OPENFE: AUTOMATED FEATURE GENERATION BEYOND EXPERT-LEVEL PERFORMANCE

Anonymous authors
Paper under double-blind review

ABSTRACT

The goal of automated feature generation is to liberate machine learning experts from the tedious task of manual feature generation, which is a typical method for improving the learning performance of tabular data. How to efficiently and accurately identify useful features from a vast pool of candidate features is the major challenge in automated feature generation. In this paper, we present OpenFE, an automated feature generation tool that provides competitive results against machine learning experts. OpenFE achieves both good efficiency and accuracy with two components: 1) A novel feature boosting method to accurately estimate the incremental performance of candidate features. 2) A feature-scoring framework to recall useful features from huge candidates by successive featurewise halving and feature importance attribution. We evaluate OpenFE in two famous Kaggle competition with thousands of data science teams participating. In the best case, OpenFE with a simple baseline model can beat 99.3% data science teams, demonstrating for the first time that automated feature generation can outperform machine learning experts.

1 INTRODUCTION

Feature generation is an important yet challenging task when applying machine learning methods to tabular data. Tabular data, where each row represents an instance and each column corresponds to a distinct feature, are ubiquitous in industrial applications and machine learning competitions. It has been well recognized that the quality of features has a significant impact on the learning performance of tabular data (Domingos, 2012). Feature generation aims at transforming the base features into more informative ones to better represent the data and improve the learning performance. For example, the Price-to-Earnings ratio (P/E ratio) is $(\text{Share Price})/(\text{Earnings per Share})$, which tells investors how much a company is worth. In practice, data scientists typically use their domain knowledge to find useful feature transformations in a trial-and-error manner, which requires tremendous human labor and expertise.

Since manual feature generation is time-consuming and requires case-by-case domain knowledge, automated feature generation emerges as an important topic in automated machine learning (Yao et al., 2018). Expand-and-reduce is arguably the most prevalent framework in automated feature generation, in which we first expand the candidate features, and then eliminate redundant ones (Kanter & Veeramachaneni, 2015; Shi et al., 2020; Luo et al., 2019). There are two challenges in a typical expand-and-reduce practice. First, the number of candidate features is usually huge in the majority of industrial applications. Calculating all the candidate features is not only computationally expensive, but also infeasible due to the enormous amount of memory required. The second challenge is how to accurately and efficiently estimate the incremental performance of a new feature, i.e., how much performance improvement a new candidate feature can offer when added to the base feature set. A majority of the existing methods use statistical tests to determine whether a new feature should be included (Kanter & Veeramachaneni, 2015; Lam et al., 2021; Shi et al., 2020). However, statistically significant features do not equal to good predictors (Lo et al., 2015). Features may be significantly associated with the target simply for a small group of instances in the population, thereby leading to poor prediction on the population (Lo et al., 2015). This phenomenon is pervasive in different types of data across the sciences (Lo et al., 2015) as well as the social sciences (Ward et al., 2010; Welch & Goyal, 2008). Besides, the effectiveness of a new feature may be covered by the base feature set, even if the new feature is significantly associated with the target. Moreover,

statistical tests fail to consider interaction effects between features, which are critical in improving the learning performance.

In this paper, we propose OpenFE, a powerful automated feature generation algorithm that can accurately retrieves useful candidate features to enhance the learning performance. First, we propose a feature boosting method that can accurately evaluate the incremental performance of new features in an efficient way. Second, in order to deal with the huge number of candidate features, we propose a two-stage evaluation framework. In the first stage, we propose a successive featurewise pruning algorithm to quickly reduce the number of candidate features. In the second stage, we propose a feature importance attribution method to rank remaining candidate features according to their contribution in improving the learning performance. We validate OpenFE on various datasets and Kaggle competitions, where OpenFE outperforms existing methods. In a fraud detection competition, the baseline model with features generated by OpenFE beats 99.3% of 6351 data science teams. More importantly, the features generated by OpenFE result in comparable or even larger performance improvement than those provided by the competition's top winner, demonstrating for the first time that automated feature generation can outperform machine learning experts.

In addition to proposing a novel method, this paper intends to address two significant problems that hinder the research process of automated feature generation. The first problem is a lack of established benchmarks (such as ImageNet (Deng et al., 2009) for computer vision) for feature generation. Existing papers perform evaluation on different datasets and it is unclear which feature generation method performs better than others. In addition, the majority of previous works do not open-source their codes and datasets, which makes the comparison even more difficult. To this end, we reproduce main methods for automated feature generation and build a benchmark on various types of datasets and tasks to facilitate future research.

The second problem is the lack of conclusive evidence regarding the necessity of feature generation in the era of deep learning. Deep neural networks (DNNs) are widely recognized for its ability in extracting feature representations. In recent years, a variety of DNNs have been carefully developed for modelling tabular data (Arık & Pfister, 2021; Gorishniy et al., 2021), and several of them have demonstrated their efficiency in automatic feature interaction learning (Song et al., 2019; Wang et al., 2021). To this end, we extensively evaluate the effect of OpenFE on a variety of DNNs on a wide range of datasets. We demonstrate that generating new transformed features with OpenFE can greatly enhance the learning performance of existing DNN architectures.

We summarize the contributions of our paper as follows:

- We propose a novel automated feature generation method that can accurately identify useful new features that can enhance learning performance. For the first time, we validate that automated feature generation can outperform human experts on Kaggle competitions.
- We build a benchmark on a diverse set of tasks and review main methods for automated feature generation. Experimental results show that our method achieves state-of-the-art on seven benchmarking datasets.
- We reveal that OpenFE can greatly improve the learning performance of existing DNN architectures and feature generation is a crucial component of modelling tabular data.

2 RELATED WORKS

Expand-and-reduce is arguably the most popular framework in automated feature generation, where we first expand the feature space and then eliminate redundant ones. Most of existing automated feature generation approaches employ statistical methods to identify and remove redundant features. For example, Data Science Machine (DSM) (Kanter & Veeramachaneni, 2015) uses the f-value between the feature and the target and One Button Machine (OneBM) uses Chi-square test to remove redundant features. DSM and OneBM focuses on feature generation in relational databases, which is different from our setting in tabular data. FCTree (Fan et al., 2010) uses information gain to select new features during the construction of decision trees. SAFE (Shi et al., 2020) uses information value to exclude uninformative features. AutoCross (Luo et al., 2019) and AutoFeat (Horn et al., 2019) uses the improvement on a linear regression model to determine whether a new feature should be included. However, evaluation methods based on linear regression fail to consider interaction effects and the non-linear relationship between features and targets, thus may not be accurate in

many scenarios. LFE (Nargesian et al., 2017) and ExploreKit (Katz et al., 2016) design meta features based on statistical methods and use meta learning approach to determine the effectiveness of new features. LFE and ExploreKit are confined to binary-classification tasks and a specific set of operators. It is not user-friendly to extend these methods to other tasks or operators, which requires designing a different set of meta features, collecting different datasets for offline training, and retraining the meta learning model.

