Evolutionary Automated Feature Engineering

Anonymous

No Institute Given

Abstract. Effective feature engineering serves as a prerequisite for many machine learning tasks. Feature engineering, which usually uses a series of mathematical functions to transform the features, aims to find valuable new features that can reflect the insight aspect of data. Traditional feature engineering is a labor-intensive and time-consuming task, which depends on expert domain knowledge and requires iterative manner with trial and error. In recent years, many automated feature engineering (AutoFE) methods have been proposed. These methods automatically transform the original features to a set of new features to improve the performance of the machine learning model. However, existing methods either suffer from computational bottleneck, or do not support high-order transformations and various feature types. In this paper, we propose EAAFE, to the best of our knowledge, the first evolutionary algorithm-based automated feature engineering method. We first formalize the AutoFE problem as a search problem of the optimal feature transformation sequence. Then, we leverage roulette wheel selection, subsequence-exchange-based DNA crossover, and ϵ-greedy-based DNA mutation to achieve evolution. Despite its simplicity, EAAFE is flexible and effective, which can not only support feature transformation for both numerical and categorical features, but also support high-order feature transformations. Extensive experimental results on public datasets demonstrate that EAAFE outperforms the existing AutoFE methods in both effectiveness and efficiency.

Keywords: Automated Feature Engineering · Evolutionary Algorithm

1 Introduction

In many practical machine learning (ML) tasks, the quality of features often directly determines the upper bound of ML algorithms. Feature engineering, which aims to extract valuable features from raw data, serves as a prerequisite for ML tasks. However, feature engineering is a labor-intensive task, which depends on extensive domain knowledge and requires an iterative manner with trial-and -error. Although several ML methods such as deep neural networks can automatically generate high-level representations from raw data, but these high-level features are generally uninterpretable [1]. Because of this, feature engineering is viewed as the most creative and time-consuming stage of ML tasks.

Recently, automated feature engineering (AutoFE) that generates useful features without any human intervention has attracted more and more attention.

The core of AutoFE is to search for the best-suited transformations (e.g., unary and binary arithmetic operators) for raw features. The *expansion-reduction* methods [10, 15] apply all possible transformations to each feature and select the features based on the improvement in model performance. Due to the composition of feature transformation, such a brute-force method leads to exponential growth in the space of constructed features. TransGraph [13] leverages a hierarchical graph structure to represent the feature transformation space. However, the feature explosion problem also exists, especially in the bottom layer of the transformation graph because each transformation will act on all features. To eliminate the feature explosion problem, LFE [20] employ meta-learning to recommend the promising transformation for each feature. However, it does not support the composition of transformations (i.e., high-order transformations). Recently, NFS [2] utilizes reinforcement learning to search a transformation sequence for each feature and achieves the state-of-the-art performance. Nevertheless, the computation efficiency of NFS is low because it needs to train an recurrent neural network (RNN) controller for each feature. Moreover, NFS cannot support feature transformations for categorical features.

Inspired by the success of evolutionary neural architecture search (NAS) [23, 22], in this paper, we propose EAAFE [1], to the best of our knowledge, the first evolutionary algorithm-based automated feature engineering method. We first formalize the feature engineering problem as a search problem for feature transformation sequences. Then, we leverage the *roulette wheel selection* [6] evolutionary algorithm to find the optimal feature transformation sequence for each raw feature. Specifically, we concatenate the feature transformation sequences of all features together and view the concatenated sequence as an individual in the population. The DNA of each individual consists of the encoding of all feature transformation operations. The encoding space is constrained by the feature type (i.e., categorical or numerical). The fitness of individual is determined by the performance of the underlying machine learning model that is trained with the raw and constructed features.

During the evolutionary process, we further propose a subsequence-exchange-based DNA crossover method. For each selected DNA pair, we randomly choose two crossover points in the transformation sequence of each feature, and then exchange the subsequences between these crossover points. After DNA crossover, we propose a ϵ-greedy-based DNA mutation method. Each transformation operation in the DNA is mutated with a probability ϵ.

Our main contributions are summarized as follows:

- We propose an evolutionary AutoFE framework called EAAFE to automatically search for the optimal transformation sequence for each raw feature. In the evolutionary process, we propose a subsequence-exchange-based DNA crossover method and a ϵ-greedy-based DNA mutation method.
- Despite its simplicity, the proposed evolutionary approach is flexible and effective, which can not only support feature transformation for both numerical and categorical features, but also support high-order feature transformations.

