SAFE: Scalable Automatic Feature Engineering Framework for Industrial Tasks

Qitao Shi, Ya-Lin Zhang†, Longfei Li, Xinxing Yang, Meng Li, Jun Zhou
Ant Financial Services Group
{qitao.sqt, lyn.zyl, longyao.llf, xinxing.yangxx, lm168260, jun.zhoujun}@antfin.com

***Abstract*—Machine learning techniques have been widely applied in Internet companies for various tasks, acting as an essential driving force, and feature engineering has been generally recognized as a crucial tache when constructing machine learning systems. Recently, a growing effort has been made to the development of automatic feature engineering methods, so that the substantial and tedious manual effort can be liberated. However, for industrial tasks, the efficiency and scalability of these methods are still far from satisfactory. In this paper, we proposed a staged method named SAFE (Scalable Automatic Feature Engineering), which can provide excellent efficiency and scalability, along with requisite interpretability and promising performance. Extensive experiments are conducted and the results show that the proposed method can provide prominent efficiency and competitive effectiveness when comparing with other methods. What's more, the adequate scalability of the proposed method ensures it to be deployed in large scale industrial tasks.**

***Index Terms*—feature engineering, automatic machine learning**

I. Introduction

Nowadays, machine learning (ML) techniques have been widely explored and applied in almost all Internet companies, and serving as essential parts in diversified fields, such as recommendation system [1]–[3], fraud detection [4]–[6], advertising [7]–[9], and face recognition [10], [11], etc. With the help of these techniques, excellent performance and significant improvement have been obtained.

Generally speaking, to build a machine learning system, a professional and complex ML pipeline is always needed, which usually includes data preparation, feature engineering, model generation, and model evaluation, etc. It is widely agreed that the performance of machine learning methods depends to a large extent on the quality of the features, and generating a good feature set becomes a crucial step to chase high performance [12]. Therefore, most machine learning engineers take a large effort to obtain useful features when building a machine learning system.

However, it is frustrating that feature engineering is often the most indispensable part of human intervention in ML pipelines since human intuition and experience are gravely required, thus, it becomes tedious, task-specific and challenging, and hence, time-consuming. On the other hand, with the growing need for ML techniques in industrial tasks, it becomes impracticable to manually perform feature engineering in all of these tasks. This promotes the birth of automatic feature engineering, which is an important topic of automatic machine learning (AutoML) [13]–[16]. The development of automatic feature engineering can not only liberate machine learning engineers from the substantial and tedious process, but also power machine learning techniques to be applied in more and more applications.

For a regular supervised learning task, problem can be formulated as using training examples to find a function $\mathcal{F} : X \to Y$, which is defined as returning the y value that obtains the highest score: $\mathcal{F}(\boldsymbol{x}) = \arg\max_y \mathcal{S}(\boldsymbol{x}, y)$, where X is the input space, Y is the output space and $\mathcal{S} : X \times Y \to R$ is a scoring function. The goal of automatic feature engineering is to learn a feature representation $\Psi : X \to Z$, to construct a new feature representation $\boldsymbol{z}$ from the original feature $\boldsymbol{x}$, with which the performance of subsequent machine learning tools can be further improved as much as possible.

Several studies have been conducted on this topic. To name a few, some methods use reinforcement learning based strategy to perform automatic feature engineering [17]–[19]. These methods require many rounds of attempts and it is necessary to generate a new feature set and evaluate it in each round, making them infeasible in industrial tasks. Transfer learning or meta-learning based strategies are also proposed for automatic feature engineering [20], [21]. However, a large number of experiments on various datasets are needed in advance to train these methods, and it is intractable to introduce new operators or increase the number of parent features. Some methods follow the generation-selection procedure [22]–[24] to do automated feature engineering. However, these methods always perform as generating all legal features in the feature generation stage and then selecting a subset features from them, thus the time and space complexity is extremely high, making it unapplicable for tasks with large data size or high feature dimension.

In industrial tasks, the size of real business data is always very huge, which introduces extremely high requirements for space and time complexity. At the same time, due to the rapid change of business, there are also high requirements for the flexibility and scalability of the algorithms. Besides, there are more requirements that need to be addressed [25]:

- Strong applicability: A tool that is highly adaptable means that it is user-friendly and easy to use. The performance of an automatic feature engineering algorithm should not depend on a large number of hyper-parameters or one

†: corresponding author.

of its hyper-parameter configurations can be applied to different data sets.
- Distributed computing: the number of samples and features in real-world business tasks are pretty large, which makes distributed computing necessary. Most parts of the automatic feature engineering algorithm should be able to be calculated in parallel.
- Real-time inference: real-time inference is involved in many real-world businesses. In such cases, once an instance is inputted, the feature should be produced instantly and the prediction can be performed subsequently.

In this paper, we approach the problem from the typical two-stage perspective and propose a method named SAFE (**S**calable **A**utomatic **F**eature **E**ngineering) to perform efficient automatic feature engineering, which includes feature generation stage and feature selection stage. We guarantee computational efficiency, scalability and the requirements mentioned above. The major contributions of this paper are summarized as follows:

- In the feature generation stage, different from the previous methods which focuses on what operator to use or how to generate all legal features, we focus on mining the original feature pairs that generate more effective new features with higher probability, to improve the efficiency of the process.
- In the feature selection stage, we propose a pipeline of feature selection, with the consideration of the power of a single feature, the redundancy of feature pairs, and the feature importance evaluated by the typical tree model. It is suitable for multiple different business data sets and various machine learning algorithms.
- We have experimentally proved the advantages of our algorithm on a large set of data sets and multiple classifiers. Compared with the original feature space, the prediction accuracy is improved by 6.50% on average.

The rest of this paper is organized as follows: Section II reviews the related work; Section III explains the problem setting; Section IV details the proposed method and provides some analyses; Section V states the detail of the data set, evaluation method and presents the experimental results to validate our method; Section VI concludes the paper.

II. Related Work

As a nonnegligible issue for automatic machine learning [13]–[16], automatic feature engineering has drawn extensive attention in recent years, and many methods have been proposed from different perspectives to solve this task [17]–[24], [26]–[28]. In this section, we mainly discuss three typical strategies, which include the generation-selection strategy, reinforcement learning based strategy and transfer learning based strategy.

Given a supervised learning data set, a typical method for automatic feature engineering is to follow the generation-selection procedure. The FICUS algorithm [26] initializes by constructing a set of candidate features, and iterates to improve it until the computation budget is exhausted. During each iteration, it performs beam search to construct new features and selects features typically by using heuristic measures based on information gain in a decision tree. TFC [27] also solves this task by an iterative framework. In each iteration, it generates all legal features based on the current feature pool and all available operators, then selects the best features from all candidate features by using information gain, and keeps them as the new feature pool. With this framework, higher-order feature combinations can be obtained as the iteration progresses. However, the exhaustive search in each iteration leads to a combinatorial explosion of feature space, making this approach non-scalable. To avoid exhaustive search, learning based methods, such as the FCTree algorithm [28], have been proposed. FCTree trains a decision tree and performs feature generation by applying several sequential transformations to the original feature, and select features according to information gain on each node of the decision tree. Once a tree is built, features chosen at internal decision nodes are used to obtain the constructed features. [24] is a regression-based algorithm, which learns the representation by mining pairwise feature associations, identifying the linear or non-linear relationship between each pair, applying regression and selecting those relationships that are stable and improve the prediction performance. These algorithms always encounter performance and scalability bottlenecks since the cost of time and resource in the feature generation and selection procedure may be extremely unsatisfactory, if without ingenious design.

Reinforcement learning based strategies are also explored. [17] formalizes feature selection as a reinforcement learning problem and introduces an adaptation of the Monte-Carlo tree search. Here, the problem of choosing a subset of the available features is cast as a single-player game whose states are all possible subsets of features and the actions consist of choosing a feature and adding it to the subset. [18] handles this problem by exploring on a directed acyclic graph which represents the relationship between different transformed versions of the data, and learns an effective strategy to explore available feature engineering choices under a given budget through Q-learning. [19] formalizes this task as an optimization problem over a Heterogeneous Transformation Graph (HTG). It proposes 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 data sets to other unseen data sets.

Transfer learning or meta-learning based strategies are also proposed for automatic feature engineering. [20] employs learning to rank techniques to evaluate the newly constructed features and select the most promising ones. It is extremely time-consuming. For instance, their reported results were obtained after running for several days on moderately sized data sets. In contrast, [21] can generate effective features within seconds on average. It is based on learning the effectiveness of applying a transformation (e.g., arithmetic or aggregate operators) on numerical features, from past feature engineering experiences. However, since the meta-features do not take the

relationship between features into account, it works better only when using unary transformations.