How to search for high-order feature transformations is challenging in automated feature generation due to the explosion in search space (Chen et al., 2019). Some learning-based methods directly search for high-order feature transformations. For example, Neural Feature Search (Chen et al., 2019) utilizes a recurrent neural network controller to search for a series of transformations. (Khurana et al., 2018) uses reinforcement learning to traverse a transformation graph for high-order feature transformations. However, the effectiveness of high-order feature transformations should be evaluated in light of all its low-order components. For example, a second-order feature transformation $f_1 \times f_2 \times f_3$ is effective only if it has additional effectiveness to all its first-order components $f_1 \times f_2$, $f_1 \times f_3$, and $f_2 \times f_3$. In addition, because high-order feature transformations are typically more difficult to interpret than low-order ones, low-order feature transformations are favoured in industrial applications where interpretability is important.

3 Problem Definition

For a given training dataset X, we first split it into a sub-training set X_{tr} and a validation set X_{vld}. Then we represent X_{tr} with a feature set $\mathcal{T} + \mathcal{S}$, where $\mathcal{T}$ is the base feature set and $\mathcal{S}$ is the generated feature set. We use a learning algorithm $\mathcal{L}$ to learn a model $\mathcal{L}(X_{tr}, \mathcal{T} + \mathcal{S})$. Then, we represent the validation set X_{vld} with the same feature set $\mathcal{T} + \mathcal{S}$ and calculate the evaluation metric $\mathcal{E}(\mathcal{L}(X_{tr}, \mathcal{T} + \mathcal{S}), X_{vld}, \mathcal{T} + \mathcal{S})$. Now, we formally define the feature generation problem as:

$$\max_{\mathcal{S} \in A(\mathcal{T})} \mathcal{E}(\mathcal{L}(X_{tr}, \mathcal{T} + \mathcal{S}), X_{vld}, \mathcal{T} + \mathcal{S}) \tag{1}$$

where $A(\mathcal{T})$ is the set of all possible candidate features generated from the base feature set. The goal of feature generation is to find a feature set $\mathcal{S}$ from all the candidate features $A(\mathcal{T})$ that can maximize the evaluation metric. We determine $A(\mathcal{T})$ by the set of operators $\mathcal{O}$ we use to transform the base features. The operators $\mathcal{O}$, also known as d-ary operators, act on d base features to generate a new feature. For example, unary operators contain mathematical transformations, such as log, $sigmoid$, $square$, etc. Binary operators contain basic arithmetic operations, such as $+$, $-$, $\times$, $\div$, min, max, and SQL-based operations, such as $GroupByThenMean$, etc. Appendix C.2 contains a list of all the operators and how we use them to transform numerical and categorical features. Our method is extendable to all kinds of operators, and users can also define their own operators.

4 OpenFE

4.1 Overview

We present an overview of OpenFE in Figure 1. OpenFE follows the expand-and-reduce framework for automated feature generation. For expansion, first, we classify all the base features into numerical features and categorical features. Then we expand the candidate feature set by using operators to enumerate all the first-order transformation of base features (see Appendix C.2). For reduction, first we train a model on the dataset and yield base predictions for feature boosting (see Section 4.2). Next, we propose a two-stage evaluation framework to quickly reduce the number of candidate features. Finally, we include top-ranked candidate features into base feature set. We employ a greedy approach and repeat the above procedure for high-order feature generation.

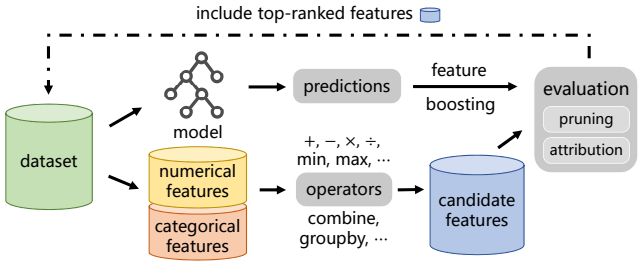

Figure 1: The overview of OpenFE.

A naive method for automated feature generation involves adding candidate features to the base feature set one by one and observing the change in validation loss after retraining the model. However, such method is computationally infeasible in many real-world applications when we have a large number of candidate features and samples. Therefore, the challenge lies in the trade-off between efficiency and accuracy in identifying useful features from candidate features. In the following sections, first we introduce a feature boosting method to accurately estimate the incremental performance of a new feature in an efficient way. Then we introduce a two-stage evaluation framework with successive featurewise pruning and feature importance attribution to quickly eliminate redundant candidate features and retrieve useful ones.

4.2 Feature Boosting

One of the key challenges in automated feature generation is how to accurately estimate the incremental performance of a new feature, i.e., how much performance improvement the new feature can offer when added to the base feature set. A standard evaluation procedure involves including the new feature to the base feature set, retraining the machine learning model, and observing the change in validation loss (Luo et al., 2019). In practical applications, however, the standard evaluation procedure is incredibly expensive due to the substantial computing resources required for training a model from scratch to convergence on the whole dataset. In this section, we introduce feature boosting to estimate the incremental performance of a new feature. There are two advantages of feature boosting. First, we use new features to boost upon the base predictions generated by base feature set, which converges much faster than training from scratch. Second, instead of training on the entire dataset, we can train on the new features alone to estimate their incremental performance.

Feature boosting is motivated by gradient boosting algorithms. Let $X \times Y$ denotes a dataset with corresponding targets. Let $\mathcal{F}$ denotes the set of decision trees defined over the base features $\mathcal{T}$ mapping $X \longrightarrow Y$. Let $lin(\mathcal{F})$ denotes the set of all linear combinations of decision trees in $\mathcal{F}$, and $C : lin(\mathcal{F}) \longrightarrow \mathbb{R}$ is the cost function on $lin(\mathcal{F})$. Gradient boosting proceeds iteratively via gradient descent to find a function $F \in lin(\mathcal{F})$ that minimizes $C(F)$. When gradient boosting procedure converges, the functions in $\mathcal{F}$ no longer point to the downhill direction of the cost function. At this time, if we introduce a new feature τ to the base feature set $\mathcal{T}$, we expand the function space from $\mathcal{F}$ to $\mathcal{F}_\tau$, where the new decision trees in $\mathcal{F}_\tau \backslash \mathcal{F}$ use the new feature τ and its interactions with the base features. When the new feature carries predictive power additional to the base feature set, the gradient boosting algorithm will proceed till new convergence and find $F_\tau \in lin(\mathcal{F}_\tau \backslash \mathcal{F})$ where $C(F_t + F_\tau) < C(F_t)$. The decrease in cost function $C(F_t) - C(F_t + F_\tau)$ quantifies the incremental performance of the new feature τ. The analysis above gives us the following motivation:

- Yield base predictions by training a machine learning model to convergence on the base feature set.
- For a new feature (or a group of new features), use gradient boosting algorithms to evaluate how much performance it (they) can boost upon the base predictions.