[1] Anonymous code is available at https://github.com/eaafe/EAAFE

– Extensive experimental results on benchmark classification and regression tasks reveal that EAAFE outperforms other existing AutoFE methods in terms of effectiveness and efficiency.

2 Related Work

2.1 Automated Feature Engineering

A basic approach for automated feature engineering is the *expansion-selection* approach. It predefines a series of feature transformations and applies them to raw features for constructing new features, then combines the constructed features and raw features as candidate features. With a feature selection step, only a subset of candidate features is preserved. The *expansion-selection* approach suffers from combinatorial explosion problem. The representative method based on *expansion-selection* is Deep Feature Synthesis [10], which relies on recursively enumerating all possible new features, then performing feature selection. One Button Machine [15] adopts similar approach to relational databases.

Another basic approach is the *performance-guided* method. After new features are constructed, their performances on the machine learning model are evaluated and only promising features will be preserved. Such an approach needs to explicitly expand the feature space and thus suffers from extensive evaluation overhead. FEADIS [5] adds constructed features greedily relying on a combination of random feature generation and feature selection. ExploreKit [11] employs a feature selection method based on machine learning to rank the newly constructed features and greedily evaluate the most promising ones.

To explore the search space more efficiently, *transformation-graph* based approaches are proposed, which describe feature transformations as a hierarchical graph structure. Each node represents a feature set and each edge represents a transformation. Automated feature engineering can be achieved by iteratively constructing the graph and designing efficient algorithms to explore it. Cognito [14] recommends a series of transformations based on a greedy heuristic tree search. The method in [13] utilizes Q-learning to explore the feature transformation tree. LAFEM [19] formalizes the feature engineering as a heterogeneous transformation graph and adopts deep reinforcement learning to achieve automated feature engineering. Although these methods construct more efficient search space, they suffer from the feature explosion problem and low search efficiency.

Recently, LFE [20] proposes a meta-learning based method, which learns from past feature engineering experiences to recommend promising transformations. LFE recommends only one transformation operation for each raw feature and the feature explosion problem no longer exists. However, LEF does not support the composition of transformations (i.e., high-order transformations), which may be valuable in practice. NFS [2] utilizes an RNN controller trained by reinforcement learning to transform each raw feature through a series of transformation operations. NFS can capture potentially valuable high-order transformations and

achieve the-state-of-art performance. However, it suffers from slow convergence and low search efficiency, because it needs to train an RNN controller for each raw feature.

Similar to NFS, the proposed approach EAAFE, models AutoFE as finding the optimal feature transformation sequence for each raw feature. Different from NFS, EAAFE employs a simple and flexible evolutionary algorithm and can achieve better performance in terms of effectiveness and efficiency.

2.2 Evolutionary algorithm

Evolutionary algorithm is a kind of random search algorithm that simulates natural selection and evolution processes of creatures. It streamlines the complex evolutionary process and abstracts a set of mathematical models. Evolutionary algorithm uses an encoding method to represent complex phenomena, and implements heuristic search for complex search spaces according to simplified genetic processes [7]. Traditional evolutionary algorithms first construct a population via randomly initialized individuals, then evaluate each individual to get the fitness, and finally perform mutations to generate a new population [4]. As a result, evolutionary algorithm can find the global optimal solution with a high probability [7]. Recently, evolutionary algorithm has been successfully adopted to the machine learning pipeline construction [21] and neural architecture search [23, 22].

In this paper, we consider encoding constraints during evolution process [3, 17], leading to different search spaces for different feature types. Moreover, we propose novel DNA crossover and mutation methods in the context of automated feature engineering.

3 The Proposed Approach

3.1 Problem Formulation

Given a dataset $\mathcal{D} = \langle F, \mathbf{y} \rangle$ containing raw features $F = \{f_1, f_2, \cdots, f_n\}$ and a target vector $\mathbf{y}$, let $V_M^E(F, \mathbf{y})$ denote the performance of a machine learning model M trained with F and measured by an evaluation metric E. The feature transformation operations mainly contain unary (e.g., *log* and *square*) and binary operations (e.g., *sum* and *multiply*). The transformation performed on three or more features can be constructed by nesting multiple binary transformations.

