Cross-data Automatic Feature Engineering via Meta-learning and Reinforcement Learning

Jianyu Zhang[1(✉)], Jianye Hao[1], and Françoise Fogelman-Soulié[2]

[1] College of Intelligence and Computing, Tianjin University, Tianjin, China
{edzhang,jianye.hao}@tju.edu.cn
[2] Hub France IA, Paris, France
francoise.soulie@hub-franceia.fr

Abstract. Feature Engineering (FE) is one of the most beneficial, yet most difficult and time-consuming tasks of machine learning projects, and requires strong expert knowledge. It is thus significant to design generalized ways to perform FE. The primary difficulties arise from the multiform information to consider, the potentially infinite number of possible features and the high computational cost of feature generation and evaluation. We present a framework called *Cross-data Automatic Feature Engineering Machine* (CAFEM), which formalizes the FE problem as an optimization problem over a *Feature Transformation Graph* (FTG). CAFEM contains two components: a FE learner (FeL) that learns fine-grained FE strategies on one single dataset by Double Deep Q-learning (DDQN) and a Cross-data Component (CdC) that speeds up FE learning on an unseen dataset by the generalized FE policies learned by Meta-Learning on a collection of datasets. We compare the performance of FeL with several existing state-of-the-art automatic FE techniques on a large collection of datasets. It shows that FeL outperforms existing approaches and is robust on the selection of learning algorithms. Further experiments also show that CdC can not only speed up FE learning but also increase learning performance.

1 Introduction

As machine learning becomes more and more widespread, it has been recognized that feature engineering (FE) is the most critical factor for models performance [1]. Various researchers have demonstrated the benefit of using additional features [11]. FE aims at reducing the model error and making learning easier by deriving, through mathematical functions (operators), new features from the original ones. Normally a data scientist combines feature generation, selection and model evaluation iteratively, generating a long sequence of decisions before obtaining the "optimal" set of derived features. This process heavily relies on expert domain knowledge, intuition and technical expertise to handle the complex feedbacks and make best decisions. As a result, the process is difficult, time-consuming and hard to automate.

H. W. Lauw et al. (Eds.): PAKDD 2020, LNAI 12084, pp. 818–829, 2020.
https://doi.org/10.1007/978-3-030-47426-3_63

Most of existing methods of automatic FE either generate a large set of possible features by predefined transformation operators followed by feature selection [3,7,15] or apply simple supervised learning (simple algorithm and/or simple meta-features derived from FE process) to recommend a potentially useful feature [4,5,9]. The former makes the process computationally expensive, which is even worse for complex features, while the latter significantly limits the performance boost.

A recently proposed FE approach [5] is based on Reinforcement Learning (RL). It treats all features in the dataset as a union, then applies traditional Q-learning [14] on FE-augmented examples to learn a strategy for automating FE under a given computing budget. RL is more promising in providing general FE solutions. However, this work uses Q-learning with linear approximation and 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 FeL (*Feature Engineering Learner*) and CAFEM (*Cross-data Automatic Feature Engineering Machine*). The former is a novel approach for automatic FE for one particular dataset based on off-policy Deep Reinforcement Learning (DRL). In order to speed up the FE process and take advantage of the FE knowledge learned from a large set of datasets, the latter extends FeL to cross-data level by Meta-Learning.

We define a *Feature Transformation Graph* (FTG), a directed graph representing relationships between different transformed versions of features, to organize the FE process. FeL sequentially trains an agent for each feature by DRL algorithms to learn the strategy for feature engineering on one dataset and corresponding FTG representation. We thus view the goal of FE as maximizing model accuracy by searching through a set of features F_+ to generate and a set of features F_- to eliminate. CAFEM extends this process to cross-data by training one agent on a large set of datasets to enable the learned policy to perform well on unseen datasets.

2 Background and Problem Formulation

In this section we review the Reinforcement Learning (RL) [10] background and describe the problem formulation.