We call such method "feature boosting" that estimates the incremental performance of new features. Throughout this paper, we use gradient boosting decision trees (GBDT) to model tabular data. GBDT is usually the best performing model on tabular data where features are individually meaningful (Gorishniy et al., 2021; Borisov et al., 2021). In various kaggle competitions on tabular data, most of the winning solutions are based on gradient boosting decision trees (Bojer & Meldgaard, 2021). We use the popular LightGBM implementation (Ke et al., 2017). Even though the feature generation method relies on GBDT, the generated features can also enhance the learning performance of a variety of DNNs (see Section 5.4).

4.3 Evaluation Framework

In this section, we present the overall framework of OpenFE (see Algorithm 1). We start with generating the base predictions by training a GBDT with default parameters on the base features $\mathcal{T}$ for feature boosting (see implementation details in Appendix C.3). Then we enumerate all the first-order candidate features by the operators and base features. In order to deal with the large number of candidate features, we propose a two-stage evaluation framework. The first stage employs a successive featurewise pruning algorithm. By using a bandit-based algorithm to dynamically allocate

computing resources to promising features, we quickly eliminate redundant features by examining the effectiveness of each feature alone using feature boosting. In the second stage, we further takes interaction effects into account and propose a feature importance attribution algorithm to accredit the remaining features in reducing the cost function.

Algorithm 1: OpenFE

input : X: dataset
input : $\mathcal{T}$: base features, $\mathcal{O}$: operators
output: New feature set with generated features.

Initialize $order \leftarrow 1$;
while *order* $<$ *max_order* **do**
  Generate base predictions p by $\mathcal{T}$ and X using cross-validation;
  Enumerate candidate features $A(\mathcal{T})$ by $\mathcal{O}$ and $\mathcal{T}$;
  # two-stage evaluation
  $A^{'}(\mathcal{T})$ = SuccessivePruning$(A(\mathcal{T}), X, p)$;
  $A^{''}(\mathcal{T})$ = FeatureAttribution$(A^{'}(\mathcal{T}), X, p)$;
  $\mathcal{T} = \mathcal{T}$ + top_k$(A^{''}(\mathcal{T}))$;
  order = order + 1;
return $\mathcal{T}$

Algorithm 2: Successive Pruning

input : X: dataset, p: base predictions, $A(\mathcal{T})$: candidate feature set
output: Pruned new feature set.

Divide X equally into 2^q data blocks;
Initialize $A_0(\mathcal{T}) = A(\mathcal{T})$;
for $i \leftarrow 0$ **to** q **do**
  for *new feature* $\tau \in \mathcal{T}$ **do**
    Use 2^i data blocks to calculate the values of τ;
    Train a GBDT using τ with p as initial predictions;
    Evaluate the improvement in metrics on validation set;
  $A_i(\mathcal{T})$ = delete_same$(A_i(\mathcal{T}))$;
  $A_{i+1}(\mathcal{T})$ = top_half$(A_i(\mathcal{T}))$;
return $A_{q+1}(\mathcal{T})$

4.3.1 Successive Featurewise Pruning

In OpenFE, first we use a successive featurewise pruning method to quickly reduce the number of candidate features (see Algorithm 2). It is motivated by the successive halving algorithm in multi-armed bandit problems (Jamieson & Talwalkar, 2016). Successive halving dynamically allocates computing resources to promising arms. In our settings, we consider each new feature as an arm, and a pull of the arm is to train a GBDT using the single feature as the input with base predictions as the initial predictions (feature boosting) on the data blocks. The reward of pulling an arm is the improvement of the metrics over the base predictions on the validation set. The dataset X is split into 2^q data blocks. Each data block has both training set and validation set, with a total of $\lfloor \frac{N}{2^q} \rfloor$ samples.

Besides successive halving, we use a simple trick in our algorithm to quickly eliminate redundant new features. It is very common in the new feature set $A(\mathcal{T})$ where two features have exactly the same values. For example, $max(\tau_1, \tau_2)$ and $max(\tau_1, \tau_3)$ are exactly the same if the minimum value of feature τ_1 is larger than the maximum value of τ_2 and τ_3. Note that features with same values have the same scores of improvement in metrics. Therefore, the $delete_same$ algorithm first ranks all the new features by their scores, and then compares the scores of adjacent features to remove redundant ones.

4.3.2 Feature Importance Attribution

After pruning the candidate feature set by the effectiveness of each candidate feature alone in stage one, we need to further consider the interaction effects to evaluate the effectiveness of remaining new features. Assume the candidate feature set after pruning is $A^{'}(\mathcal{T})$. We use candidate features and base features to train a GBDT using feature boosting and obtain $F_t + F_{A'(\mathcal{T})}$. The decrease in cost function $C(F_t) - C(F_t + F_{A'(\mathcal{T})})$ indicates the incremental performance of the remaining candidate features $A^{'}(\mathcal{T})$. We need to attribute the contri-

Algorithm 3: Feature Attribution

input : X: dataset, p: base predictions, $\mathcal{T}$: base features, $A(\mathcal{T})$: candidate feature set
output: Sorted $A(\mathcal{T})$.

Train a GBDT using $\mathcal{T} + A(\mathcal{T})$ with p as initial predictions;
Attribute the reduction in loss to each feature in $A(\mathcal{T})$ using MDI;
Sort $A(\mathcal{T})$ according to MDI;
return sorted $A(\mathcal{T})$

bution of each candidate feature in decreasing
the cost function. Mean decrease in impurity (MDI) is a popular method for feature attribution in tree ensembles. MDI sums up the total reduction of loss in all splits for a given feature, which measures the contribution of a given feature in reducing the cost function. Two other popular methods for attribution are permutation and SHAP. Permutation measures the increase in cost function after we randomly permute the feature's values. SHAP is a game-theoretic method for feature attribution (Lundberg et al., 2018). We also compare MDI with permutation and SHAP in Section 5.6. We show that MDI is fast to compute and provides comparable performance with permutation and SHAP.

5 EXPERIMENTS

5.1 DATASETS AND EVALUATION METRICS

We use a diverse set of seven public datasets (see Appendix A for detailed descriptions) with two regression datasets, three binary-classification datasets, and two multi-classification datasets. Each dataset has exactly one train-validation-test split and all the methods use the same split. The datasets we collect include: California Housing (CA, real estate data, (Pace & Barry, 1997)), Microsoft (MI, search queries,), Diabetes (DI, hospital readmission, (Strack et al., 2014)), Nomao (NO, data deduplication, (Candillier & Lemaire, 2012)), Vehicle (VE), Jannis (JA, anonymized dataset, (Guyon et al., 2019)), Covertype (CO, forest characteristics, (Blackard & Dean, 1999)). We summarize the dataset properties in Table 1. The features in each dataset are classified into numerical, categorical, and ordinal features. An ordinal feature can be treated as both a categorical feature or a numerical feature when generating transformed features (see more details in Appendix C.2).

Table 1: Properties of datasets used in our experiments. Notation: "RMSE" $\sim$ root-mean-square error, "AUC" $\sim$ area-under-curve, "Acc." $\sim$ accuracy.