The feature set F can be further divided into a categorical feature set F_c and a numerical feature set F_n. Different feature types correspond to different transformation operations. Consequently, we define five types of feature transformation operations, i.e., $\mathcal{T} = \{\mathcal{T}_c, \mathcal{T}_{cc}, \mathcal{T}_n, \mathcal{T}_{nn}, \mathcal{T}_{cn}\}$, where $\mathcal{T}_n$ and $\mathcal{T}_{nn}$ denote sets of unary and binary transformations performed on numerical features, respectively. $\mathcal{T}_c$ and $\mathcal{T}_{cc}$ are similar to $\mathcal{T}_n$ and $\mathcal{T}_{nn}$, but are performed on categorical features. $\mathcal{T}_{cn}$ denotes the aggregate transformation operations between categorical and numerical features.

As shown in Figure 1, each raw feature f_i corresponds to a transformation sequence $t_i = \langle t_{i1}, \cdots, t_{ij}, \cdots t_{il} \rangle$, where $t_{i\cdot}$ denotes the j-th transformation function in the sequence and l denotes the length of the transformation sequence. Let $T = \{t_1, t_2, \cdots, t_n\}$ denotes the feature transformation sequences of all features, which transforms the raw feature set F to $\hat{F}$. We take the union set of F and $\hat{F}$, i.e., $F \cup \hat{F}$, as the newly constructed feature set. As a result, the goal of automated feature engineering is to search for the optimal T which can maximize the performance of the given ML model. Formally as

$$\arg\max_{T} V_M^E(F \cup \hat{F}, \mathbf{y}), \quad \hat{F} = T(F)$$

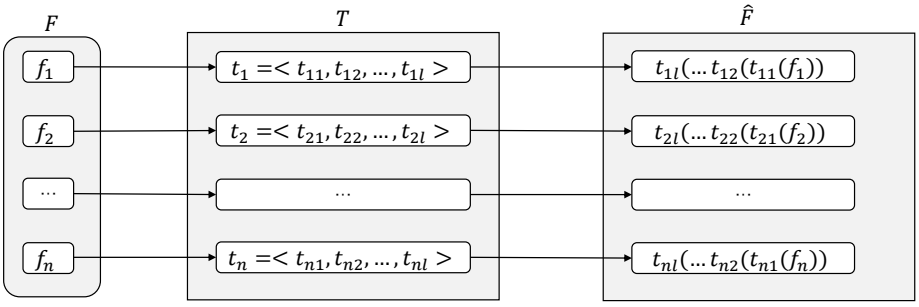

Fig. 1: Feature transformation sequences.

In practice, the lengths of transformation sequences for different raw features may be required to be different. To support variable sequence lengths, we define a terminate operation, which indicates the early-stopping of the feature transformation sequence. The terminate operation also enables our approach to automatically find the proper length of the transformation sequence.

3.2 Constrained DNA Encoding

During the evolutionary process, we take the concatenated transformation sequence T as an individual in the population, the transformation operation t_{ij} as a chromosome, and elements in t_{ij} as genes. The DNA of each individual can be viewed as the encoding of all operations in T. The encoding space is represented by non-overlapping integer intervals. Since a binary transformation operator takes two features as input, we convert it to a set of unary transformations. In addition to the operation itself, each unary transformation has a feature index attached to it. For example, we can convert the binary transformation *sum* to a set of unary transformations $\{sum(f_1), \cdots, sum(f_i), \cdots, sum(f_n)\}$.

Moreover, different feature types (i.e., categorical or numerical) correspond to different encoding spaces. Let $\mathcal{S}_n$ denote the encoding space of numerical

features, which is determined by the feature transformation sets $\mathcal{T}_n$ and $\mathcal{T}_{nn}$. Correspondingly,

$$\mathcal{S}_n = [0, |\mathcal{T}_n|] \bigcup [|\mathcal{T}_n|, |\mathcal{T}_n| + |\mathcal{T}_{nn}| * |F_n|] \tag{1}$$

where $|\mathcal{T}_n|$ and $|F_n|$ denote the number of transformations and the number of numerical features, respectively. $\mathcal{S}_n$ is the combination of the encoding spaces for the unary and binary transformations. Similarly, let $\mathcal{S}_c$ denote the encoding space of categorical features, which is determined by the feature transformation sets $\mathcal{T}_c$, $\mathcal{T}_{cc}$, and $\mathcal{T}_{cn}$. Thus, $\mathcal{S}_c$ is expressed as:

$$\mathcal{S}_c = [-|\mathcal{T}_c|, 0] \bigcup [-|\mathcal{T}_c| - |\mathcal{T}_{cc}| * |F_c| - |\mathcal{T}_{cn}| * |F_n|, -|\mathcal{T}_c|] \tag{2}$$