Beyond that, there are also other methods that direct at different settings. For example, [22] automatically constructs features from relational databases via deep feature synthesis. It focuses on the relationships between the various tables in the database to generate new features. A similar approach is adopted by [23]. What's more, many studies try to perform feature engineering simultaneously while training the model, by introducing operations such as feature cross [29] or using techniques like self-attentive neural networks [30]. We need to address that, different from the methods that learn feature representations simultaneously with model training, we are aiming at learning a new representation for each sample based on the original features, which can be used to perform the subsequent machine learning models, and we have no constraint on what model to be used afterward. At the same time, for industrial tasks, interpretability is always required [31]. The generated features in our framework can be easily explained, to satisfy the interpretability requirement in industrial tasks.

To apply automatic feature engineering techniques in real-world applications, especially for industrial tasks, the efficiency and scalability of the aforementioned methods are still far from satisfactory. Methods with excellent efficiency and scalability, along with requisite interpretability and promising performance are in high demand.

III. Problem Statement

Consider a predictive modeling task, which consists of:

- A dataset of input-output pairs. Let $\boldsymbol{x} \in X$ be a record of the input space with M original features $\left\{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(M)}\right\}$. Let $y \in Y$ be the corresponding output label. Training data with N records can be denoted as $\mathcal{D}_{train} = \{X_{train}, Y_{train}\} = \{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_N, y_N)\}$. Similarly, validation data and test data can be denoted as $\mathcal{D}_{valid}$ and $\mathcal{D}_{test}$.
- A machine learning algorithm $\mathcal{A}$ that accepts a training set and a validation set as input and produces a function $\mathcal{F} : X \to Y$, which return the predicted label y given the input x.
- A loss function $\mathcal{L}$ which computes the loss of a learned function $\mathcal{F}$, according to the ground-truth label y.

The goal of automatic feature engineering is to learn a feature generation function $\Psi : X \to Z$ to generate new feature representation $\boldsymbol{z} \in Z$ based on the original features $\boldsymbol{x} \in X$, by using the set of k operations $\mathcal{O} = \{o_1, \cdots, o_k\}$, so that the learning algorithm $\mathcal{A}$ can find a function $\mathcal{F}$ that minimizes the loss function $\mathcal{L}$. i.e., to approximate the true underlying input-output function as much as possible. More formally, we want to obtain the feature generation function, with which the loss $\mathcal{L}$ of the learned predictive function $\mathcal{F}$ can be minimized:

$$\Psi^* = \arg\min_{\Psi} \mathcal{L}(\mathcal{F}(\Psi(X_{test})), Y_{test}) \tag{1}$$

in which the predictive function can be obtained by $\mathcal{F} = \mathcal{A}(\mathcal{D}_{train\_new}, \mathcal{D}_{valid\_new})$, $\mathcal{D}_{train\_new}$ and $\mathcal{D}_{valid\_new}$ are the generated training and validation dataset, i.e., $\mathcal{D}_{train\_new} = \{\Psi(X_{train}), Y_{train}\}$, and $\mathcal{D}_{valid\_new} = \{\Psi(X_{valid}), Y_{valid}\}$.

The operators $\mathcal{O}$, also known as n-ary operators, acts on n original features for feature generation, and it can be divided into unary operators $\mathcal{O}_1$, binary operators $\mathcal{O}_2$, ternary operators $\mathcal{O}_3$, etc. It should be noted that operators which do not satisfy the commutative property will be treated as multiple different operators in our subsequent descriptions and experiments, such as "$\div$".

Unary operators are used for discretizing, normalizing, or mathematically transforming unit features:

- Discretization is the process of transferring continuous features into discrete features. It plays an important role in feature processing. It is robust to anomalous data and can make the trained model more stable. Typical feature discretization methods include ChiMerge, equidistant binning, equal-frequency binning, clustering binning, etc.
- Normalization refers to a process that makes features more normal or regular. Typical feature normalization methods include Min-Max normalization, Z-score, standardization of dispersion, etc.
- Mathematical transformations acting on unit features include log, sigmoid, square, square root, tanh, round, etc.

Binary operators combine two original features to generate a new feature:

- Four basic arithmetic operations: $+$, $-$, $\times$, $\div$.
- Logical operators act on two boolean features, such conjunction ($\wedge$), disjunction ($\vee$), alternative denial ($\uparrow$), joint denial ($\downarrow$), material conditional ($\rightarrow$), converse implication ($\leftarrow$), biconditional ($\leftrightarrow$), exclusive or ($\nleftrightarrow$), etc.
- *GroupByThenMax*, *GroupByThenMin*, *GroupByThenAvg*, *GroupByThenStdev* and *GroupByThenCount*. These operators implement the SQL-based operations with the same name.
- Ridge regression and kernel ridge regression in [24] can also be considered as binary operators.

Ternary operators combine three features to generate a new feature. A common ternary operator is a conditional operator, which is a basic conditional statement in many programming languages. For the conditional expression $a?b : c$, if the value of a is true, the value of b is obtained; otherwise, the value of c is obtained.

There are also many operators that can accept multiple original features as input, such as MAX, MIN, MEAN, etc. We divide them into different categories when they accept a different number of inputs.

It should be pointed out that there are still many operators that apply in specific fields, we call them domain-specific operators, such as lag operators in time series analysis, genetic operators in biology, etc.

Because of the existence of various operators, an applicable automatic feature engineering algorithm framework should not limit operators and new operators should be easily added.

What's more, to ensure the method to be feasible for large scale industrial tasks, the whole automatic feature engineer-

ing framework should be time and space-friendly, and with requisite interpretability and promising performance.

IV. Proposed Method

A. Overview

As shown in Fig. 1 and Algorithm 1, our automatic feature engineering algorithm is an iterative process where each iteration comprises of two phases: feature generation and feature selection. The number of iterations is limited by the calculation time or computation space.

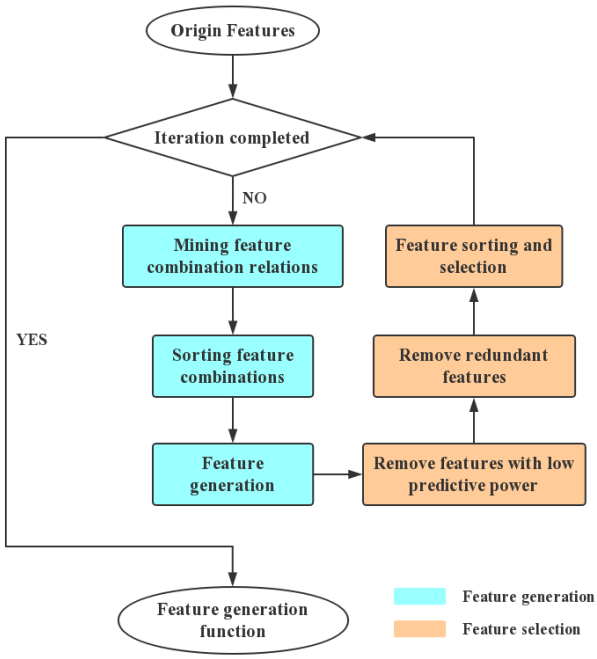


Fig. 1. Flow chart of SAFE

As discussed above, exhaustive searching is ineffective due to the infinite feature space. Even if the numbers of operators and iterations are limited, exhaustive searching can also result in a combinatorial explosion. To avoid this problem, we use a tree based method, i.e., XGBoost [32], to mine the relationships between the current set of base features X^i to narrow down the search space for feature combinations and then sort and filter the feature combinations by information gain ratio. We then apply the predefined operators on the filtered feature combinations with a high information gain ratio and obtain the new feature set $\widetilde{X}^i$. By combining the base features X^i and the generated features $\widetilde{X}^i$, the candidate feature set can be obtained, which is denoted as $\hat{X}^i = X^i \cup \widetilde{X}^i$.

As the number of the current set of features $\hat{X}^i$ is still very large, we use efficient and effective feature ranking and selection methods after that. The basic idea is to find the informative features, remove the redundant ones, and then each feature with be attached a score so that the filter process can be performed if necessary. Concretely, we first use the information value to pick out the features that have a high impact on the label, which are regarded as more informative features. Then, we use the pearson correlation coefficient to remove the redundant features. Finally, we use XGBoost to score the remaining features by the average gain across all splits in which the feature is used. We only choose the features with the highest scores as X^{i+1} for the next iteration, with the consideration of scalability and efficiency.