	CA	MI	DI	NO	VE	JA	CO
#objects	20640	1200192	101766	36645	98528	83733	581012
#num. features	7	111	3	34	100	54	9
#cat. features	0	0	34	29	0	0	0
#ord. features	1	25	10	55	0	0	45
metric	RMSE	RMSE.	AUC	AUC	AUC	Acc.	Acc.
#classes	–	–	2	2	2	4	7

5.2 BASELINE METHODS FOR COMPARISONS

We summarize the baseline methods in the following:

- **Base**. The base feature set without feature generation.
- **FCTree** (Fan et al., 2010). FCTree uses information gain to select new features during the construction of decision trees.
- **SAFE** (Shi et al., 2020). SAFE uses information value, Pearson correlation, and importance in XGBoost (Chen & Guestrin, 2016) to select new features.
- **AutoFeat** (Horn et al., 2019). AutoFeat is an open-source python package that uses improvement in a linear regression model to select new features.
- **AutoCross** (Luo et al., 2019). AutoCross employs beam search and field-wise logistic regression to evaluate feature set.

Most of existing automated feature engineering methods do not open-source their codes. We reproduce FCTree, SAFE, and AutoCross according to the descriptions in the paper. In order to make sure the reproduction is correct, we use the reproduced version to run the datasets in their paper and see if we can get similar results (see more details in Appendix B.2). Some other learning-based methods, such as LFE (Nargesian et al., 2017), ExploreKit (Katz et al., 2016) (meta learning) and NFS (Chen et al., 2019), TransGraph (Khurana et al., 2018) (reinforcement learning) lack critical details for reproduction. For example, the choice of training datasets is critical for generalization in

learning-based approaches, while the details are missing in previous studies Nargesian et al. (2017); Khurana et al. (2018). We compare with these methods on the test datasets they use in Appendix B.1.

Table 2: Comparisons between OpenFE and baseline methods with standard deviation. We report mean and standard deviations of metric values on test set over 10 random seeds. The top result for each dataset is marked in **bold**. SAFE and AutoCross can only run on binary-classification datasets.

Method	CA ↓	MI ↓	DI ↑	NO ↑	VE ↑	JA ↑	CO ↑
				LightGBM			
Base	$0.4323_{\pm 3.7e\text{-}3}$	$0.7439_{\pm 3.0e\text{-}4}$	$0.7313_{\pm 1.2e\text{-}3}$	$0.9958_{\pm 2.0e\text{-}4}$	$0.9250_{\pm 3.0e\text{-}4}$	$0.7212_{\pm 1.9e\text{-}3}$	$0.9685_{\pm 1.1e\text{-}3}$
FCTree	$0.4324_{\pm 2.7e\text{-}3}$	$0.7440_{\pm 1.0e\text{-}4}$	$0.7313_{\pm 1.0e\text{-}0}$	$0.9956_{\pm 2.0e\text{-}4}$	$0.9255_{\pm 2.0e\text{-}4}$	$0.7190_{\pm 1.3e\text{-}3}$	$0.9710_{\pm 7.0e\text{-}4}$
SAFE	–	–	$0.7295_{\pm 1.7e\text{-}3}$	$0.9958_{\pm 3.0e\text{-}4}$	$0.9250_{\pm 1.0e\text{-}3}$	–	–
AutoFeat	$0.4440_{\pm 2.5e\text{-}3}$	$0.7438_{\pm 2.0e\text{-}4}$	$0.7318_{\pm 1.1e\text{-}3}$	$0.9958_{\pm 2.0e\text{-}4}$	$0.9250_{\pm 3.0e\text{-}4}$	$0.7205_{\pm 7.0e\text{-}4}$	$0.9682_{\pm 1.1e\text{-}3}$
AutoCross	–	–	$0.7319_{\pm 1.1e\text{-}3}$	$0.9932_{\pm 3.0e\text{-}4}$	$0.9207_{\pm 3.0e\text{-}4}$	–	–
OpenFE	$\mathbf{0.4206}_{\pm \mathbf{1.6e\text{-}3}}$	$\mathbf{0.7382}_{\pm \mathbf{3.0e\text{-}4}}$	$\mathbf{0.8878}_{\pm \mathbf{2.5e\text{-}3}}$	$\mathbf{0.9967}_{\pm \mathbf{2.0e\text{-}4}}$	$\mathbf{0.9281}_{\pm \mathbf{4.0e\text{-}4}}$	$\mathbf{0.7286}_{\pm \mathbf{1.4e\text{-}3}}$	$\mathbf{0.9737}_{\pm \mathbf{5.0e\text{-}4}}$
				FT-Transformer			
Base	$0.4590_{\pm 3.3e\text{-}3}$	$0.7461_{\pm 4.0e\text{-}4}$	$0.7281_{\pm 2.4e\text{-}3}$	$0.9933_{\pm 5.0e\text{-}4}$	$0.9272_{\pm 6.0e\text{-}4}$	$0.7328_{\pm 2.3e\text{-}3}$	$0.9696_{\pm 8.0e\text{-}4}$
FCTree	$0.4570_{\pm 2.7e\text{-}3}$	$0.7461_{\pm 4.0e\text{-}4}$	$0.7282_{\pm 1.4e\text{-}3}$	$0.9930_{\pm 5.0e\text{-}4}$	$0.9273_{\pm 4.0e\text{-}4}$	$0.7298_{\pm 2.4e\text{-}3}$	$\mathbf{0.9713}_{\pm \mathbf{7.0e\text{-}4}}$
SAFE	–	–	$0.7281_{\pm 1.1e\text{-}3}$	$0.9933_{\pm 4.0e\text{-}4}$	$0.9276_{\pm 3.0e\text{-}3}$	–	–
AutoFeat	$0.4574_{\pm 3.0e\text{-}3}$	$0.7463_{\pm 3.0e\text{-}4}$	$0.7281_{\pm 2.6e\text{-}3}$	$0.9931_{\pm 4.0e\text{-}4}$	$0.9271_{\pm 5.0e\text{-}4}$	$0.7337_{\pm 1.2e\text{-}3}$	$0.9704_{\pm 7.0e\text{-}4}$
AutoCross	–	–	$0.7292_{\pm 9.0e\text{-}4}$	$0.9905_{\pm 3.0e\text{-}4}$	$0.9205_{\pm 3.0e\text{-}4}$	–	–
OpenFE	$\mathbf{0.4530}_{\pm \mathbf{2.7e\text{-}3}}$	$\mathbf{0.7411}_{\pm \mathbf{7.0e\text{-}4}}$	$\mathbf{0.8870}_{\pm \mathbf{1.2e\text{-}3}}$	$\mathbf{0.9950}_{\pm \mathbf{3.0e\text{-}4}}$	$\mathbf{0.9289}_{\pm \mathbf{3.0e\text{-}4}}$	$\mathbf{0.7379}_{\pm \mathbf{2.3e\text{-}3}}$	$0.9708_{\pm 7.0e\text{-}4}$

5.3 Comparison Results

In this section, we compare OpenFE with other baseline methods. We need a learning algorithm to fit the data and evaluate the effectiveness of the new features generated by different methods. For GBDT, we choose LightGBM implementation Ke et al. (2017). While different GBDT implementations (such as LightGBM, XGBoost (Chen & Guestrin, 2016), and Catboost (Prokhorenkova et al., 2018)) vary in details, their performance do not differ much on most of the tasks (Prokhorenkova et al., 2018). For neural networks, we choose FT-Transformer (Gorishniy et al., 2021). FT-Transformer is an adaptation of Transformer with feature tokenizers. We choose this model because it was benchmarked in a convincing way to be considered a "best case" so far for deep learning models on tabular data (Gorishniy et al., 2021).