2.1 Reinforcement Learning

RL is a family of algorithms that formalizes the interaction of an agent $\mathbb{A}$ with her environment using a Markov Decision Process (MDP) and allows it to devise an optimal sequence of actions. An MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ a set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ a transition function that maps each state-action pair to a probability distribution over the

possible successor states, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ a reward function and $\gamma \in [0, 1]$ a discount factor for controlling the importance of future rewards. A policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ is a mapping from states to actions. At every time step t, an agent in state s_t produces an action $a_t = \pi(s_t)$. Based on transition function $\mathcal{T}$ the agent gets into next state s_{t+1} with probability $\mathcal{T}(s_t, a_t)$ and obtains immediate reward $r_t = \mathcal{R}(s_t, a_t, s_{t+1})$. The goal of an agent is to find an optimal policy π^* maximizing her expected discounted cumulated reward $\mathbb{E}[R_0|s_0]$, where $R_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i$ is the discounted sum of future rewards.

Q-learning is a well-known model-free RL algorithm for finding an optimal policy π^* for any finite MDP. In Q-learning we define the Q-function or action-value function as $Q(s_t, a_t) = \mathbb{E}[R_t|s_t, a_t]$.

Given an optimal policy π^*, we are interested in the optimal function $Q^{\pi^*}(s, a)$, or $Q^*(s, a)$ for short, where $\forall \pi, s \in \mathcal{S}, a \in \mathcal{A}, Q^{\pi^*}(s, a) \geq Q^{\pi}(s, a)$. As a result, Q^* satisfies the following equation:

$$Q^*(s_t, a_t) = r_t + \mathop{\mathbb{E}}_{s_{t+1} \sim \mathcal{T}(s_t, a_t)} [\gamma \max_a Q(s_{t+1}, a)] \tag{1}$$

Double Deep Q-network (DDQN) [12] is a model-free RL algorithm, which estimates the state-action value approximately through a deep neural network with parameters θ. It uses an ϵ-greedy policy to get the next action.

During training, the tuples $\langle s_t, a_t, r_t, s_{t+1} \rangle$ generated by the ϵ-greedy policy are stored in R, the so-called *replay buffer*. Then the neural network is trained by sampling from the *replay buffer*, using mini-batch, and performing gradient descent on loss $\mathcal{L} = \mathbb{E}((Q(s_t, a_t) - y_t)^2)$, where $y_t = r_t + \gamma \max_a Q(s_{t+1}, a), Q(s, t)$ is approximated by the network g with parameter θ.

2.2 Meta-learning

The goal of meta-learning is to quickly train a model for a new task with the help of data from many other similar tasks.

Model-Agnostic Meta-Learning (MAML) [2] is one of the best meta-learning algorithms that were trained by gradient descent. We denote $\{T\}$ as a set of tasks. MAML performs one step gradient descent for a task T_i on loss $\mathcal{L}$ with network g and network parameters θ and gains $\theta_i^{'}$ as Equation (2). Then it performs a second gradient descent $\bigtriangledown_\theta$ step on loss $\mathcal{L}$ with network parameters $\theta_i^{'}$ as Equation (3). Finally, MAML finds parameters θ that are close to the optimal parameters of every task.

$$\theta_i^{'} = \theta - \alpha \bigtriangledown_\theta \mathcal{L}_{T_i}(g_\theta) \tag{2}$$

where α is the learning rate of each task T_i.

$$\theta = \theta - \beta \bigtriangledown_\theta \sum_{T_i \in \{T\}} \mathcal{L}_{T_i}(g_{\theta_i^{'}}) \tag{3}$$

where β is the meta step size.

2.3 Problem Formulation

We consider a collection of typical supervised learning tasks (binary classification or regression) $T = \{T_1, T_2, ..., T_N\}$ and each task T_i can be represented as $T_i = \langle D, L, m \rangle$, where $D = \langle F, y \rangle$ is a dataset with a set of features $F = \{f_1, f_2, ..., f_n\}$ and a corresponding target variable y, L is a learning algorithm (e.g. Random Forest, Logistic Regression, Neural Network) to be applied on dataset D and m is an evaluation measure (e.g. log-loss, relevant absolute error, f1-score) to measure the performance.

We use $P_L^m(F, y)$ or $P(D)$ to denote the cross-validation performance of learning algorithm L and evaluation measure m on dataset D. The goal of each task is to maximize $P(D)$.

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 follows 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 $\mathbb{T}$ to denote the set of all transformation operators.