In the next two subsections, we will explain the feature generation and feature selection process in detail.

B. Feature Generation

The goal of this phase is to ingeniously generate the set of the new feature set $\widetilde{X}^i$ using the current feature set X^i. Moreover, we want to reduce the number of newly generated features, while the effective ones should not be omitted. The training set, validation set, and test set at this point are represented as D^i_{train}, D^i_{valid} and D^i_{test}.

The search space of original feature generation is:

$$\mathcal{S} = \bigcup_{i=1}^{M} \left\{ \left\{ \bigcup_{1 \le s_1 \le \cdots \le s_i \le M} \{x^{s_1}, \cdots, x^{s_i}\} \right\} \times \mathcal{O}_i \right\} \tag{2}$$

where M is the number of original features and $\mathcal{O}_i$ represents the set of i-ary operators.

The number of elements in the search space is:

$$\mathcal{T} = \sum_{i=1}^{M} (\mathcal{A}^i_M \times |\mathcal{O}_i|) \tag{3}$$

where $\mathcal{A}^k_n$ represents the number of ways of obtaining an ordered subset of k elements from a set of n elements.

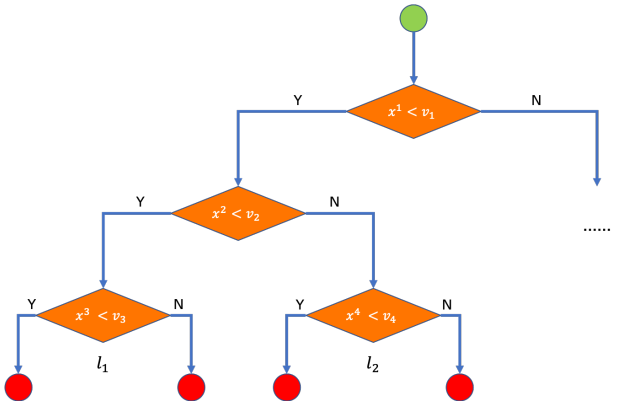


Fig. 2. Example of a regression tree in the XGBoost model

1) Mine feature combination relations: As mentioned earlier, this search space is so large that we have to narrow it down, and the informative feature combinations should not be ignored. First we train a tree model, i.e., XGBoost, on D^i_{train} and D^i_{valid}. Consider a regression tree in the XGBoost model, as shown in Fig. 2. We call $\{\boldsymbol{x}^{(i)}\}$ as the split features and $\{v_i\}$ as the corresponding split values, and the features which do not act as a split feature are called non-split features. The parent node of the leaf node is represented as l_j and the different split features on a path of the tree from the root node to l_j can be represented as p_j. For example, $p_1 = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}\}$

Algorithm 1 Pseudo-code of proposed method SAFE

Input: The raw data set D_{train} and D_{valid}; The operators $\mathcal{O}$; Number of iterations $nIter$ or iteration time $tIter$.
Output: Feature generation function Ψ.

$i \leftarrow 0, t_{start} \leftarrow now()$
while $i < nIter$ and $now() - t_{start} < tIter$ **do**
  Train XGBoost model on D_{train} and D_{valid}.
  Constitute feature combinations from the same path of the model.
  Sort and filter feature combinations to get $\widetilde{P}$ by information gain ratio. See Algorithm 2.
  Apply the operations on the filtered feature combinations $\widetilde{P}$ to get the generated features $\widetilde{X}$.
  Construct candidate features $\hat{X}^i$ by combining the generated features $\widetilde{X}$ and the base features X in this iteration.
  Remove features with low predictive power from $\hat{X}$, and get $\hat{X}^A$. See Algorithm 3.
  Remove redundant features $\hat{X}^A$, and get $\hat{X}^B$. See Algorithm 4.
  Sort the remaining candidate features $\hat{X}^B$ and select the features with high importance to form $\hat{X}^C$
  $D_{train} \leftarrow \left\{\hat{X}^C_{train}, Y_{train}\right\}$, $D_{valid} \leftarrow \left\{\hat{X}^C_{valid}, Y_{valid}\right\}$
  $i \leftarrow i + 1$
end while
The feature generation function Ψ is obtained from the selected features in the last iteration.
return Ψ.

and $p_2 = \left\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(4)}\right\}$ in Fig. 2. All k paths of all trees in the XGBoost model can be represented as $P = \{p_1, \cdots, p_k\}$. Our automatic feature engineering algorithm SAFE is based on two basic assumptions:

- For unary operators, features that generated based on split features are more efficient than that generated based on non-split features.
- For other operators, new features that generated based on split features which from the same path are more efficient than the new features that generated based on split features which from different path, while the latter is still more efficient than features that generated based on non-split features.

We empirically verify the rationality of these two assumptions in section V. Based on these two basic assumptions, we can find features or feature combinations on these paths for feature generation, which not only greatly reduces the search space, guarantees efficiency, but also ensures the validity of feature generation. The search space of feature generation by this way is:

$$\mathcal{S}^* = \bigcup_{i=1}^{k} \bigcup_{j=1}^{|p_i|} \left\{ \left\{ \bigcup_{1 \le s_1 \le \cdots \le s_j \le |p_i|} \left\{p_i^{s_1}, \cdots, p_i^{s_j}\right\} \right\} \times \mathcal{O}_j \right\} \quad (4)$$

where k is the number of paths and $\mathcal{O}_i$ represents the set of i-ary operators.

The maximum number of elements in the search space is:

$$\mathcal{T}^* = \sum_{i=1}^{k} \sum_{j=1}^{|p_i|} (\mathcal{A}^j_{|p_i|} \times |\mathcal{O}_j|) \quad (5)$$

where p_i^j means the j-th feature on the path. It should be noted that some combinations of features on different paths may be the same, so the actual number will be much smaller than this value. It can be found through formulas and experiments that: $\mathcal{T}^* << \mathcal{T}$.

2) Sort feature combinations: To further narrow down the search space of feature generation, we use the information gain ratio to sort the features and feature combinations in the search space. Take a feature combination with q elements $\left\{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(q)}\right\}$ as an example, we already know their split values $V_1, \cdots, V_q$. Here, V_i is a collection because a split feature may appear multiple times in a path. These split features and split values can divide all records into $\prod_{i=1}^{q} (|V_i| + 1)$ parts. The information gain ratio can be calculated by subtracting the original information entropy from the information entropy on these parts. The pseudo-code of the algorithm at this phase is shown in Algorithm 2.

Algorithm 2 Sorting feature combinations

Input: All paths in the XGBoost model P; Number of output features or feature combinations γ.
Output: γ features or feature combinations $\widetilde{P}$.

for each path p^i in P **do**
  for each combination $c_j = \left\{x^{(s_1)}, \cdots, x^{(s_q)}\right\}$ in p^i **do**
    Divide all records into $\prod_{i=1}^{q} (|V_{s_i}| + 1)$ parts according to $\left\{x^{(s_1)}, \cdots, x^{(s_q)}\right\}$ and $\left\{V_{s_1}, \cdots, V_{s_q}\right\}$.
    Calculate the information gain raito of c_j.
  end for
end for
Select the γ features or feature combinations with the highest information gain ratio and mark them as $\widetilde{P}$.
return $\widetilde{P}$.

3) Generate features: γ features or feature combinations with the highest information gain ratio will be used to generate $\sum_{i=1}^{M} (\gamma_i \times |\mathcal{O}_i|)$ new features $\widetilde{X}^i$, where γ_i denotes the number of feature combinations with i features.

Since the number of features that searched and generated is much less than exhaustive searching, this allows us to employ iterative feature generation strategies on large data sets.

C. Feature Selection

Candidate features $\hat{X}^i$ that composed of the base features X^i in this iteration and generated features $\widetilde{X}^i$ might not be of equal importance. To computationally-efficiently select the more informative features, we use a three-step feature selection process: Firstly, according to the information value, the features with low predictive power are removed. Then the redundant features are removed according to the pearson correlation coefficient. Finally, the remaining features are sorted by using a tree based method, i.e., XGBoost [32].

1) Remove uninformative features: Since some of the candidate features will inevitably have little or no impact on the target, we first remove the features with low predictive power. The pseudo-code of the algorithm at this phase is shown in Algorithm 3.