We present the comparison results in Table 2. For each dataset, we generate first-order new features and include the same number of new features for different methods for a fair comparison (see details in Appendix C). We can observe that OpenFE has clear advantages over baseline methods and achieve SOTA on most of the cases. The features generated by OpenFE greatly enhance the performance of both LightGBM and FT-Transformer. One interesting observation is that OpenFE narrows the gap between LightGBM and FT-Transformer on six out of seven datasets. Another interesting observation is that, for baseline methods that fail to recall useful new features, including uninformative features into base feature set brings additional noise and does harm to the generalization on the test set.

5.4 Feature Generation for Neural Networks

In this section, we show that new feature generated by OpenFE can greatly improve the performance of a variety of neural networks designed specifically for tabular data. The models include:

- **AutoInt** (Song et al., 2019). An attention-based network that captures feature interactions.
- **DCN V2** (Wang et al., 2021). Consists of an MLP-like module and feature crossing module to capture feature interactions.
- **NODE** (Popov et al., 2019). A differentiable ensemble of oblivious decision trees.
- **TabNet** (Arık & Pfister, 2021). Uses sequential attention to choose which features to reason from at each decision step.

- **FT-Transformer** (Gorishniy et al., 2021). An adaptation of Transformer with feature tokenizers.

We follow the same implementations and hyperparameter tuning of these neural networks in (Gorishniy et al., 2021). We present the results in Table 3. The features generated by OpenFE greatly enhance the performance of different models on most of the cases. For AutoInt and DCN V2 that claim to be able to learn feature interactions, feature generation can also improve the performance. Even though OpenFE relies on GBDT to measure the incremental performance of new features, the generated features are effective for a variety of DNNs.

Table 3: Results for OpenFE with standard deviations. We report mean and standard deviations of metric values on test set over 10 random seeds. Depending on the dataset, the highest accuracy or the lowest RMSE is in **bold**. Notation: $\downarrow$ RMSE, $\uparrow$ accuracy

Model	Features	CA $\downarrow$	MI $\downarrow$	NO $\uparrow$	VE $\uparrow$	JA $\uparrow$	CO $\uparrow$
AutoInt	Original	$0.4741_{\pm 3.7e\text{-}3}$	$0.7501_{\pm 5.0e\text{-}4}$	$0.9927_{\pm 3.0e\text{-}4}$	$0.9270_{\pm 9.0e\text{-}4}$	$0.7207_{\pm 2.3e\text{-}3}$	$0.9342_{\pm 4.2e\text{-}3}$
	OpenFE	$0.4618_{\pm 5.1e\text{-}3}$	$0.7447_{\pm 4.0e\text{-}4}$	$0.9943_{\pm 4.0e\text{-}4}$	$0.9287_{\pm 7.0e\text{-}4}$	$0.7243_{\pm 2.7e\text{-}3}$	$0.9415_{\pm 4.1e\text{-}3}$
DCN-V2	Original	$0.4832_{\pm 2.1e\text{-}3}$	$0.7486_{\pm 7.0e\text{-}4}$	$0.9926_{\pm 4.0e\text{-}4}$	$0.9236_{\pm 1.5e\text{-}3}$	$0.7158_{\pm 1.6e\text{-}3}$	$0.9653_{\pm 1.3e\text{-}3}$
	OpenFE	$0.4722_{\pm 4.0e\text{-}3}$	$0.7431_{\pm 4.0e\text{-}4}$	$0.9944_{\pm 1.0e\text{-}4}$	$0.9260_{\pm 4.0e\text{-}4}$	$0.7163_{\pm 1.6e\text{-}3}$	$0.9633_{\pm 7.0e\text{-}4}$
TabNet	Original	$0.5096_{\pm 4.9e\text{-}3}$	$0.7508_{\pm 1.1e\text{-}3}$	$0.9932_{\pm 3.0e\text{-}4}$	$0.9236_{\pm 1.0e\text{-}3}$	$0.7237_{\pm 4.3e\text{-}3}$	$0.9594_{\pm 4.3e\text{-}3}$
	OpenFE	$0.5011_{\pm 9.0e\text{-}3}$	$0.7464_{\pm 8.0e\text{-}4}$	$0.9948_{\pm 2.0e\text{-}4}$	$0.9258_{\pm 7.0e\text{-}4}$	$0.7237_{\pm 4.3e\text{-}3}$	$0.9592_{\pm 1.4e\text{-}3}$
NODE	Original	$0.4639_{\pm 1.6e\text{-}3}$	$0.7454_{\pm 3.0e\text{-}4}$	$0.9937_{\pm 4.0e\text{-}4}$	$0.9265_{\pm 3.0e\text{-}4}$	$0.7271_{\pm 1.8e\text{-}3}$	$0.9589_{\pm 1.2e\text{-}3}$
	OpenFE	$0.4569_{\pm 1.8e\text{-}3}$	$0.7395_{\pm 2.0e\text{-}4}$	$0.9953_{\pm 5.0e\text{-}4}$	$0.9289_{\pm 4.0e\text{-}4}$	$0.7312_{\pm 1.1e\text{-}3}$	$0.9632_{\pm 9.0e\text{-}4}$
FT-Transformer	Original	$0.4590_{\pm 3.3e\text{-}3}$	$0.7461_{\pm 4.0e\text{-}4}$	$0.9933_{\pm 5.0e\text{-}4}$	$0.9272_{\pm 6.0e\text{-}4}$	$0.7328_{\pm 2.3e\text{-}3}$	$0.9696_{\pm 8.0e\text{-}4}$
	OpenFE	$0.4530_{\pm 2.7e\text{-}3}$	$0.7411_{\pm 7.0e\text{-}4}$	$0.9950_{\pm 3.0e\text{-}4}$	$0.9289_{\pm 3.0e\text{-}4}$	$0.7379_{\pm 2.3e\text{-}3}$	$0.9708_{\pm 7.0e\text{-}4}$

Table 4: Results of OpenFE and ExpertFE (feature generation by experts) in two Kaggle competitions. Notation: pub. $\sim$ public, pri. $\sim$ private. The final standing is determined according to scores in private leaderboard.