Feature engineering aims at constructing a subset of features $F^* = F_o \bigcup F_+ - F_-$, where F_o is the set of original features in dataset D, F_+ the set of derived features and $F_- \subseteq F_o$ the set of features that we decide to drop out from original features. For a given dataset D, a feature engineering strategy π specifies a derived feature set $F^* = \pi(D)$, where $F^* = F_o \bigcup F_+ - F_-$. The goal of feature engineering is to find a good policy π^* that maximizes the model performance for a given algorithm L and measure m on a dataset D.

$$\pi^* = \arg\max_{\pi} P_C^m(\pi(D), y) \tag{4}$$

3 Method

In this section, we present a new framework called *Cross-data Automatic Feature Engineering Machine* (CAFEM). In order to highlight the differences between features and integrate *feature generation* and *feature selection* effectively, we propose a *Feature Transformation Graph* (FTG) to represent the FE process at feature level. Based on FTG, CAFEM can perform feature engineering for each particular feature based on the information related with it. Thus, it avoids the drawback of generating a large set of features at each step in [5], especially for complex features and large number of features. One component of CAFEM called FE Learner (FeL) uses Reinforcement Learning to find the optimal feature set F^* for each feature iteratively, instead of using expensive graph search algorithm [6]. FeL focus on one particular supervised learning task which gives FeL the ability to dig deeply into that task. However, it loses the opportunity to learn and integrate useful experiments from other tasks which can speed up FE process on a similar task. In order to balance performance and speed, another component of CAFEM called Cross-data Component (CdC) applies a Model-Agnostic

Meta-Learning (MAML) [2] method, which is originally designed for supervised learning and on-policy reinforcement learning algorithms, on off-policy reinforcement learning algorithms to speed up FE learning on one particular dataset by integrating the FE knowledges from a set of datasets.

3.1 Feature Transformation Graph

We propose a structure called *Feature Transformation Graph* (FTG) G, which is a directed acyclic dynamic graph, to represent the FE process. Each node f in FTG corresponds to either one original feature in F_o or one feature derived from original features. An edge from node f_i to f_j, $j > i > 0$, with label τ indicates that feature f_j is transformed from feature f_i by transformation operator τ, e.g. $f_{n+2} = square(f_{n+1})$ or transformed partially from f_i by τ, e.g. $f_{n+8} = product(f_{n+4}, f_{n+6})$. At the start of FE, G contains n nodes which correspond to n original features $\{f_1, f_2, ..., f_n\}$. As FE process goes, FTG dynamically grows up (adds more nodes and edges). So we denote FTG at time step t as G_t. An illustrating example is given in Fig. 1.

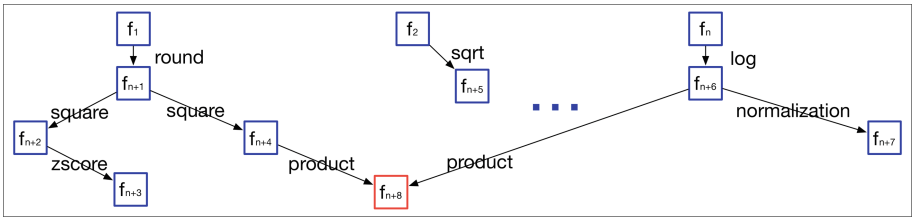

Fig. 1. *Example of FTG*

3.2 MDP Formulation

So far, we have introduced the representation of FE with FTG in our automatic FE framework. After that, what we need to do is to find a suitable strategy to control the growth of FTG. An important property is that FTG is not designed for any particular strategy, but to be a general representation of an FE process. As a result, we can apply many different strategies on the FTG to control it, such as graph search or RL. In this paper, we choose RL to learn a strategy that can make a sequence of decisions on top of FTG, due to its efficiency.

Consider the FE process with FTG on one dataset D as an MDP problem defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$. At each time step t, a state $s_t \in \mathcal{S}$ consists of the Feature Transformation graph G_t and the features $\{f_t\}$ we are working on. Due to the complexity of transformation operators, $\{f_t\}$ could contain one or more features. For example, $\{f_t\}$ contains one feature for Order-1 operators (e.g. $log, square$), two features for Order-2 operators (e.g. $product, sum$).