Algorithm 3 Remove features with low predictive power

Input: The data set $D^*_{train} = \left\{\hat{X}_{train}, Y_{train}\right\}$; Threshold of information value α; Number of bins β.
Output: Selected candidate feature set $\hat{X}^A_{train}$.

$\hat{X}^A_{train} \leftarrow \varnothing$
for each candidate feature $x^{(i)}$ in $\hat{X}_{train}$ **do**
  Pack D^*_{train} into β bins at the same frequency.
  Calculate IV_i, the IV of feature $x^{(i)}$ by Eq. (6).
  if $IV_i > \alpha$ **then**
    $\hat{X}^A_{train} = \hat{X}^A_{train} \cup \left\{x^i\right\}$.
  end if
end for
return $\hat{X}^A_{train}$.

Information value (IV) is a very useful concept for feature selection during model building, and it is widely used in the industrial tasks. IV measures the degree to which a feature affects the target. The formula for information value is shown below:

$$IV = \sum_i (\frac{n^i_p}{n_p} - \frac{n^i_n}{n_n}) \times \frac{n^i_p/n_p}{n^i_n/n_n} \tag{6}$$

where n_p and n_n represent the number of all positive records and negative records. n^i_p and n^i_n represent the number of positive records and negative records in the i-th bin.

TABLE I
INFORMATION VALUE

Information Value	Predictive Power
0 to 0.02	Useless for prediction
0.02 to 0.1	Weak predictor
0.1 to 0.3	Medium predictor
0.3 to 0.5	Strong predictor
> 0.5	Extremely strong predictor

As shown in Table I, the rules of thumb guide us on how to remove features with low predictive power. Typically, variables with medium and strong predictive powers are selected for model development. Therefore, we take the threshold of feature selection as $\alpha = 0.1$.

2) Remove redundant features: The candidate features at this time are with certain predictive power, but some of them are redundant. For example, "speed" and "one-hour travel" are highly relevant, we only need to keep one. The pseudo-code of the algorithm at this phase is shown in Algorithm 4.

Algorithm 4 Remove redundant features

Input: The data set $D^*_{train} = \left\{\hat{X}^A_{train}, Y_{train}\right\}$; Threshold of pearson correlation θ.
Output: Selected candidate feature set $\hat{X}^B_{train}$.

$\hat{X}^B_{train} \leftarrow \varnothing$
for each candidate feature $x^{(i)}$ in $\hat{X}^A_{train}$ **do**
  for each candidate feature $x^{(j)}$ in $\hat{X}^A_{train}$ **do**
    if $i < j$ **then**
      Calculate $Pearson(i, j)$ by Eq. (7).
      if $|Pearson(i, j)| > \theta$ **then**
        if $IV_i > IV_j$ **then**
          $\hat{X}^B_{train} = \hat{X}^B_{train} \cup \left\{x^{(i)}\right\}$
        else
          $\hat{X}^B_{train} = \hat{X}^B_{train} \cup \left\{x^{(j)}\right\}$
        end if
      end if
    end if
  end for
end for
return $\hat{X}^B_{train}$.

A pearson correlation is a number between -1 and 1 that indicates the extent to which two features are linearly related. Its absolute value of 1 means that the two features are completely linearly related, and its absolute value of 0 means there is no linear relationship between the two features. $Pearson(i, j)$, a pearson correlation between features $\boldsymbol{x^{(i)}}$ and $\boldsymbol{x^{(j)}}$ is calculated by:

$$Pearson(i,j) = \frac{\sum_{k=1}^{N} (\boldsymbol{x_k^{(i)}} - \overline{\boldsymbol{x^{(i)}}})(\boldsymbol{x_k^{(j)}} - \overline{\boldsymbol{x^{(j)}}})}{\sqrt{\sum_{k=1}^{N} (\boldsymbol{x_k^{(i)}} - \overline{\boldsymbol{x^{(i)}}})^2} \sqrt{\sum_{k=1}^{N} (\boldsymbol{x_k^{(j)}} - \overline{\boldsymbol{x^{(j)}}})^2}} \tag{7}$$

where $\overline{\boldsymbol{x^{(i)}}}$ and $\overline{\boldsymbol{x^{(j)}}}$ means the average of all elements of feature i and feature j.

The larger the absolute value of the correlation coefficient, the stronger the correlation is. Usually, the relative strength of the variable is judged by the range of values in Table II. Therefore, we set the threshold θ of pearson correlation to 0.8. If the pearson correlation coefficient of the two features is greater than 0.8, the feature with the smaller IV of them will be removed.

TABLE II
PEARSON CORRELATION

Pearson Correlation Coefficient	Correlation
0 to 0.2	Very weak or no correlation
0.2 to 0.4	Weak correlation
0.4 to 0.6	Moderate correlation
0.6 to 0.8	Strong correlation
0.8 to 1	Extremely strong correlation

3) Rank feature importance: At this stage, we use a lightweight tree-based method, i.e., XGBoost, to sort the remaining candidate features by the average gain across all splits, and further filter can be performed if a maximum value of the number of final selected features is required to make the later process more efficient.

D. Time complexity Analysis

In this section, we analyze the time complexity of the algorithm. We first analyze the time complexity of some existing algorithms. Reinforcement learning based strategies are beyond our consideration since the executing time of them is too long. We mainly analyze generation-selection based strategies (TFC, FCTree, AutoLearn) and transfer learning or meta-learning based strategies (ExploreKit, LFE). Then we analyze the time complexity of SAFE. It should be noted that for the sake of simplicity and generality, we only consider the first iteration of all iterative algorithms (TFC, ExploreKit, SAFE), and we only consider binary operators. Recall that we denote the number of records and features as N and M, respectively, and $\mathcal{A}_n^k$ represents the number of ways to obtain an ordered subset of k elements from a set of n elements.

1) TFC: TFC [27] generates all legal features and then selects the best ones using information gain. The time complexity of feature generation is $O(N\mathcal{A}_M^2) = O(NM^2)$, the time complexity of feature selection is $O(NM^2)$, the time complexity of feature ranking is $O(M^2 \log M^2)$. For real business data with a large amount of data, $\log M$ is always much smaller than N, so its time complexity is:

$$O_{TFC} = O(M^2(N + \log M^2)) = O(NM^2) \tag{8}$$

2) FCTree: The time complexity of decision tree algorithm is $O(NMD) \leq O(NM \log N)$, in which D is the depth of the tree. FCTree [28] algorithm adds n_e features at each level of decision tree, so its time complexity is $O(NM \log N + \frac{1}{2}N(n_e(\log N - 1)) * \log N) = O((M + n_e \log N)N \log N)$. For real business data with large amount of data, M is always much smaller than $\log N$, so its time complexity is:

$$O_{FCTree} = O(n_e N(\log N)^2) \tag{9}$$

3) AutoLearn: AutoLearn [24] identifies the linear or non-linear relationship between each pair and uses randomized lasso and mutual information for feature selection. The time complexity of feature generation is $O(M^2) \times (O_{Ridge} + O_{KernelRidge}) = O(NM^2)$, the time complexity of feature selection is $O_{Lasso} + O_{MI} = O_{Lasso} + O(NM^2)$, the time complexity of feature ranking is $O(M^2 \log M^2)$. So the time complexity is:

$$O_{AutoLearn} = O(M^2(N + \log M^2)) + O_{Lasso} \tag{10}$$

4) ExploreKit: ExploreKit [20] is a meta-learning based strategies. In the feature generation phase, it needs an exhaustive combination of features, so its time complexity of feature generation is $O(NM^2)$. In the feature ranking phase, it calculates the meta-features associated with the original data set and candidate features, such as entropy-based measures and statistical tests, to score each candidate feature, so its time complexity of feature generation is $O(NM^2) + M^2 \times O_{score}$. In addition, it needs O_{Meta} to train a meta-learning model in advance. So the all-time complexity is:

$$O_{ExploreKit} = O_{Meta} + O(NM^2) + M^2 \times O_{score} \tag{11}$$

5) LFE: LFE [21] is also a meta-learning based strategy. The difference is that it does not require exhaustive feature generation. At the feature selection stage, meta-features are only related to the original features. So the whole complexity can be calculated as:

$$O_{LFE} = O_{Meta} + O(NM) + M^2 \times O_{score} \tag{12}$$

6) SAFE: The most important calculation at the phases of mining feature combination relations and feature importance ranking is to train an XGBoost. Their time complexity is $O(NMK_1D_1 + NM \log R)$ and $O(N\hat{M}^B K_2 D_2 + N\hat{M}^B \log R)$, respectively. Where K_1 and K_2 mean the total number of trees, D_1 and D_2 mean the maximum depth of the tree and R is the maximum number of rows in each block [32] and $\hat{M}^B$ is the number of features after removing redundant features. The number of features after feature generation and removing uninformative features can be denoted as $\hat{M}$ and $\hat{M}^A$, respectively. Next, we analyze the time complexity of the other four phases:

- Sorting feature combinations: As shown in Algorithm 2, there are $2^{D_1}K_1\mathcal{A}_{D_1}^2$ binary feature combinations and the time complexity of this phase is $O(2^{D_1}K_1N{D_1}^2)$.
- Feature generation: As shown in section IV-B3, $\gamma_2 \times |\mathcal{O}_2|$ new features will be generated, so the time complexity of this phase is $O(\gamma_2 N|\mathcal{O}_2|) = O(\gamma_2 N)$.
- Remove uninformative features: As the Algorithm 3 shows, the time complexity of the third and fourth steps is $O(N)$ and $O(N)$, respectively. So the overall time complexity of this phase is $O(\hat{M}N)$.
- Remove redundant features: As the Algorithm 4 shows, pearson correlation is calculated once for each feature pair. Because the time complexity of Pearson correlation calculation is $O(N)$, the overall time complexity of this phase is $O(N(\hat{M}^A)^2)$.

The trees in XGBoost are usually not deep, so we can treat D as a constant and ignore it. For real business data with large amount of data, $\log R < \log N < M < \hat{M}_C < \hat{M}^B < \hat{M}^A <$

$\hat{M} << N$, $\hat{M} = \gamma_2|\mathcal{O}_2|$ and $\log R < K_1, K_2$. So the all time complexity is:

$$\begin{aligned} O_{SAFE} &= O(N(\hat{M}^A)^2) + O(N\hat{M}^B K_2) \\ &\leq O(N\hat{M}(\hat{M} + K_2)) = O(N\gamma_2(\gamma_2 + K_2)) \\ &\leq O(N2^{D_1}K_1\mathcal{A}_{D_1}^2(2^{D_1}K_1\mathcal{A}_{D_1}^2 + K_2)) \\ &= O(NK_1(K_1 + K_2)) \end{aligned} \tag{13}$$

As shown in Eq. (13), we can easily control the number of features generated and the time complexity of the algorithm by controlling the total number of trees of XGBoost.

E. Discussion

The time complexity and space complexity of our algorithm is very low and can be adjusted flexibly according to actual needs. Below we will continue to discuss whether the algorithm meets the requirements in section I.

1) Strong applicability: Our algorithm is user-friendly and does not require learning a cumbersome model like reinforcement learning and transfer learning based methods. Besides, the hyperparameters which needed to set in advance are only used to control the complexity of the algorithm, such as the number of iterations or iteration time, the number of trees in the forest and the depth of each tree. Therefore, the setting of these hyperparameters is not complicated.

2) Distributed computing: XGBoost is recognized as an algorithm that leverages the parallelism of computing resources, and it has been proven that XGBoost can push the limits of computing power for boosted trees algorithms. At the same time, other aspects of our algorithm can be easily parallelized, such as calculating the information value of the individual feature and the pearson correlation of each feature pair in parallel.

3) Real-time inference: In our algorithm, whether newly generated features can be used for real-time inference depends on the operators $\mathcal{O}$ that used for feature generation. Users can choose different operators according to the actual situation to meet the real-time requirements of the business.

V. Experiments

For simplicity and versatility, we only select four basic binary operators $+$, $-$, $\times$ and $\div$ when experimenting with each algorithm.

TABLE IV
THE INFORMATION OF THE BENCHMARK DATA SETS

Dataset	#Train	#Valid	#Test	#Dim
valley	900	-	312	100
banknote	1,000	-	372	4
gina	2,800	-	668	970
spambase	3,800	-	801	57
phoneme	4,500	-	904	5
wind	5,000	-	1,574	14
ailerons	9,000	2,000	2,750	40
eeg-eye	10,000	2,000	2,980	14
magic	13,000	3,000	3,020	10
nomao	22,000	6,000	6,000	118
bank	35,211	4,000	6,000	51
vehicle	60,000	18,528	20,000	100

A. Experiments on benchmark data sets

We first conduct experiments on 12 benchmark data sets, with different sample and feature size. All of these data are available on the OpenML database[1]. The number of training, validation, and test samples are shown in Table IV, with the feature size. Note that for the data set whose sample size is less than 10000, no validation set is splitted, and we simply use training data for validation if necessary. All experiments are performed on a 4-core computer with 16GB of RAM.

1) Algorithms for comparison: We compare our model with the original features (ORIG), two other state-of-the-art feature generating algorithms, i.e., FCTree [28] and TFC [27], and two of our own comparison algorithms, i.e., Random (RAND) and SAFE-Important (IMP). We use Area Under Curve (AUC) as the evaluation metric.

RAND algorithm randomly selects γ different feature combinations of all original features for feature generation. Different from it, IMP algorithm only randomly selects γ different feature combinations with the split features of XGBoost for feature generation. RAND and IMP follow the same feature selection process as SAFE. For the convenience of comparison, The maximum number of RAND, IMP, and SAFE output features are set to $2M$. Features generated by FCTree will also be reduced to $2M$ according to information gain. Moreover, TFC, RAND, IMP and SAFE only perform one iteration.

2) Classification performance: We evaluate the generated features (and also the original features) of each compared algorithm on 9 state-of-the-art classification algorithms (CLF), which are AdaBoost (AB), Decison Tree (DT), Extremely randomized Trees (ET), k nearest neighbors (kNN), Logistic Regression (LR), Multi Layered Perceptron (MLP), Random Forest (RF), SVM with linear kernel (SVM) and XGBoost. All parameters of these algorithms are set as the default values in scikit-learn [33] and XGBoost [32]. We performed n times experiments, and obtain the final results by averaging the results of these experiments (n is 100 for the first 9 data sets and 10 for the rest data sets).

The reported performances are measured in terms of AUC, which are shown in Table III. The value in the table means $100 \times$ AUC. It can be seen from the experimental results that SAFE has a significant advantage over all other compared algorithms no matter what model is performed after the feature generation process. Compared with the original feature space, the features generated by our model can improve the overall prediction AUC by 6.50% on average. Compared with FCTree and TFC, SAFE can improve the performance by 2.03% and 3.74% on average, respectively. What's more, SAFE performs better than RAND and IMP, indicating that our algorithm does mine a combination of features that are more likely to generate better features.

3) Feature importance: We compare the importance of generated features with original features. We combine the M original features with the top-ranked generated features (up to M) to form a new data set and use random forest to