		IEEE-CIS Fraud Detection				BNP Paribas Cardif Claims Management			
feature	order	pub. score	pub. rank	pri. score	pri. rank	pub. score	pub. rank	pri. score	pri. rank
Base	-	$0.9463_{\pm 3.7e\text{-}4}$	2408/6351	$0.9182_{\pm 3.2e\text{-}4}$	2286/6351	$0.4382_{\pm 1.7e\text{-}4}$	38/2922	$0.4359_{\pm 2.0e\text{-}4}$	31/2922
ExpertFE	first	$0.9602_{\pm 2.6e\text{-}4}$	52/6351	$0.9327_{\pm 3.0e\text{-}4}$	70/6351	$0.4334_{\pm 1.7e\text{-}4}$	17/2922	$0.4308_{\pm 1.0e\text{-}4}$	12/2922
	high	$0.9597_{\pm 3.1e\text{-}4}$	54/6351	$0.9320_{\pm 3.3e\text{-}4}$	76/6351	$0.4322_{\pm 1.5e\text{-}4}$	13/2922	$0.4301_{\pm 2.2e\text{-}4}$	12/2922
OpenFE	first	$0.9617_{\pm 3.5e\text{-}4}$	38/6351	$0.9360_{\pm 4.0e\text{-}4}$	44/6351	$0.4349_{\pm 3.9e\text{-}4}$	22/2922	$0.4324_{\pm 4.3e\text{-}4}$	16/2922
	high	$0.9617_{\pm 3.6e\text{-}4}$	38/6351	$0.9363_{\pm 2.8e\text{-}4}$	42/6351	$0.4327_{\pm 3.4e\text{-}4}$	14/2922	$0.4304_{\pm 3.0e\text{-}4}$	12/2922

5.5 Case Studies: Comparing OpenFE with Machine Learning Experts

The goal of automated feature generation is to liberate machine learning experts from tedious manual feature generation. In this section, we present two case studies with two Kaggle competitions to show that the feature generated by OpenFE can result in comparable or even larger performance improvement than those provided by the competition's top winners.

The first Kaggle competition is IEEE-CIS Fraud Detection[1], where the goal is to predict whether an online transaction is fraudulent. This competition is one of the largest and most competitive tabular data competitions in Kaggle, with 6,351 data science teams participating. The competition's first place winners made public their magic of feature generation after the competition ended[2], which we refer to as ExpertFE. This competition heavily relies on feature generation, where a baseline model of XGBoost without feature generation ranks at 2286 among 6351 teams in the private leaderboard.

[1] https://www.kaggle.com/competitions/ieee-fraud-detection/overview
[2] https://www.kaggle.com/code/cdeotte/xgb-fraud-with-magic-0-9600

However, the baseline model with features generated by the winners ranks at 76/6351. Moreover, the baseline model with feature generated by OpenFE ranks at 42/6351, which beats 99.3% teams in the competition. We present the results in Table 4. OpenFE generates 600 first-order features and 10 second-order features. ExpertFE generates 68 first-order features and 6 high-order features. High-order features do not seem to be very useful in this competition. We investigate the influence on the number of new first-order features in Figure 2. The advantage of OpenFE is to automatically explore and generate more useful feature transformations than ExpertFE to improve learning performance.

The second competition is BNP Paribas Cardif Claims Management[3], where the goal is to predict whether we should make a claim in personal insurance. The competition's 8th place winners made public their generated features after the competition ended[4] (the winners with higher rankings did not share their codes). We evaluate the performance of OpenFE in a similar way. A baseline model of Catboost (Prokhorenkova et al., 2018) without feature generation ranks at 31 among 2920 teams. The baseline model with features generated by experts ranks at 14/2920, while the baseline model with features generated by OpenFE ranks at 12/2920. We present the results in Table 4. OpenFE generates 200 first-order features and 150 second-order features, while ExpertFE generates 200 first-order features and 50 high-order features.

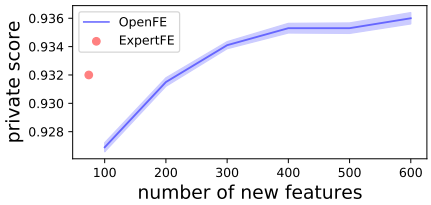

Figure 2: Comparison between ExpertFE and OpenFE.

Table 5: Results for ablation study. The model is LightGBM.

	CA ↓	MI ↓	DI ↑	NO ↑	VE ↑	JA ↑	CO ↑
OpenFE	$0.4206_{\pm 1.6e\text{-}3}$	$0.7382_{\pm 3.0e\text{-}4}$	$0.8878_{\pm 2.5e\text{-}3}$	$0.9967_{\pm 2.0e\text{-}4}$	$0.9281_{\pm 4.0e\text{-}4}$	$0.7286_{\pm 1.4e\text{-}3}$	$0.9737_{\pm 5.0e\text{-}4}$
OpenFE\FB	$0.4299_{\pm 2.2e\text{-}3}$	$0.7441_{\pm 4.0e\text{-}3}$	$0.8856_{\pm 3.3e\text{-}3}$	$0.9959_{\pm 2.0e\text{-}4}$	$0.9263_{\pm 2.0e\text{-}4}$	$0.7251_{\pm 1.2e\text{-}3}$	$0.9704_{\pm 7.0e\text{-}4}$
OpenFE\SS	$0.4282_{\pm 2.3e\text{-}3}$	$0.7439_{\pm 5.0e\text{-}4}$	$0.8846_{\pm 2.2e\text{-}3}$	$0.9952_{\pm 3.0e\text{-}4}$	$0.9267_{\pm 3.0e\text{-}4}$	$0.7316_{\pm 1.1e\text{-}3}$	$0.9649_{\pm 7.0e\text{-}4}$
OpenFE-MI	$0.4282_{\pm 2.3e\text{-}3}$	$0.7439_{\pm 3.0e\text{-}4}$	$0.8867_{\pm 4.4e\text{-}3}$	$0.9959_{\pm 1.0e\text{-}4}$	$0.9261_{\pm 3.0e\text{-}4}$	$0.7293_{\pm 1.2e\text{-}3}$	$0.9713_{\pm 5.0e\text{-}4}$

5.6 Ablation Study

We conduct an ablation study to justify the design choices of OpenFE. We name different variants of OpenFE as follows:

- OpenFE\FB. OpenFE without feature boosting and train GBDT from scratch.
- OpenFE\SS. OpenFE without successive featurewise pruning. We subsample the data so that generating all the features can fit in the memory.
- OpenFE-MI. We use mutual information between the feature and the target as the criterion in successive featurewise pruning.

We present the results in Table 5. The results show that: 1) Feature boosting significantly improve the results by estimating the incremental performance of candidate features. 2) Directly subsampling the data usually hurts the performance. 3) Mutual information does not provide desirable results in eliminating features.

6 Conclusion

We propose OpenFE, an automated feature generation tool beyond expert-level performance. In addition to proposing a strong method, our contributions include: 1) building a benchmark on a diverse set of tasks and review main methods for automated feature generation. 2) performing extensive experiments to show that feature generation can greatly improve the learning performance of existing DNN architectures.