In Equation 1 and Equation 2, the number 0 indicates the encoding of the terminate operation. And the transformations for numerical features and categorical features are encoded by positive integers and negative integers, respectively. Moreover, $\mathcal{S}_n$ and $\mathcal{S}_c$ are not overlapped.

3.3 Evolutionary Search Algorithm

In this subsection, we introduce the evolutionary search algorithm for automated feature engineering in EAAFE.

In the beginning, we randomly initialize the population according to the encoding space constrained by the feature type. Suppose that the initial population contains p individuals. Then, we perform evolution iteratively. Each evolution is composed of four steps: calculation of DAN fitness, roulette wheel selection, DNA crossover, and DNA mutation.

Calculation of DNA Fitness. The fitness reflects the adaptability of an individual. For an individual T, the fitness of is determined by the performance of given ML model $V_M^E(F \cup \hat{F}, \mathbf{y})$, where $\hat{F}$ is constructed by transforming the raw feature set F with T. The evaluation metric E can be F1-score or mean squared error. And the underlying ML model M is set to random forest by default in our experiments. We also evaluate the generalization of EAAFE through different underlying models in Section 4.5. Moreover, since the transformation processes of different raw features are independent with each other, we can transform different features in parallel, which can alleviate the combinatorial explosion problem and further improve the efficiency.

Roulette Wheel Selection. [6] To balance the exploration (i.e., population diversity) and exploitation (e.g., higher fitness), we leverage the roulette wheel selection evolutionary algorithm to select N individuals from current population according to the fitness. The core of roulette wheel selection is that the probability of an individual to be selected is proportional to its fitness.

Specifically, let $p(x_i)$ indicate the probability of the individual x_i being selected into the next-generation population. There is

$$p(x_i) = \frac{f(x_i)}{\sum_{j=1}^{m} f(x_j)} \tag{3}$$

where $f(x_i)$ denotes the fitness of the individual x_i and m is the size of current population. Then, we calculate the cumulative probability of each individual, denoted by $q(x_i)$.

$$q(x_i) = \sum_{j=1}^{i} p(x_i) \tag{4}$$

Next, we randomly generate an array A of size m, where each element lies in (0,1). The array A is sorted in ascending order. If $q(x_i) > A[i]$, then x_i will be selected. Otherwise, $q(x_{i+1})$ will be compared with $A[i+1]$ until one individual is selected. Then, we randomly generate an array once again, and the above process is repeated until N individuals are selected.

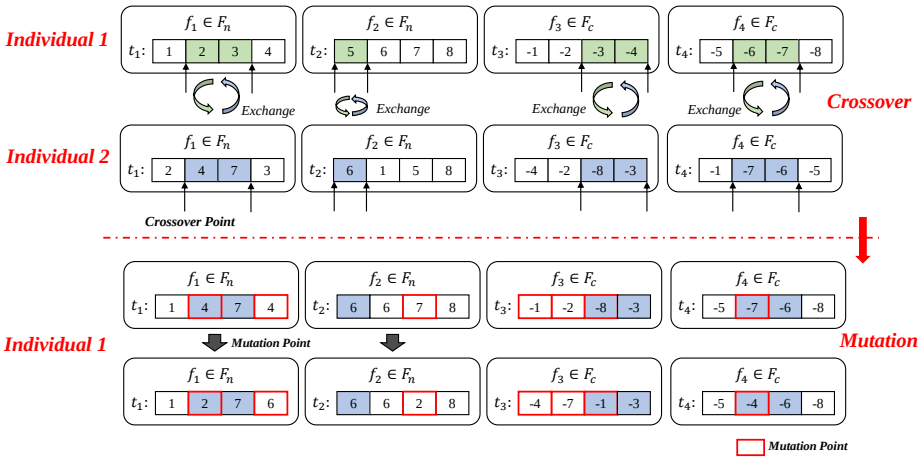

Fig. 2: Crossover and mutation in the evolutionary process.