An action $a_t \in \mathcal{A} = \mathcal{A}_G \bigcup \mathcal{A}_S$ comes from the following two groups of actions:

- $\mathcal{A}_G$ is a set of actions for *feature generation*, which apply a transformation $\tau \in \mathbb{T}$ on current features $\{f_t\}$ to derive one new feature.
- $\mathcal{A}_S$ contains one action for *feature selection* by RL, which drops current feature f_t and moves back to the previous feature. One special case is that current feature $f_t \in \{f_1, f_2, .., f_n\}$ belongs to original features. In this case, *feature selection* action drops it and stops current FE process.

The learning objective here is to find a state s_i with feature set F^* in FTG that maximizes the model accuracy $P_C^m(F^*, y)$. The trajectory from original feature to a new feature f_i indicates the final feature engineering strategy for f_i.

Since the target of FE is to maximize the performance $P(D)$, the reward r_t of this FE problem in FTG at time step t is set as:

$$r_t = P_L^m(D_{t+1}) - P_L^m(D_t) \tag{5}$$

3.3 CAFEM Framework

Until now, we have introduced the organization of FE process and the MDP formulation of FE problem. The most critical part is the algorithm to find a good strategy of FE. We introduce CAFEM framework which mainly contains two parts: 1) an algorithm called FeL that can apply an off-policy DRL algorithm $\mathbb{A}$ (such as DQN [8], Double DQN [12]) on FTG for one particular dataset to perform automatic FE; 2) an extended version of model-agnostic meta-learning [2] algorithm on off-policy DRL to speed up FE learning by taking advantage of the generalized FE strategies learned from a set of datasets. It is called off-policy, since the policy being learned can be different from the policy being executed.

In the following sub-section, we will introduce the details of these two parts.

Feature Engineering Learner (FeL): Although FeL works as a component of CAFEM in this paper, it is also a complete algorithm that sequentially optimizes FE strategies for each feature on one particular dataset. The details of FeL algorithm are shown in Algorithm 1. Given a supervised learning task T with n features $F = \{f_1, f_2, ..., f_n\}$, n off-policy DRL agents $\{\mathbb{A}_i\}$, FeL sequentially optimizes a FE policy for each feature (line 2 in Algorithm 1). As traditional training stage of off-policy RL algorithms, FeL starts with performing M episodes of FE process by ϵ-greedy and stores corresponding transitions in replay buffer (line 3–10). In this process, FeL either generates a new feature $\hat{f}$ from feature f by action a_t ($a_t \in \mathcal{A}_G$) or drops current feature f and moves back to previous feature $\hat{f}$ ($a_t \in \mathcal{A}_S$). Then FeL trains the corresponding agent $\mathbb{A}_i$ by performing gradient descent on a mini-batch sampled from *replay buffer* R_i (line 11–14). During test stage the same FE method as Algorithm 1 with $\epsilon = 0$ is used to perform FE for each feature sequentially. Note that the operators in transformation operators set $\mathbb{T}$ are not of the same complexity level. For example, some unary features (e.g. $log(f_i), square(f_i)$) are less complex than binary features (e.g. $product(f_i, f_j), sum(f_i, f_j)$).

As in [15], we introduce features along feature complexity, driving simple features first (e.g. unary features) then complex features (e.g. binary features).

Algorithm 1. FeL

input: An dataset $D = \langle F, y \rangle$ with n features $F = \{f_0, ...f_n\}$, n replay buffer $\{R_i, ..., R_n\}$ for each features, n off-policy DRL agents $\{\mathbb{A}_i\}$, number of epochs and episodes E, M, batches to train N

```
1: while epoch = 1, E do
2:     for f_i in F do
3:         for episode = 1, M do
4:             Get initial state s_0
5:             while not terminal do
6:                 Get an action a_t by ε-greedy and execute a_t on f: f̂ = a_t(f)
7:                 Obtain reward r_t and next state s_{t+1}
8:                 Store transition (s_t, a_t, r_t, s_{t+1}) in R_i and reset current feature: f ← f̂
9:             end while
10:        end for
11:        for t = 1, N do
12:            Sample a mini-batch from replay buffer R_i
13:            Perform one optimization step on A_i
14:        end for
15:    end for
16:    Reset dataset D = ⟨{f_i, ..., f_n}, y⟩
17: end while
```