[3] https://www.kaggle.com/competitions/bnp-paribas-cardif-claims-management/
[4] https://www.kaggle.com/code/confirm/xfeat-cudf-lightgbm-catboost-wip

REFERENCES

Sercan O Arık and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *AAAI*, volume 35, pp. 6679–6687, 2021.

Jock A Blackard and Denis J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 24(3):131–151, 1999.

Casper Solheim Bojer and Jens Peder Meldgaard. Kaggle forecasting competitions: An overlooked learning opportunity. *International Journal of Forecasting*, 37(2):587–603, 2021.

Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *arXiv preprint arXiv:2110.01889*, 2021.

Laurent Candillier and Vincent Lemaire. Design and analysis of the nomao challenge active learning in the real-world. In *Proceedings of the ALRA: Active Learning in Real-world Applications, Workshop ECML-PKDD*. Citeseer, 2012.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Xiangning Chen, Qingwei Lin, Chuan Luo, Xudong Li, Hongyu Zhang, Yong Xu, Yingnong Dang, Kaixin Sui, Xu Zhang, Bo Qiao, et al. Neural feature search: A neural architecture for automated feature engineering. In *2019 IEEE International Conference on Data Mining (ICDM)*, pp. 71–80. IEEE, 2019.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. IEEE, 2009.

Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

Wei Fan, Erheng Zhong, Jing Peng, Olivier Verscheure, Kun Zhang, Jiangtao Ren, Rong Yan, and Qiang Yang. Generalized and heuristic-free feature construction for improved accuracy. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pp. 629–640. SIAM, 2010.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34, 2021.

Isabelle Guyon, Lisheng Sun-Hosoya, Marc Boullé, Hugo Jair Escalante, Sergio Escalera, Zhengying Liu, Damir Jajetic, Bisakha Ray, Mehreen Saeed, Michèle Sebag, et al. Analysis of the automl challenge series. *Automated Machine Learning*, pp. 177, 2019.

Franziska Horn, Robert Pack, and Michael Rieger. The autofeat python library for automated feature engineering and selection. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 111–120. Springer, 2019.

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial intelligence and statistics*, pp. 240–248. PMLR, 2016.

James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*, pp. 1–10. IEEE, 2015.

Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Explorekit: Automatic feature generation and selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 979–984. IEEE, 2016.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

Udayan Khurana, Horst Samulowitz, and Deepak Turaga. Feature engineering for predictive modeling using reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Hoang Thanh Lam, Beat Buesser, Hong Min, Tran Ngoc Minh, Martin Wistuba, Udayan Khurana, Gregory Bramble, Theodoros Salonidis, Dakuo Wang, and Horst Samulowitz. Automated data science for relational data. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 2689–2692. IEEE, 2021.

Adeline Lo, Herman Chernoff, Tian Zheng, and Shaw-Hwa Lo. Why significant variables aren't automatically good predictors. *Proceedings of the National Academy of Sciences*, 112(45):13892–13897, 2015.

Scott M Lundberg, Gabriel G Erion, and Su-In Lee. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888*, 2018.

Yuanfei Luo, Mengshuo Wang, Hao Zhou, Quanming Yao, Wei-Wei Tu, Yuqiang Chen, Wenyuan Dai, and Qiang 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*, pp. 1936–1945, 2019.

Fatemeh Nargesian, Horst Samulowitz, Udayan Khurana, Elias B Khalil, and Deepak S Turaga. Learning feature engineering for classification. In *Ijcai*, pp. 2529–2535, 2017.

R Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33 (3):291–297, 1997.

Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. *arXiv preprint arXiv:1909.06312*, 2019.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. *Advances in neural information processing systems*, 31, 2018.

Qitao Shi, Ya-Lin Zhang, Longfei Li, Xinxing Yang, Meng Li, and Jun Zhou. Safe: Scalable automatic feature engineering framework for industrial tasks. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pp. 1645–1656. IEEE, 2020.

Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 1161–1170, 2019.

Beata Strack, Jonathan P DeShazo, Chris Gennings, Juan L Olmo, Sebastian Ventura, Krzysztof J Cios, and John N Clore. Impact of hba1c measurement on hospital readmission rates: analysis of 70,000 clinical database patient records. *BioMed research international*, 2014, 2014.

Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the Web Conference 2021*, pp. 1785–1797, 2021.

Michael D Ward, Brian D Greenhill, and Kristin M Bakke. The perils of policy by p-value: Predicting civil conflicts. *Journal of peace research*, 47(4):363–375, 2010.

Ivo Welch and Amit Goyal. A comprehensive look at the empirical performance of equity premium prediction. *The Review of Financial Studies*, 21(4):1455–1508, 2008.

Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.

A DATA

Table 6: Datasets description

Name	Abbr	# Train	# Validation	# Test	# Num	# Cat	# Ord	Task type
California Housing	CA	13209	3303	4128	7	0	1	Regression
Microsoft	MI	723412	235259	241521	111	0	25	Regression
Diabetes	DI	65129	16283	20354	3	34	10	Binclass
Nomao	NO	22465	6000	6000	34	29	55	Binclass
Vehicle	VE	60000	18528	20000	100	0	0	Binclass
Jannis	JA	53588	13398	16747	54	0	0	Multiclass
Covertype	CO	371847	92962	116203	9	0	45	Multiclass

All the datasets can be found in the supplementary materials.

B ADDITIONAL RESULTS

Table 7: Comparison between OpenFE and other baselines. The results of baseline methods are from Table I of NFS (Chen et al., 2019). Our results are averaged by 10 different random seeds.

Dataset	Source	C\ R	Instances\Features	Random	TransGraph	LFE	NFS	OpenFE
Airfoil	UCIrvine	R	1503\5	0.753	0.801	-	0.796	**0.808**
German Credit	UCIrvine	C	1000\24	0.655	0.724	-	0.805	**0.815**
Higgs Boson Subset	UCIrvine	C	50000\28	0.699	0.729	0.68	0.731	**0.741**
Ionosphere	UCIrvine	C	351\34	0.934	0.941	0.932	0.969	**0.986**
SpamBase	UCIrvine	C	4601\57	0.937	**0.961**	0.947	0.955	**0.961**
SpectF	UCIrvine	C	467\44	0.748	0.788	-	0.876	**0.877**
Sonar	UCIrvine	C	208\60	0.723	-	0.801	0.839	**0.929**

B.1 COMPARISONS WITH OTHER BASELINES

We compare OpenFE with other baselines, including some learning-based methods that lack critical details for code reproduction. These mehtods include:

- **Random** (Khurana et al., 2018). Randomly include features from candidate feature set multiple times and select new features with improvement according to CV scores.
- **TransGraph** (Khurana et al., 2018). TransGraph uses reinforcement learning to traverse a transformation graph for feature transformations.
- **LFE** (Nargesian et al., 2017). LFE recommends feature transformations by meta learning approaches.
- **NFS** (Chen et al., 2019). NFS uses a recurrent neural network controller to search for a series of transformations.

Following previous studies (Chen et al., 2019), the metric for regression datasets is $1 -$ (relative absolute error) and the metric for classification datasets is F1-score. We present the results in Table 7. OpenFE also surpasses other baseline methods in these datasets.