[1]https://www.openml.org/

TABLE III
CLASSIFICATION PERFORMANCE ON BENCHMARK DATA SETS

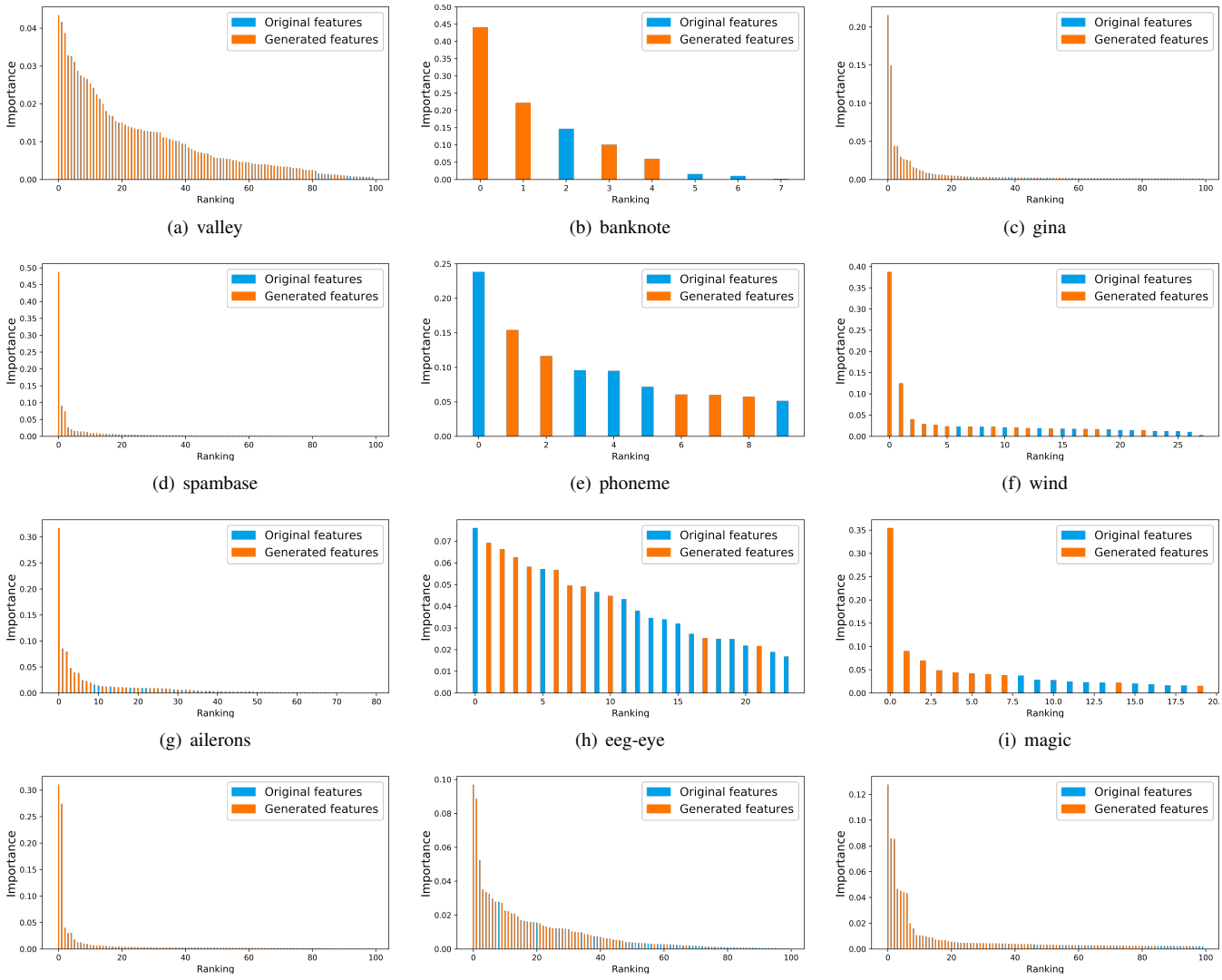
Dataset	CLF	ORIG	FCT	TFC	RAND	IMP	SAFE	Dataset	CLF	ORIG	FCT	TFC	RAND	IMP	SAFE
	AB	52.10	78.27	87.33	88.20	88.30	**88.92**		AB	99.55	99.38	96.59	**99.62**	99.58	99.53
	DT	54.38	70.78	77.37	77.37	77.43	**78.32**		DT	98.20	98.90	96.38	98.83	98.78	**98.99**
	ET	55.54	76.64	**88.44**	85.77	86.03	88.33		ET	99.83	99.76	96.95	99.88	99.88	**99.93**
	*k*NN	51.84	78.69	**94.81**	93.01	92.69	93.66		*k*NN	99.86	99.75	97.45	99.88	99.90	**99.96**
valley	LR	58.54	80.69	92.53	93.31	93.39	**93.80**	banknote	LR	94.80	96.47	88.14	97.24	97.12	**97.79**
	MLP	59.63	81.27	92.71	93.93	93.97	**94.08**		MLP	98.77	98.51	88.07	99.25	99.21	**99.36**
	RF	54.54	76.31	87.06	84.34	84.34	**87.11**		RF	99.01	99.46	96.98	99.51	99.48	**99.57**
	SVM	65.62	81.53	93.39	94.22	94.38	**94.94**		SVM	98.29	98.03	88.22	98.56	98.53	**98.64**
	XGB	54.75	86.92	94.75	95.14	95.23	**95.68**		XGB	99.83	99.88	99.25	99.91	99.89	**99.91**
	AB	85.15	85.24	86.22	85.60	87.54	**90.19**		AB	93.61	93.49	92.06	93.61	93.59	**93.74**
	DT	85.50	85.75	84.01	85.69	86.11	**87.53**		DT	91.28	91.06	90.72	91.51	91.65	**92.18**
	ET	91.65	91.31	88.44	91.59	92.27	**92.79**		ET	94.22	93.09	92.43	94.34	**94.48**	94.45
	*k*NN	83.95	**91.98**	88.46	85.03	88.03	89.08		*k*NN	89.10	89.30	**91.21**	90.38	90.44	91.09
gina	LR	85.12	86.23	86.38	85.61	87.34	**90.35**	spambase	LR	87.71	87.86	83.68	89.48	89.53	**90.15**
	MLP	93.09	93.81	89.09	93.21	93.75	**93.83**		MLP	93.81	93.43	91.04	93.84	93.81	**93.94**
	RF	90.50	90.45	87.96	90.40	91.23	**91.86**		RF	93.94	92.74	92.58	93.96	94.01	**94.17**
	SVM	81.40	86.25	84.90	82.07	84.58	**88.59**		SVM	90.21	90.08	86.51	91.13	91.07	**91.60**
	XGB	97.76	97.33	96.46	97.74	97.71	**97.90**		XGB	98.27	98.18	97.86	98.34	98.41	**98.46**
	AB	76.90	79.25	73.98	79.70	79.64	**79.90**		AB	**85.62**	85.43	83.84	85.47	85.40	85.44
	DT	**84.07**	83.92	79.68	83.65	83.95	84.05		DT	79.76	79.63	78.41	79.99	80.12	**80.13**
	ET	86.31	85.69	82.07	87.03	86.87	**87.09**		ET	84.41	84.49	82.92	85.03	84.90	**85.06**
	*k*NN	83.92	83.65	79.59	84.60	84.47	**84.65**		*k*NN	83.91	**84.91**	83.49	84.56	84.59	84.80
phoneme	LR	66.32	66.91	64.69	67.21	67.42	**67.47**	wind	LR	85.12	85.18	83.90	85.28	85.28	**85.34**
	MLP	76.56	77.23	72.94	78.25	78.27	**78.39**		MLP	**86.77**	86.52	84.93	86.67	86.66	86.66
	RF	85.72	85.72	81.25	86.02	86.13	**86.14**		RF	84.66	84.65	83.39	85.10	85.11	**85.24**
	SVM	66.88	67.21	65.20	68.15	68.47	**68.68**		SVM	85.09	85.12	84.24	85.32	85.30	**85.46**
	XGB	**93.81**	93.56	90.96	93.63	93.67	93.58		XGB	93.72	93.71	92.51	93.69	**93.74**	93.70
	AB	87.16	**87.16**	81.58	87.08	87.07	87.14		AB	73.48	74.61	73.48	74.76	74.59	**75.18**
	DT	82.92	**83.46**	78.38	83.28	83.37	83.46		DT	82.37	83.46	82.55	83.76	83.55	**83.99**
	ET	83.32	84.14	79.04	85.04	85.64	**86.25**		ET	89.24	90.34	89.41	90.31	90.27	**90.67**
	*k*NN	85.72	85.51	78.95	86.08	86.41	**86.60**		*k*NN	86.32	89.38	86.49	89.37	89.48	**90.34**
ailerons	LR	87.47	87.55	80.61	87.58	87.60	**87.64**	eeg-eye	LR	51.25	52.92	50.61	53.69	53.73	**53.75**
	MLP	87.54	87.79	80.09	87.83	**87.99**	87.96		MLP	54.58	55.71	52.52	55.85	55.81	**56.64**
	RF	84.46	85.60	79.29	86.05	86.39	**86.82**		RF	88.21	89.02	87.84	89.03	88.99	**89.34**
	SVM	87.49	87.55	80.54	87.65	87.66	**87.75**		SVM	54.56	56.56	54.62	56.97	56.80	**57.23**
	XGB	95.48	95.62	90.62	95.58	95.60	**95.64**		XGB	91.70	93.02	91.63	92.89	92.84	**93.28**
	AB	81.01	81.89	77.72	82.53	82.53	**82.95**		AB	93.13	93.47	92.46	93.66	93.73	**94.04**
	DT	79.89	80.04	77.51	80.45	80.43	**80.67**		DT	93.26	92.90	91.95	93.06	93.14	**93.28**
	ET	82.64	81.60	80.48	83.29	83.41	**83.76**		ET	95.58	95.44	95.07	95.56	95.63	**95.69**
	*k*NN	79.55	80.17	79.40	80.70	80.89	**81.10**		*k*NN	94.25	94.09	93.29	94.29	94.32	**94.44**
magic	LR	74.24	75.84	75.84	77.00	76.92	**77.31**	nomao	LR	93.30	93.12	91.93	93.73	93.74	**93.76**
	MLP	83.66	83.70	80.38	84.26	84.28	**84.43**		MLP	95.00	94.84	91.94	94.98	95.02	**95.19**
	RF	83.70	82.93	80.87	84.15	84.21	**84.43**		RF	95.53	95.33	94.27	95.49	95.52	**95.62**
	SVM	74.09	75.92	75.34	76.94	76.89	**77.14**		SVM	93.54	93.42	92.01	94.01	94.03	**94.05**
	XGB	92.14	92.21	89.74	92.68	92.70	**92.88**		XGB	98.95	98.97	98.73	99.01	99.05	**99.13**
	AB	67.05	67.83	67.09	67.37	67.46	**68.95**		AB	85.81	85.86	83.88	86.52	86.48	**86.69**
	DT	70.88	70.67	68.28	70.24	70.15	**70.94**		DT	78.44	78.42	75.81	**79.53**	79.21	79.27
	ET	64.65	67.40	65.39	68.09	68.70	**68.77**		ET	85.53	85.17	83.03	85.90	85.89	**86.19**
	*k*NN	61.49	64.75	**68.20**	62.97	63.57	64.76		*k*NN	78.58	81.64	**82.19**	79.24	79.97	81.88
bank	LR	65.03	65.98	61.16	66.03	66.03	**66.28**	vehicle	LR	85.66	85.68	83.02	85.74	85.71	**85.96**
	MLP	72.37	72.36	68.26	71.79	**72.70**	72.26		MLP	86.79	**87.10**	83.27	86.63	86.48	87.03
	RF	66.54	66.29	65.51	67.53	67.71	**68.24**		RF	85.55	85.18	82.93	86.28	86.16	**86.50**
	SVM	63.95	64.13	60.61	64.15	64.28	**64.44**		SVM	85.31	85.33	82.54	85.48	85.37	**85.81**
	XGB	91.84	91.98	90.77	91.90	92.07	**92.29**		XGB	91.86	92.05	90.85	92.30	92.29	**92.47**