Cross-data Component: In order to speed up FE process and take advantage of a large set of datasets, we apply Model-agnostic Meta-Learning [2] on off-policy RL to perform cross-data level automatic feature engineering. The details of the Cross-data Component (CdC) are shown in Algorithm 2. Given a set of datasets $\{D = \langle F, y \rangle\}$ and an off-policy RL agent $\mathbb{A}$ (we use DDQN here as it can gain relevant a good performance in many tasks [12]) represented by g_θ, Cross-data Component samples a batch of features $\{f_i\}$ and corresponding dataset $\{D_{f_i}\}$ and constructs a batch of supervised learning tasks $\{T_i = \langle D_{f_i}, P, m \rangle\}$ (line 2). For each task $T_i \in \{T_i\}$, CdC uses the RL agent $\mathbb{A}$ together with ϵ-greedy exploration to perform M episodes for T_i and stores the corresponding transitions in *replay buffer* R_i (line 4–5). Then CdC samples K transitions from R_i and computes one step gradient descent as Algorithm 2 (line 7–8) where the loss $\mathcal{L}$ is the same as Algorithm 1. Finally, we sample a batch of transitions and perform meta-update (line 9–11).

Network Design: Until now, we have discussed the details of FeL algorithm and cross-data component. One remaining part is the structure of the neural network that can approximate the Q-values of DDQN in FeL algorithm. In this project, instead of building one approximation function with parameter θ for each action a [5], we use one union function that is approximated by a neural network, for all actions. Thus, we only need to train one DRL model.

As we discussed in Sect. 4.2, the state s_t at time t indicates the FTG G_t and the features $\{f_t\}$ it is working on at time t. In order to cover these two parts of information in the representation of each state s_t, we use the following features to represent s_t:

Algorithm 2. CrossDataComponent

input: a set of tasks $\{T\}$, an off-policy DRL agent represented by g_θ, number of epochs and episodes E, M

Randomly initialize θ
while epoch = 1, E **do**
  Sample batch of tasks $\{T_i\}$ from $\{T\}$
  for all $\{T_i\}$ **do**
    Perform M episodes on task T_i with ϵ-greedy
    Store all transitions in R_i
    Sample K transitions $\mathcal{T}$ from R_i
    Compute adapted parameters with gradient descent: $\theta_i^{'} = \theta - \alpha \bigtriangledown_\theta \mathcal{L}_{T_i}(f_\theta)$
    Sample transitions $\mathcal{T}_i^{'}$ from R_i for meta-update
  end for
  Update $\theta = \theta - \beta \bigtriangledown_\theta \sum_{T_i \sim \{T\}} \mathcal{L}_{T_i}(f_{\theta_i^{'}})$ using each $\mathcal{T}_i^{'}$ and $\mathcal{L}_{T_i}$ in Equation 3
end while

1. Extended Quantile Sketch Array (ExQSA) representation of features. Quantile Sketch Array (QSA) uses *quantile* data sketch [13] to represent feature values associated with a class label. For each feature f and binary target y, QSA builds equi-width bins for f with target $y = 0$ and $y = 1$ separately. For regression problems, we extend QSA (ExQSA) by building equi-width bins for f with numeric target $y > median(y)$ and $y < median(y)$ separately.
2. Previous N-step FE history on FTG.
3. The number of each transformation operators used in G_t.
4. The number of next node visited for each action.
5. The number of each operator used from $\{f_t\}$ to its root.
6. Node depth of a feature in HTG.
7. Average performance improvement of each action.

Totally, we use 293 features to represent each state. A neural network with three fully connected hidden layers (128-128-64 neurons) and ReLU activation function is used to approximate Q-values.

4 Experiments

This section describes our experimental results. First, we introduce our experimental settings as well as our training procedure. Then we use F1-score (for classification) and 1 - Relevant Absolute Error (1-RAE) (for regression) criteria to compare the performance of FeL algorithm with several state-of-the-art automatic FE techniques. After that, we evaluate the robustness of our algorithm with respect to different learning algorithms (Random Forest, Logistic Regression). Finally, we show the efficiency of CAFEM on different supervised learning tasks by comparing it with FeL. To our surprise, CAFEM can help improving the prediction performance. Source codes are posted on Github (https://github.com/TjuJianyu/CAFEM.git).