B.2 REPRODUCTION

We reproduce FCTree, SAFE, and AutoCross according to the descriptions in their papers. In order to make sure that we reproduce a reasonable version of these methods, we compare the results of our

Table 8: Comparisons between the results of reproduced methods and results from the paper. The results are averaged by 10 different random seeds.

(a) Comparison between FCTree (from paper) and FCTree (reproduced).

	Metric	FCTree (from paper)	FCTree (reproduced)
Transfusion	Accuracy	0.752	0.793
Nuclear(All)	AUC	0.629	0.625

(b) Comparison between SAFE (from paper) and SAFE (reproduced).

	Metric	SAFE (from paper)	SAFE (reproduced)
Magic	AUC	0.9288	0.9370
Spambase	AUC	0.9846	0.9837

(c) Comparison between AutoCross (from paper) and AutoCross (reproduced).

	Metric	AutoCross (from paper)	AutoCross (reproduced)
Bank	AUC	0.9455	0.9456
Adult	AUC	0.9280	0.9251
Credit	AUC	0.8567	0.8624

Table 9: Comparisons between MDI, permutation, and SHAP in feature importance attribution.

	MDI	permutation	SHAP
Rank	2.07	1.79	2.14
Time	0s	25min	42s

reproduced methods with the ones in their paper. We present the results in Table 8. We can see that most of the results of the reproduced methods closely match or even outperform the results from the papers.

B.3 Comparing Feature Importance Attribution Methods

In this section, we compare MDI, permutation, and SHAP in feature importance attribution in OpenFE. We use the results of OpenFE on seven benchmarking datasets to rank these methods. The training model is LightGBM. We present the results in Table 9. We also present the running time of each method on Microsoft, the benchmarking dataset with largest number of samples. Different feature attribution methods do not differ much on most of the datasets. MDI can be obtained for free from LightGBM, while permutation and SHAP may require much more running time, depending on the scale of datasets.

C Implementation Details

C.1 Experimental Environment

For all the experiments, feature generation is carried out on a workstation with Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz, 40 cores, 512G memory. Model tuning and model training is performed on one or more NVidia Tesla V100 16Gb.

Table 10: Unary operators.

Numerical	Categorical
Freq, Abs, Log, Sqrt, Sigmoid, Round, Residual	Freq

Table 11: Binary operators. (N refers to Numerical, C refers to Categorical)

N × N	N × C	C × C
Min, Max, +, −, ×, ÷	GroupByThenMin, GroupByThenMax, GroupByThenMean, GroupByThenMedian, GroupByThenStd, GroupByThenRank	Combine, CombineThenFreq, GroupByThenNUnique

C.2 Operators and Feature Transformations

All the operators are classified into unary operators (Table 10) and binary operators (Table 11), where unary operators function on one feature and binary operators function on two features. Then the operators are further classified according to the type of features they function on. For example, GroupByThenMean requires a categorical feature and a numerical feature, while Max requires two numerical features. All the features in a dataset are classified into numerical features, categorical features, and ordinal features. The difference between ordinal features and categorical features is that an ordinal feature has a clear ordering of categories (such as "age"). Ordinal features are treated as both numerical and categorical when generating features transformation. For example, we can calculate GroupByThenMean(age, gender), which is the average age of each gender. We can also calculate GroupByThenMean(income, age), which is the average income of people in different ages. For anonymized datasets where the meanings of features are unknown, features with string values are treated as categorical features. Features with discrete values (the number of unique values is less than 100) are treated as ordinal features. Features with continuous values are treated as numerical features.

We list the details of all the operators below. f stands for a numerical feature and c stands for a categorical feature.

- $\text{Freq}(f)$. The frequency of feature f.
- $\text{Freq}(c)$. The frequency of feature c.
- $\text{Abs}(f)$. Element-wise absolute value.
- $\text{Log}(f)$. Element-wise logarithm.
- $\text{Sqrt}(f)$. Element-wise square root.
- $\text{Sigmoid}(f)$. Element-wisely apply function $x \mapsto \frac{1}{1+e^{-x}}$.
- $\text{Round}(f)$. Element-wise rounding.
- $\text{Residual}(f)$. Element-wisely take decimal part.
- $\text{Min}(f_1, f_2)$. Element-wise minimum.
- $\text{Max}(f_1, f_2)$. Element-wise maximum.
- $f_1 + f_2$. Element-wise addition.
- $f_1 - f_2$. Element-wise subtraction.
- $f_1 \times f_2$. Element-wise multiplication.
- $f_1 \div f_2$. Element-wise division.
- $\text{GroupByThenMin}(f, c)$. The minimum value of f in each category of feature c.
- $\text{GroupByThenMax}(f, c)$. The maximum value of f in each category of feature c.

Table 12: The parameters of OpenFE for each dataset.

	CA	MI	DI	NO	VE	JA	CO
# data blocks	1	64	8	32	16	1	8
# new features	10	10	10	10	10	50	50

- $\mathrm{GroupByThenMean}(f, c)$. The average value of f in each category of feature c.
- $\mathrm{GroupByThenMedian}(f, c)$. The median of f in each category of feature c.
- $\mathrm{GroupByThenStd}(f, c)$. The standard deviation of f in each category of feature c.
- $\mathrm{GroupByThenRank}(f, c)$. The ranking of f in each category of feature c. The rankings are normalized between 0 and 1.
- $\mathrm{Combine}(c_1, c_2)$. Comebine the categories in feature c_1 and c_2 to be new categories. For example, for a sample, "vocation" is "doctor" and "hobby" is "football", then the value for $\mathrm{Combine}(\text{vocation}, \text{hobby})$ of this sample is a category of "doctor $-$ football".
- $\mathrm{CombineThenFreq}(c_1, c_2)$. Same as $\mathrm{freq}\left(\mathrm{Combine}(c_1, c_2)\right)$.
- $\mathrm{GroupByThenNUnique}(c_1, c_2)$. The number of unique values of c_1 in each category of feature c_2.

C.3 Generate Base Predictions

We use 5-fold cross-validation to generate base predictions. For each fold, we first train a LightGBM on the train set until early stopping on the validation set. Then we generate prediction on the validation set. We concatenate the predictions on each fold to yield base predictions. The LightGBM is trained using a set of default parameters for all the datasets. The default parameter has 10000 estimators, 0.1 learning rate, 31 leaves, and 200 early stopping rounds. Other parameters follow the default settings in LightGBM.

C.4 Parameters

We present the hyperparameters of OpenFE for each dataset in Table 12. For datasets with few samples or the number of candidate new features is small, the number of data blocks for successive featurewise pruning is set to be small. When the number of data blocks is one, we use all the data to calculate and evaluate new features in successive featurewise pruning, and drop new features with negative scores. For most of the datasets, the number of new features is set to be 10. The effective new features are usually sparse in the vast pool of candidate new features. For two multi-classification datasets, the number of new features is set to be 50. All the baseline methods include the same number of new features for a fair comparison.