score feature importance. The experimental results are shown in Fig. 3. It is evident that the new features generated by SAFE (indicated in orange) are relatively more important than the original features (indicated in blue), which validates the effectiveness of the generated features.

4) Execution time: Section IV-D has analyzed the time complexity of each method. Table V lists the actual executing time. It can be seen that SAFE has a great advantage and the execution time of it is on average 0.13 (0.08) times the execution time of FCTree (TFC).

5) Feature stability: We further compare the stability of the generated features of each algorithm. The basic idea is that the generated features are more stable if the same features are generated each time when we repeat the automatic feature engineering procedure; and if the features generated each time are different, then the stability of the generated features is unsatisfactory.

Suppose we have conducted T experiments, each time the automatic feature engineering algorithm will generate $2M$ features. Therefore, a total of $2MT$ features will be



(a) valley (b) banknote (c) gina

(d) spambase (e) phoneme (f) wind

(g) ailerons (h) eeg-eye (i) magic

(j) nomao (k) bank (l) vehicle

Fig. 3. Feature importance

TABLE V
EXECUTION TIME (IN SECOND)

Dataset	FCT	TFC	RAND	IMP	SAFE
valley	9.80	228.93	0.55	0.65	0.73
banknote	0.16	0.27	0.08	0.31	0.12
gina	84.03	95.73	3.39	5.28	5.31
spambase	23.84	262.11	2.85	3.23	3.17
phoneme	10.83	2.48	0.46	0.58	0.51
wind	25.03	22.87	2.13	2.39	2.33
ailerons	80.73	336.53	2.12	2.57	2.72
egg-eye	58.88	42.13	1.09	1.20	1.18
magic	52.45	36.79	2.55	2.96	3.32
nomao	104.59	1469.19	22.22	26.61	28.82
bank	838.17	552.48	13.70	13.81	12.28
vehicle	1355.19	2748.89	52.86	40.34	62.14

generated. Their distribution can be expressed as $Dis = \{(x^{(s_1)}, t_1), \cdots, (x^{(s_k)}, t_k)\}$, where s_i represents the feature id, t_i represents the number of occurrences of the feature and $t_1 \geq t_2 \geq \cdots \geq t_k$. Therefore, the distribution with the best feature stability is $\hat{Dis} = \{(x^{(s_1)}, T), \cdots, (x^{(s_{2M})}, T)\}$, and the worst is $\widetilde{Dis} = \{(x^{(s_1)}, 1), \cdots, (x^{(s_{2MT})}, 1)\}$.

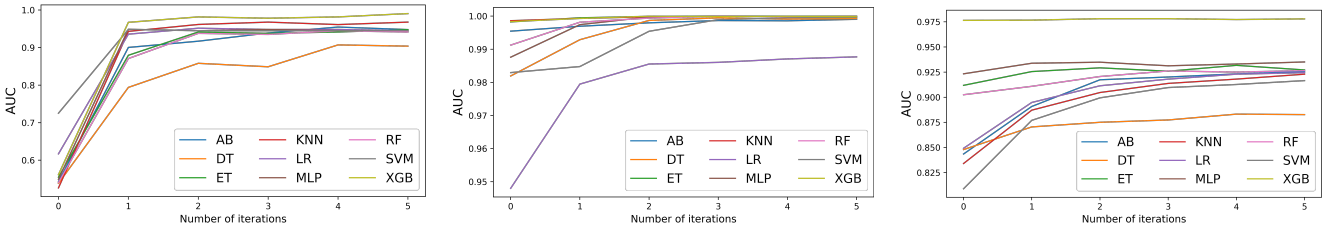
We use Jensen-Shannon Divergence (JSD) [34] to evaluate the stability of the feature distribution generated by different automatic feature engineering algorithms. JSD is a variant of Kullback-Leibler divergence (KLD) [35], which is converted as:

$$JSD(P||Q) = \frac{1}{2} \times (KLD(P||R) + KLD(Q||R)) \quad (14)$$

where $R = \frac{1}{2} \times (P + Q)$ and KLD is calculated as:

$$KLD(P||Q) = \sum_i P(i) ln \frac{P(i)}{Q(i)} \quad (15)$$

We take T as 100 and calculate the stability of the generated features of each algorithm. That is, the JSD between the actual



(a) valley (b) banknote (c) gina

Fig. 4. Performance at different iterations

feature distribution Dis and the ideal distribution $\hat{Dis}$. The smaller the value is, the better it is. The experimental results are shown in Table VI. We did not compare the TFC algorithm because the execution time of TFC is too long, so it is difficult to calculate so many times. From the experimental results, it can be seen that the stability of the generated features of SAFE has certain advantages.

TABLE VI
FEATURE STABILITY

Dataset	FCT	RAND	IMP	SAFE
valley	0.6991	0.4933	0.4947	**0.4710**
banknote	0.4405	0.3233	0.3174	**0.1197**
gina	0.5700	0.4639	0.4739	**0.4163**
spambase	0.5101	0.4571	0.4399	**0.3587**
phoneme	**0.1947**	0.3269	0.3294	0.2616
wind	0.3707	0.3608	0.3570	**0.3230**
ailerons	0.4440	0.4590	0.3963	**0.3330**
egg-eye	0.3529	0.3768	0.3741	**0.3212**
magic	**0.1847**	0.3306	0.3384	0.2620
nomao	0.5061	0.5032	0.4735	**0.4065**
bank	0.3713	0.4240	0.4072	**0.2853**

6) Performance at different iterations: We then validate whether the performance can be further improved as the iteration process goes on. We set the iteration round to 5, and the sampled results are shown in Fig. 4. As we can see, the performance may further be improved as the round proceeds, and become stable after some rounds. This is reasonable, since that in the first some rounds, more useful feature combinations can be excavated so that the performance can be further improved, and after some rounds, there may be no new useful feature combinations can be found, thus the features will not be updated, and the performance keeps unchanged.

B. Experiments on business data sets

Experiments on extra-large scale business data sets are further conducted to verify the effectiveness and scalability of the proposed method on real industrial tasks. The data sets come from the tasks for fraud detection in Ant Financial, which aims at finding the potential fraud transactions (or malicious users), so that the system can stop these transactions (or catch these users) to avoid the economic losses. Table VII presents the detailed information of these data sets. As we can see, the number of samples is extremely large (e.g., up to 8 million training samples for Data3), which severely hinders the employment of many preceding state-of-the-art methods. All parameters of the evaluated models are set as the default values as before.