4.1 Experimental Settings

We randomly collect 120 binary classification or regression datasets, which do not contain missing values and too many features and instances, from OpenML. We randomly split them into 100 datasets for training and the other 20 datasets for testing. Following [5,9], we choose 13 transformation operators (set $\mathbb{T}$) including Order-1: *Log, Round, Sigmoid, Tanh, Square, Square Root, ZScore, Min-Max-Normalization* and Order-2: *Sum, Difference, Product, Division*. Following [9], we choose Random Forest and Logistic Regression (Lasso for regression) (from Scikit-learn http://scikit-learn.org) as our learning algorithm and use F1-score/1-RAE to measure the performance. A 5-fold cross validation (same seed for all experiments) using random stratified sampling is used to measure the average performance. FeL is performed on 20 testing datasets directly, while CAFEM is trained on the 100 training datasets by meta-learning. For Order-2 operators, as the number of candidate features is very large, FeL randomly sample a small batch (100) at each step.

To showcase the ability of different FE algorithms, we compare the performance of FeL with the following approaches:

- **Baseline:** applies learning algorithm on original dataset (features) directly.
- **Random-FeL (RS):** is an algorithm where we apply random strategy on FTG rather than the strategy learned by RL like CAFEM to find a set of features that can maximize $P(D)$. This shows the effect of FTG without RL and Meta-learning. This algorithm can be seen as random graph search method on FTG. As some graph search algorithms, such as depth-first search (DFS) or breadth-search algorithm (BFS), are extremely time consuming [5], we do not compare FeL with DFS or BFS in this paper.
- **Brute-force (BF):** is inspired by DSM [3], OneBM [7] and [15]. It applies all transformation operators to all original features and performs feature selection on the augmented dataset. (top-down approach).
- **LFE** [9]**:** uses QSA to generate the representation of each feature in classification problems. Following [9], a neural network with one hidden layer, $L2$ regularization and dropout is used to predict whether a feature with a transformation operator will gain 1% model performance improvement.
- **FERL:** organizes the FE problem into a Transformation Graph, where each node is either the original dataset D or a dataset transformed from D. Then it uses Q-learning with linear approximation. We use the same setting as [5]. For Order-2 transformation operators, native FERL is extremely computation expensive since the number of new features is very large. During training stage, we prune the branches in Transformation Graph that would generate more than 10,000 new features next to make it trainable.

As the source codes of all these methods are not publicly available and some experiments details are not provided (such as, the random seed of learning algorithm and train-test dataset splitting), we implemented ourselves all the benchmarks. For all the FE approaches except Baseline, we evaluate the performance for Order-1 and (Order-1 & Order-2) transformation operators to compare the ability of handling simple and complex transformation operators.

Table 1. Comparing Performance by F1-score/1-RAE, Random Forest and 5-fold Cross-validation (- indicates cannot finish within 36 h, x indicates the algorithm can not handle corresponding dataset).

