perform one optimization step on $\mathbb{A}$
  end while
end while

model evaluation so any *feature generation* or *feature selection* step would never have any immediately reward. As a result, the reward would be really sparse and delayed, so-called delayed reward. To solve this problem, we design reward shaping by modifying the value of $P_C^m(D_t)$ where D_t is not an evaluation state as:

$$P_C^m(D_t) = \frac{(P_C^m(D_{t+\alpha}) - P_C^m(D_{t-\beta}))}{\alpha + \beta + 1} \tag{2}$$

where α and β are the number of steps to next model evaluated state and last model evaluated state, respectively. Hence, *feature generation* and *feature selection* action $a \in \mathcal{A}_G \bigcup \mathcal{A}_S$ can gain immediate reward at each step. In most datasets, the framework outperforms state-of-the-art automatic FE approaches in both model performance and time efficiency for both simple and complex transformation operators.

3 CONCLUSION

In this paper, we present a novel framework called LAFEM to perform automatic feature engineering (FE) which combines *feature generation, feature selection* and *model evaluation*. It includes a heterogeneous transformation graph (HTG) that organized the processing of FE and a Deep Reinforcement Learning algorithm that can perform automatic FE on the HTG.

ACKNOWLEDGEMENT

The work is supported by the National Natural Science Foundation of China (Grant Nos.: 61702362, U1836214), Special Program of Artificial Intelligence, Tianjin Research Program of Application Foundation and Advanced Technology (No.: 16JCQNJC00100), and Special Program of Artificial Intelligence of Tianjin Municipal Science and Technology Commission (No.: 569 17ZXRGGX00150)

REFERENCES

[1] James Max Kanter and Kalyan Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*. IEEE, 1–10.

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

[3] Udayan Khurana, Horst Samulowitz, and Deepak Turaga. 2017. Feature Engineering for Predictive Modeling using Reinforcement Learning. *arXiv preprint arXiv:1709.07150* (2017).

[4] Hoang Thanh Lam, Johann-Michael Thiebaut, Mathieu Sinn, Bei Chen, Tiep Mai, and Oznur Alkan. 2017. One button machine for automating feature engineering in relational databases. *arXiv preprint arXiv:1706.00327* (2017).

[5] Fatemeh Nargesian, Horst Samulowitz, Udayan Khurana, Elias B Khalil, and Deepak Turaga. 2017. Learning feature engineering for classification. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI*, Vol. 17. 2529–2535.

[6] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning.. In *AAAI*, Vol. 2. Phoenix, AZ, 5.

[7] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. 2015. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581* (2015).

[8] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.

[9] Jianyu Zhang, Françoise Fogelman-Soulié, and Christine Largeron. 2018. Towards Automatic Complex Feature Engineering. In *International Conference on Web Information Systems Engineering*. Springer, 312–322.