Subsequence-exchange-based DNA Crossover. As shown in Figure 2, the DNA of an individual is composed of n chromosomes. Each chromosome corresponds to a transformation sequence of a raw feature. In the DNA crossover process, we first randomly select two crossover points in each chromosome. Then, we exchange the subsequences between the crossover points.

ϵ-greedy-based DNA Mutation. The crossover operation can retain outstanding genes. However, to avoid falling into the local optima, we need to further perform DNA mutation to increase the diversity of population. Every gene (i.e., transformation operation) in the DNA can mutate with a certain probability ϵ. Once a mutation occurs, the selected gene will be replaced by another transformation randomly selected from the same encoding space. We continuously perform crossover and mutation until a new population is generated.

The above-mentioned evolution process is repeated until the predefined number of iterations or the time limit is reached.

4 Experiments

In this section, we conduct extensive experiments to evaluate the following aspects of the proposed EAAFE: (1) the effectiveness of EAAFE; (2) the efficiency of EAAFE; (3) the effectiveness of the high-order transformation; (4) the generalization performance of EAAFE.

4.1 Experiment Setup

As the state-of-the-art method NFS [2], we use 23 public datasets from OpenML [2], UCI repository [3], and Kaggle [4]. There are 13 classification (C) datasets and 10 regression (R) datasets that have various numbers of features (5 to 57) and instances (500 to 30000). Excepting that different ML algorithms are used in Section 4.5, we utilize *Random Forest* as the underlying ML model in all other experiments. For evaluation metrics, we use *1-(relative absolute error)* [24] for regression tasks, and *F1-score* for classification tasks. The 5-fold cross validation using random stratified sampling is employed and the average result of 30 runs is reported.

In the evolutionary process, we set the total number of evolution iterations to 5000, the population size to 48, and the mutation probability to 0.1. The length of the transformation sequence for each raw feature is set to 5 for all experiments except the one to verify the effectiveness of high-order transformation. Without loss of generality, we utilize 12 feature transformation operations which cover five types of transformations defined as follows: 1) unary transformations for numerical features: *sqrt*, *minmaxsacler*, *log*, *reciprocal*; 2) binary transformations for numerical features: *add*, *sub*, *mul*, *div*; 3) unary transformation for categorical features: *count*; 4) binary transformations for category features: *cat2cat_count* and *cat2cat_unique*; 5) binary transformation for categorical and numerical features: *cat2num_mean*. Other feature transformation operations can be flexibly integrated into EAAFE.

4.2 Effectiveness of EAAFE

In order to verify the effectiveness of EAAFE, we compared it with following methods:

- **Base**, which directly uses raw datasets for evaluation.
- **Random**, which generates feature transformation sequences randomly according to the proposed encoding space.
- **DFS** [10]: a well-known *expansion-reduction* method.
- **AutoFeat** [8], a popular Python library for automated feature engineering and selection.

[2] https://www.openml.org/
[3] https://archive.ics.uci.edu/
[4] https://www.kaggle.com/

Table 1: Comparison between EAAFE and other existing automated feature engineering methods†. The best results are marked by bold. F1-Score is reported for classification tasks and 1-(Relative Absolute Error) is reported for regression tasks. "Inst." indicates the number of instances and "Feat." indicates the number of features.