Datasets	#Row	#Feature	Baseline	Order-1					Order-1 & 2				
				FeL	BF	LFE	RS	FERL	FeL	BF	LFE	RS	FERL
Balance_scale	625	5	88.2%	88.3%	86.4%	88.2%	88.2%	**88.6%**	95.0%	**97.0%**	95.1%	92.7%	-
Boston	506	21	88.2%	**90.2%**	86.7%	89.2%	89.5%	88.7%	**89.9%**	85.6%	88.2%	89.8%	-
ClimateModel	540	21	95.5%	**96.0%**	95.6%	95.5%	95.7%	95.9%	**96.1%**	95.5%	95.5%	**96.1%**	-
Cpu_small	8,192	13	86.3%	**87.1%**	84.5%	85.8%	86.6%	86.8%	**87.1%**	86.2%	86.3%	87.0%	-
Credit card	14,240	31	50.5%	**68.7%**	64.8%	50.5%	63.8%	64.0%	**71.4%**	65.1%	65.1%	64.6%	-
Disclosure_x	662	4	44.8%	**51.7%**	46.6%	46.8%	49.7%	49.8%	51.4%	46.4%	46.4%	51.4%	**51.8%**
Disclosure_z	662	4	53.8%	**57.7%**	55.6%	53.1%	55.6%	57.0%	**57.0%**	53.8%	55.0%	56.7%	56.9%
fri_c1_1000_25	1,000	26	84.9%	87.7%	85.8%	85.8%	86.7%	**88.0%**	**87.1%**	77.9%	82.1%	**87.1%**	-
Fri_c2_100_10	1,000	11	86.3%	**89.7%**	85.8%	86.8%	88.6%	89.3%	**91.0%**	87.2%	86.7%	89.3%	-
Fri_c3_100_5	1,000	6	88.2%	89.2%	88.5%	88.2%	88.4%	**89.4%**	**90.7%**	87.3%	87.1%	89.3%	-
fri_c3_1000_50	1,000	51	79.7%	83.7%	**88.5%**	80.9%	80.7%	87.8%	83.1%	**88.4%**	78.3%	80.8%	-
Gina_agnostic	3,468	971	92.3%	92.8%	78.9%	92.3%	92.8%	**93.5%**	**92.8%**	-	92.5%	92.8%	-
Hill-valley	1,212	101	57.5%	**61.7%**	59.2%	57.5%	60.8%	61.1%	**100%**	**100%**	57.5%	99.9%	-
Ilpd	583	11	41.3%	**45.7%**	38.7%	38.9%	43.6%	44.9%	**45.9%**	**45.9%**	42.4%	44.8%	-
Kc1	2,109	22	40.4%	**44.5%**	35.3%	38.9%	42.0%	42.7%	**44.4%**	39.9%	38.8%	43.4%	-
openml_589	1,000	25	66.9%	67.7%	55.0%	X	67.2%	**72.6%**	75.0%	**76.9%**	X	68.1%	-
Pc4	1,458	38	47.7%	57.0%	36.2%	45.3%	53.8%	**58.4%**	**58.1%**	50.1%	55.1%	56.5%	-
Pc3+C14	1,563	38	25.9%	**33.4%**	27.9%	23.0%	30.3%	32.0%	**33.3%**	24.6%	27.4%	31.6%	-
Spectrometer	531	103	77.3%	**83.9%**	80.0%	75.2%	80.4%	83.0%	82.7%	**90.8%**	73.2%	81.8%	-
Strikes	625	7	96.6%	**99.5%**	98.7%	97.8%	99.1%	98.9%	**99.5%**	97.8%	93.4%	99.4%	98.9%

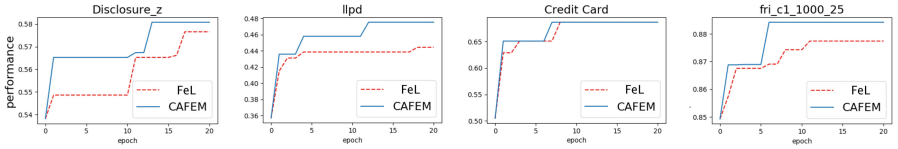

Fig. 2. *CAFEM vs FeL over 4 different datasets*

4.2 Performance Comparison of FeL

Table 1 compares the model performance of our automatic FE approach FeL to other state-of-the-art FE approaches on 20 datasets. The first four columns in this table report the dataset, the number of instances (rows) and original features, the baseline performance (F1-score/1-RAE of 5-fold cross validation) of the datasets. The number of instances ranges from 506 to 14,240 and for features, it ranges from 4 to 971. In the middle five columns, we compare different automatic FE approaches with Order-1 transformation operators, and in the last five columns, the performance with Order-1 & Order-2 transformation operators.

In Order-1 transformation operators, FeL outperforms all approaches on most datasets. On average, FeL improves performance by 4.2% on test datasets. In the best two cases, Credit card and Pc4, FeL even improves baseline performance by 18.2% and 9.3%. One interesting phenomenon is that the Random method (random graph search on FTG) can obtain a relevant higher performance on

some datasets. This indicates that FTG represents the FE process in an effective way and significantly contributes further strategy learning of FE.

On Order-1 & Order-2 transformation operators, the complexity of FE increases significantly. Thus, it is expected that an inefficient method would easily run out of time, memory space or even would not work. FeL improves performance by 6.9% on average compared with existing approaches. In the best two cases, Pc4 and Ilpd, it improves by 42.5% and 20.9%. As we mentioned above, some FE approaches would be strongly limited as the complexity of transformation operators increasing. Comparing the performance of each approaches on Order-1 & Order-2 with that on Order-1, we found that LFE and Brute-force approaches get a worse performance (-1.54% in average) on half of the datasets, while FeL does not get any performance decrease. FERL approach is really computation expensive here: most of the datasets run out of time (36 h).