TABLE VII
THE INFORMATION OF THE BUSSINESS DATA SETS

Dataset	#Train	#Valid	#Test	#Dim
Data1	2,502,617	625,655	625,655	81
Data2	7,282,428	1,820,607	1,820,607	44
Data3	8,000,000	2,000,000	2,000,000	73

The results are shown in table VIII. TFC and FCTree are not compared since the execution time is too long for these two methods when applying for these extremely large scale data sets. Thanks to the delicate design in the feature generation procedure of SAFE, the whole time consuming is acceptable even for the industrial tasks. More important, as we can see, the proposed method SAFE can consistently improve the performance, which validate the effectiveness of the proposed method when applying in real industrial tasks and make it a choice for performing automatic feature engineering for extremely large scale industrial data sets. Actually, this framework has been deployed in our system, providing help for many different real-world tasks.

TABLE VIII
CLASSIFICATION PERFORMANCE OF BUSINESS DATA SETS

Dataset	CLF	ORIG	RAND	IMP	SAFE
Data1	LR	93.07	95.81	95.83	**95.93**
	RF	96.26	97.62	97.60	**98.20**
	XGB	97.04	96.59	97.35	**97.46**
Data2	LR	90.24	90.26	90.26	**90.31**
	RF	88.26	88.71	88.61	**88.95**
	XGB	90.13	90.33	90.44	**90.61**
Data3	LR	89.64	89.82	89.84	**89.94**
	RF	86.98	87.05	88.26	**88.59**
	XGB	89.77	89.74	89.92	**90.37**

VI. CONCLUSION

Automatic feature engineering has become an important topic of autoML in recent years, and many different methods have been proposed to handle this task. However, the efficiency and scalability of these methods are still far from satisfactory, especially for industrial tasks, while automatically performing

feature engineering is severely demanded. In this paper, we propose a scalable and efficient method named SAFE for automatic feature engineering. Extensive experiments on both benchmark data sets and extra-large scale business data sets are conducted, and detailed analysis is provided, which shows that the proposed method can provide prominent efficiency and competitive effectiveness when comparing with other methods.

REFERENCES

[1] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. V. Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, and D. Sampath, "The youtube video recommendation system," in *Proceedings of the 2010 ACM Conference on Recommender Systems, RecSys 2010, Barcelona, Spain, September 26-30, 2010*, 2010, pp. 293–296.

[2] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," in *The Adaptive Web, Methods and Strategies of Web Personalization*, 2007, pp. 325–341.

[3] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, and J. Q. Candela, "Practical lessons from predicting clicks on ads at facebook," in *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising, ADKDD 2014, August 24, 2014, New York City, New York, USA*, 2014, pp. 5:1–5:9.

[4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, 2009.

[5] Y. Zhang, L. Li, J. Zhou, X. Li, and Z. Zhou, "Anomaly detection with partially observed anomalies," in *Companion of the Web Conference 2018, WWW 2018, Lyon, France, April 23-27, 2018*, 2018, pp. 639–646.

[6] Y. Zhang, J. Zhou, W. Zheng, J. Feng, L. Li, Z. Liu, M. Li, Z. Zhang, C. Chen, X. Li, Y. A. Qi, and Z. Zhou, "Distributed deep forest and its application to automatic detection of cash-out fraud," *ACM TIST*, vol. 10, no. 5, pp. 55:1–55:19, 2019.

[7] A. Lacerda, M. Cristo, M. A. Gonçalves, W. Fan, N. Ziviani, and B. A. Ribeiro-Neto, "Learning to advertise," in *SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, Washington, USA, August 6-11, 2006*, 2006, pp. 549–556.

[8] B. A. Ribeiro-Neto, M. Cristo, P. B. Golgher, and E. S. de Moura, "Impedance coupling in content-targeted advertising," in *SIGIR 2005: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Salvador, Brazil, August 15-19, 2005*, 2005, pp. 496–503.

[9] H. Zhu, J. Jin, C. Tan, F. Pan, Y. Zeng, H. Li, and K. Gai, "Optimized cost per click in taobao display advertising," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, 2017, pp. 2191–2200.

[10] M. Alajmi, K. Awedat, A. Essa, F. Alassery, and O. S. Faragallah, "Efficient face recognition using regularized adaptive non-local sparse coding," *IEEE Access*, vol. 7, pp. 10 653–10 662, 2019.

[11] H. W. F. Yeung, J. Li, and Y. Y. Chung, "Improved performance of face recognition using CNN with constrained triplet loss layer," in *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*, 2017, pp. 1948–1955.

[12] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*, ser. Springer Series in Statistics. Springer, 2009.

[13] Q. Yao, M. Wang, H. J. Escalante, I. Guyon, Y. Hu, Y. Li, W. Tu, Q. Yang, and Y. Yu, "Taking human out of learning applications: A survey on automated machine learning," *CoRR*, vol. abs/1810.13306, 2018.

[14] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning*. Springer, 2019.

[15] M. Zöller and M. F. Huber, "Survey on automated machine learning," *CoRR*, vol. abs/1904.12054, 2019.

[16] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *CoRR*, vol. abs/1908.00709, 2019.

[17] R. Gaudel and M. Sebag, "Feature selection as a one-player game," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, 2010, pp. 359–366.

[18] U. Khurana, H. Samulowitz, and D. S. Turaga, "Feature engineering for predictive modeling using reinforcement learning," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, 2018, pp. 3407–3414.

[19] J. Zhang, J. Hao, F. Fogelman-Soulié, and Z. Wang, "Automatic feature engineering by deep reinforcement learning," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*, 2019, pp. 2312–2314.

[20] G. Katz, E. C. R. Shin, and D. Song, "Explorekit: Automatic feature generation and selection," in *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, 2016, pp. 979–984.

[21] F. Nargesian, H. Samulowitz, U. Khurana, E. B. Khalil, and D. S. Turaga, "Learning feature engineering for classification," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 2017, pp. 2529–2535.

[22] J. M. Kanter and K. Veeramachaneni, "Deep feature synthesis: Towards automating data science endeavors," in *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Campus des Cordeliers, Paris, France, October 19-21, 2015*, 2015, pp. 1–10.

[23] H. T. Lam, J. Thiebaut, M. Sinn, B. Chen, T. Mai, and O. Alkan, "One button machine for automating feature engineering in relational databases," *CoRR*, vol. abs/1706.00327, 2017.

[24] A. Kaul, S. Maheshwary, and V. Pudi, "Autolearn - automated feature generation and selection," in *2017 IEEE International Conference on Data Mining, ICDM 2017, New Orleans, LA, USA, November 18-21, 2017*, 2017, pp. 217–226.

[25] Y. Luo, M. Wang, H. Zhou, Q. Yao, W. Tu, Y. Chen, W. Dai, and Q. Yang, "Autocross: Automatic feature crossing for tabular data in real-world applications," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019.*, 2019, pp. 1936–1945.

[26] S. Markovitch and D. Rosenstein, "Feature generation using general constructor functions," *Machine Learning*, vol. 49, no. 1, pp. 59–98, 2002.

[27] S. Piramuthu and R. T. Sikora, "Iterative feature construction for improving inductive learning algorithms," *Expert Syst. Appl.*, vol. 36, no. 2, pp. 3401–3406, 2009.

[28] W. Fan, E. Zhong, J. Peng, O. Verscheure, K. Zhang, J. Ren, R. Yan, and Q. Yang, "Generalized and heuristic-free feature construction for improved accuracy," in *Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 - May 1, 2010, Columbus, Ohio, USA*, 2010, pp. 629–640.

[29] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep & cross network for ad click predictions," in *Proceedings of the ADKDD'17, Halifax, NS, Canada, August 13 - 17, 2017*, 2017, pp. 12:1–12:7.

[30] W. Song, C. Shi, Z. Xiao, Z. Duan, Y. Xu, M. Zhang, and J. Tang, "Autoint: Automatic feature interaction learning via self-attentive neural networks," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, 2019, pp. 1161–1170.

[31] Y. Zhang and L. Li, "Interpretable MTL from heterogeneous domains using boosted tree," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, 2019, pp. 2053–2056.

[32] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, 2016, pp. 785–794.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[34] J. Lin, "Divergence measures based on the shannon entropy," *IEEE Trans. Information Theory*, vol. 37, no. 1, pp. 145–151, 1991.

[35] D. J. C. MacKay, *Information theory, inference, and learning algorithms*. Cambridge University Press, 2003.