Dataset	Source	C\R	Inst.\Feat.	Base	Random	DFS†	AutoFeat†	NFS†	EAAFE
Bikeshare DC	Kaggle	R	10886\11	0.816	0.844	0.821	0.85	0.975	**0.981**
Housing Boston	UCIrvine	R	506\13	0.434	0.445	0.341	0.469	0.501	**0.701**
Airfoil	UCIrvine	R	1503\5	0.496	0.573	0.435	0.596	0.616	**0.81**
OpenML 586	OpenML	R	1000\25	0.662	0.651	0.65	0.728	0.74	**0.792**
OpenML 589	OpenML	R	1000\25	0.644	0.642	0.636	0.686	0.714	**0.757**
OpenML 607	OpenML	R	1000\50	0.634	0.629	0.639	0.67	0.687	**0.734**
OpenML 616	OpenML	R	500\50	0.573	0.571	0.572	0.603	0.592	**0.726**
OpenML 618	OpenML	R	1000\50	0.627	0.617	0.634	0.632	0.64	**0.754**
OpenML 620	OpenML	R	1000\25	0.633	0.618	0.626	0.687	0.675	**0.739**
OpenML 637	OpenML	R	1000\25	0.514	0.527	0.519	0.576	0.569	**0.619**
PimaIndian	UCIrvine	C	768\8	0.756	0.757	0.75	0.763	0.784	**0.802**
SpectF	UCIrvine	C	267\44	0.775	0.828	0.791	0.816	0.85	**0.94**
German Credit	UCIrvine	C	1001\24	0.741	0.755	0.749	0.76	0.782	**0.803**
Ionosphere	UCIrvine	C	351\34	0.923	0.934	0.918	0.912	0.952	**0.986**
Credit Default	UCIrvine	C	30000\25	0.804	0.806	0.806	0.806	0.805	**0.815**
Messidorfeatures	UCIrvine	C	1150\19	0.658	0.688	0.672	0.736	0.746	**0.797**
Wine Quality Red	UCIrvine	C	999\12	0.532	0.564	0.548	0.524	0.584	**0.611**
Wine Quality White	UCIrvine	C	4900\12	0.494	0.493	0.488	0.502	0.515	**0.524**
SpamBase	UCIrvine	C	4601\57	0.91	0.924	0.91	0.924	0.93	**0.984**
Credit-a	UCIrvine	C	690\6	0.838	0.845	0.819	0.839	0.865	**0.883**
Fertility	UCIrvine	C	100\9	0.853	0.83	0.75	0.79	0.87	**0.91**
Hepatitis	UCIrvine	C	155\6	0.786	0.83	0.826	0.768	0.877	**0.923**
Megawatt1	UCIrvine	C	253\37	0.889	0.897	0.877	0.889	0.913	**0.953**

– **NFS** [2], which utilizes the RNN-based controller trained by reinforcement learning to find transformation operations. NFS is the state-of-the-art AutoFE method that achieves better performance than other existing approaches (e.g., TransGraph [13]).

The results of baselines are obtained using the open-sourced code. The experimental settings of these methods, such as the order of transformed features and the evaluation metrics are same as EAAFE. Table 1 shows the comparison results between EAAFE and other baseline methods. From Table 1, we can see that EAAFE achieves the best performance in all datasets. AutoFeat, NFS and EAAFE achieve average improvements of 4.0%, 8.0% and 17.9% over Base, and average improvements of 2.1%, 5.9% and 15.6% over Random, respectively. The experimental results demonstrate the importance of feature engineering, and the effectiveness of automated feature engineering. Compared with other AutoFE methods, EAAFE is the most effective and achieves an average improvement of 10% and 28.2% over Base on regression tasks and classification tasks, respectively. The classification tasks benefit more from the feature engineering. Moreover, EAAFE can achieve highly competitive performance even

Table 2: Comparison between EAAFE and NFS. The number of evaluations is limited to 5000. The best results are marked by bold.

Dataset	Base	NFS	EAAFE	Dataset	Base	NFS	EAAFE
Bikeshare DC	0.816	0.969	**0.98**	German Credit	0.741	0.767	**0.803**
Housing Boston	0.434	0.478	**0.525**	Ionosphere	0.923	0.949	**0.971**
Airfoil	0.496	0.606	**0.673**	Credit Default	0.804	0.808	**0.815**
OpenML 586	0.662	0.662	**0.732**	Messidorfeatures	0.658	0.727	**0.791**
OpenML 589	0.644	0.648	**0.672**	Wine Quality Red	0.532	0.569	**0.604**
OpenML 607	0.634	0.638	**0.723**	Wine Quality White	0.494	0.508	**0.522**
OpenML 616	0.573	0.581	**0.714**	SpamBase	0.91	0.927	**0.98**
OpenML 618	0.627	0.627	**0.748**	Credit-a	0.838	0.861	**0.883**
OpenML 620	0.633	0.633	**0.714**	Fertility	0.853	0.87	**0.91**
OpenML 637	0.514	0.542	**0.601**	Hepatitis	0.786	0.858	**0.923**
PimaIndian	0.756	0.768	**0.802**	Megawatt1	0.889	0.905	**0.945**
SpectF	0.775	0.828	**0.925**				

on relatively larger datasets such as *Credit Default* (with size 30000×25) and *Bikeshare DC* (with size 10886×11).