4.3 Robustness of FeL on Different Learning Algorithms

In order to showcase the robustness of FeL, we evaluate the performance of FeL with two learning algorithms: Random Forest (tree-based ensemble learning algorithm) and Logistic Regression (Lasso for regression) (general linear algorithm) on 20 test datasets. FeL gains 10.8% and 4.2% performance increase on average with Logistic regression and Random Forest, respectively. The performance of FeL with Logistic regression ranges from 0.2% to 25.8%. For Random Forest, the performance of FeL ranges from 0% to 18.2%. It shows that our algorithm is robust with respect to different learning algorithms.

4.4 Performance of Cross-data Component

One main aim of Cross-data Component is to speed up FE learning. We evaluate FeL and CAFEM on the test datasets and randomly show the comparison on 4 datasets due to the space limitation. Figure 2 shows that CAFEM can increase model performance more rapidly (gain a high score within the first epoch, outperform the best of FeL within around ten epochs). To our surprise, CAFEM can gain a better final model performance than FeL in most of the cases. We hypothesize the reason of this phenomena as that CAFEM learnt some general FE rules from a large set of datasets to help the agent quickly learn a new dataset and regularize its behavior.

5 Conclusion

In this paper, we present a novel framework called CAFEM to perform automatic feature engineering (FE) and transfer FE experiences from a set of datasets to a particular one. It contains a feature transformation graph (FTG) that organized the process of FE, a Single-data FE learner and a Cross-data component. In most datasets, the framework outperforms state-of-the-art automatic FE approaches for both simple and complex transformation operators. With the

help of cross-data component, CAFEM can speed up FE and increase FE performance. Moveover, the framework is robust to the choice of different learning algorithms.

Acknowledgments. The work is supported by the National Natural Science Foundation of China (Grant Nos.: 61702362, U1836214).

References

1. Domingos, P.: A few useful things to know about machine learning. Commun. ACM **55**(10), 78–87 (2012)
2. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: Proceedings of the 34th International Conference on Machine Learning, vol. 70, pp. 1126–1135 (2017). JMLR.org
3. Kanter, J.M., Veeramachaneni, K.: Deep feature synthesis: towards automating data science endeavors. In: IEEE International Conference on Data Science and Advanced Analytics (DSAA), vol. 36678, pp. 1–10. IEEE (2015)
4. Katz, G., Shin, E.C.R., Song, D.: Explorekit: automatic feature generation and selection. In: Proceedings of the IEEE 16th International Conference on Data Mining ICDM 2016, pp. 979–984. IEEE (2016)
5. Khurana, U., Samulowitz, H., Turaga, D.: Feature engineering for predictive modeling using reinforcement learning. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
6. Khurana, U., Turaga, D., Samulowitz, H., Parthasrathy, S.: Cognito: automated feature engineering for supervised learning. In: Proceedings of the IEEE 16th International Conference on Data Mining Workshops ICDMW 2016, pp. 1304–1307. IEEE (2016)
7. Lam, H.T., Thiebaut, J.-M., Sinn, M., Chen, B., Mai, T., Alkan, O.: One button machine for automating feature engineering in relational databases. arXiv preprint arXiv:1706.00327 (2017)
8. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
9. Nargesian, F., Samulowitz, H., Khurana, U., Khalil, E.B., Turaga, D.: Learning feature engineering for classification. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI, vol. 17, pp. 2529–2535 (2017)
10. Sutton, R.S., Barto, A.G., et al.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
11. Töscher, A., Jahrer, M., Bell, R.M.: The BigChaos solution to the Netflix grand prize. Netflix prize documentation, pp. 1–52 (2009)
12. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double Q-learning. In: AAAI, Phoenix, AZ, vol. 2, p. 5 (2016)
13. Wang, L., Luo, G., Yi, K., Cormode, G.: Quantiles over data streams: an experimental study. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 737–748. ACM (2013)
14. Watkins, C.J., Dayan, P.: Q-learning. Mach. Learn. **8**(3–4), 279–292 (1992)
15. Zhang, J., Fogelman-Soulié, F., Largeron, C.: Towards automatic complex feature engineering. In: Hacid, H., Cellary, W., Wang, H., Paik, H.-Y., Zhou, R. (eds.) WISE 2018. LNCS, vol. 11234, pp. 312–322. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02925-8_22