4.3 Efficiency of EAAFE

The main bottleneck of automated feature engineering methods is the evaluation process which needs to train the ML model from sketch. Moreover, in practice, the computational resource is always limited. In this experiment, in order to verify the efficiency of EAAFE, we limit the total number of evaluations to 5000, and compare EAAFE with NFS .

From Table 2, we can see that EAAFE consistently outperforms NFS and achieves an average improvement of 13% over NFS in the case of limited evaluation budget. Moreover, on most datasets, EAAFE achieves comparable results, but NFS obtains even inferior results compare to their best results in Table 1, which can also demonstrate the efficiency and effectiveness of EAAFE.

4.4 Effectiveness of the High-Order Transformation

In order to verify the effectiveness of the high-order transformation, we change the length of transformation sequence from 1 to 6, and report the relative improvements of EAAFE over Base on *Housing Boston*, *Airfoil*, *OpenML 586* and *OpenML 616*. We omit other datasets, since the experimental results are similar.

Figure 3 illustrates that the performance of EAAFE increases stably with longer sequence length, but when the sequence length is higher than 5, the performance degrades. Thus, overly complex features will not necessarily bring performance improvement. Moreover, the search space also exponentially increases with the increasing of transformation sequence length, leading to low search efficiency. Therefore, considering both performance and efficiency, we choose 5 as the length of the feature transformation sequence in other experiments.

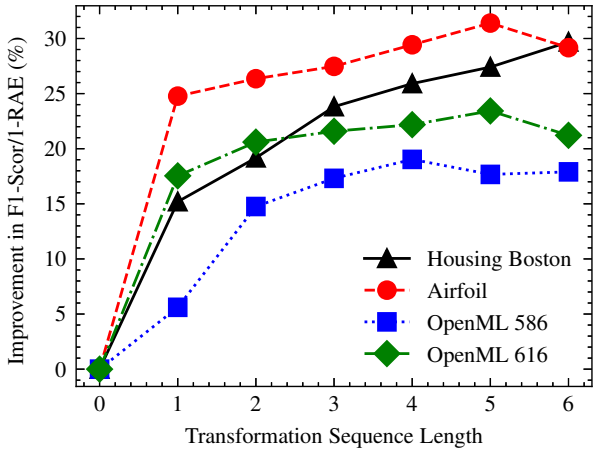

Fig. 3: Improvements of EAAFE over Base with different lengths of feature transformation sequences.

4.5 Generalization Performance of EAAFE

In the previous experiments, we only use *Random Forest* as the underlying ML model to evaluate EAAFE. In order to verify whether EAAFE can generalize to other frequently-used machine learning models, we further utilize *Lasso Regression* [26], *Linear SVR* [25], and *LightGBM* [12] for regression tasks, and *Logistic Regression* [9], *Linear SVC* [16], and *LightGBM* [12] for classification tasks. Due to lack of space, we omit detailed results and only report the average improvements of EAAFE on all datasets over these models.

Table 3: Average improvements of EAAFE over different ML models.

	Algorithm	Average Improvement
Classification	Logistic Regression	9.65%
	LinearSVC	26%
	LgbmClassifier	10%
Regression	Lasso Regression	14.7%
	LinearSVR	22.1%
	LgbmRegressor	13.8%

Table 3 shows the average improvements of EAAFE over different ML models, which can demonstrate the generalization performance of EAAFE. In conclusion, EAAFE can adapt to various tasks and machine learning models, which is an essential advantage for real-world applications.

5 Conclusion and Future Work

In this paper, we propose an effective and efficient automated feature engineering framework, named EAAFE. We first formalize the AutoFE problem as a search problem of the optimal feature transformation sequence. Then we construct an expressive search space by encoding different types of feature transformations to different sub-spaces, and propose an effective evolutionary algorithm to explore the constrained search space. Moreover, we design novel roulette wheel selection, subsequence-exchange-based crossover, and ϵ-greedy-based mutation strategies for the evolution. Despite its simplicity, EAAFE is flexible and effective, which can support not only both numerical and categorical features, but also high-order feature transformations. Extensive experimental results on public datasets demonstrate that EAAFE consistently outperforms other state-of-the-art AutoFE methods in both effectiveness and efficiency.

In the future, we plan to extend EAAFE to support more feature types and transformation operations. In addition, we intend to employ Ray [18] to parallelize EAAFE for higher efficiency.

References

1. Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence **35**(8), 1798–1828 (2013)
2. Chen, X., Qiao, B., Zhang, W., Wu, W., Zhang, X.: Neural feature search: A neural architecture for automated feature engineering. In: 2019 IEEE International Conference on Data Mining (ICDM) (2019)
3. Coello, C.A.C.: Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Computer methods in applied mechanics and engineering **191**(11-12), 1245–1287 (2002)
4. Deb, K., Anand, A., Joshi, D.: A computationally efficient evolutionary algorithm for real-parameter optimization. Evolutionary computation **10**(4), 371–395 (2002)
5. Dor, O., Reich, Y.: Strengthening learning algorithms by feature discovery. Information Sciences **189**, 176–190 (2012)
6. Goldberg, D.E.: Genetic algorithms. Pearson Education India (2006)
7. Holland, J.H., et al.: Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press (1992)
8. Horn, F., Pack, R., Rieger, M.: The autofeat python library for automated feature engineering and selection. arXiv preprint arXiv:1901.07329 (2019)
9. Hosmer Jr, D.W., Lemeshow, S., Sturdivant, R.X.: Applied logistic regression, vol. 398. John Wiley & Sons (2013)
10. Kanter, J.M., Veeramachaneni, K.: Deep feature synthesis: Towards automating data science endeavors pp. 1–10 (2015)
11. Katz, G., Shin, E.C.R., Song, D.: Explorekit: Automatic feature generation and selection pp. 979–984 (2016)
12. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: Lightgbm: A highly efficient gradient boosting decision tree. In: Advances in neural information processing systems. pp. 3146–3154 (2017)

13. Khurana, U., Samulowitz, H., Turaga, D.S.: Feature engineering for predictive modeling using reinforcement learning pp. 3407–3414 (2018)
14. Khurana, U., Turaga, D.S., Samulowitz, H., Parthasrathy, S.: Cognito: Automated feature engineering for supervised learning pp. 1304–1307 (2016)
15. Lam, H.T., Thiebaut, J., Sinn, M., Chen, B., Mai, T., Alkan, O.: One button machine for automating feature engineering in relational databases. arXiv: Databases (2017)
16. Luts, J., Ojeda, F., De Plas, R.V., De Moor, B., Van Huffel, S., Suykens, J.A.K.: A tutorial on support vector machine-based methods for classification problems in chemometrics. Analytica Chimica Acta **665**(2), 129–145 (2010)
17. Michalewicz, Z., Schoenauer, M.: Evolutionary algorithms for constrained parameter optimization problems. Evolutionary computation **4**(1), 1–32 (1996)
18. Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M.I., Stoica, I.: Ray: A distributed framework for emerging ai applications. pp. 561–577 (2018)
19. Mozer, S., C, M., Hasselmo, M.: Reinforcement learning: An introduction. IEEE Transactions on Neural Networks **16**(1), 285–286 (2005)
20. Nargesian, F., Samulowitz, H., Khurana, U., Khalil, E.B., Turaga, D.S.: Learning feature engineering for classification pp. 2529–2535 (2017)
21. Olson, R.S., Moore, J.H.: Tpot: A tree-based pipeline optimization tool for automating machine learning. pp. 66–74 (2016)
22. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: Proceedings of the AAAI onference on Artificial Intelligence. vol. 33, pp. 4780–4789 (2019)
23. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., Kurakin, A.: Large-scale evolution of image classifiers. In: Proceedings of the 34th International Conference on Machine Learning. pp. 2902–2911 (2017)
24. Shcherbakov, M.V., Brebels, A., Shcherbakova, N.L., Tyukov, A.P., Kamaev, V.A.: A survey of forecast error measures. World Applied ences Journal **24**(24), 171–176 (2013)
25. Smola, A.J., Scholkopf, B.: A tutorial on support vector regression. Statistics and Computing **14**(3), 199–222 (2004)
26. Tibshirani, R.: Regression shrinkage and selection via the lasso. Journal of the Royal Statal Society, Series B **58**(1) (1996)