OpenFE: Automated Feature Generation with Expert-level Performance

Tianping Zhang [1] **Zheyu Zhang** [1] **Zhiyuan Fan** [1] **Haoyan Luo** [2] **Fengyuan Liu** [3] **Qian Liu** [4] **Wei Cao** [5] **Jian Li** [1]

Abstract

The goal of automated feature generation is to liberate machine learning experts from the laborious task of manual feature generation, which is crucial for improving the learning performance of tabular data. The major challenge in automated feature generation is to efficiently and accurately identify effective features from a vast pool of candidate features. In this paper, we present OpenFE, an automated feature generation tool that provides competitive results against machine learning experts. OpenFE achieves high efficiency and accuracy with two components: 1) a novel feature boosting method for accurately evaluating the incremental performance of candidate features and 2) a two-stage pruning algorithm that performs feature pruning in a coarse-to-fine manner. Extensive experiments on ten benchmark datasets show that OpenFE outperforms existing baseline methods by a large margin. We further evaluate OpenFE in two Kaggle competitions with thousands of data science teams participating. In the two competitions, features generated by OpenFE with a simple baseline model can beat 99.3% and 99.6% data science teams respectively. In addition to the empirical results, we provide a theoretical perspective to show that feature generation can be beneficial in a simple yet representative setting.

1. Introduction

Feature generation is an important yet challenging task when applying machine learning methods to tabular data. It has been well recognized that the quality of features has a significant impact on the learning performance of tabular data (Domingos, 2012). The goal of feature generation is to transform the base features into more informative ones to better describe the data and enhance the learning performance. Figure 1 demonstrates an example tabular data from Strack et al. (2014), where the goal is to predict whether a patient will be readmitted to the hospital. In the example, we can transform the base features "Age" and "# Diagnose" via "GroupByThenMean(Age, # Diagnose)", which calculates the mean number of diagnoses for each age group, and may be used to better characterize the age group and enhance the learning performance. 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.

Race	Age	# Medications	...	# Diagnose	Readmitted
Caucasian	[0-10)	1.0	…	1.0	1
Caucasian	[10-20)	18.0	…	9.0	0
African	[20-30)	13.0	…	6.0	0
…	…	…	…	…	…
American	[10-20)	16.0	…	5.0	1

Figure 1: An example tabular data from Strack et al. (2014). The blue columns indicate the base features. The red (last) column is the prediction target.

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 (Erickson et al., 2020; Lu, 2019). Expand-and-reduce is arguably the most prevalent framework in automated feature generation, in which we first create a pool of candidate features by expanding the base features and then eliminate the ineffective candidate features (Kanter & Veeramachaneni, 2015; Lam et al., 2021; Kaul et al., 2017; Shi et al., 2020; Katz et al., 2016). There are two challenges in a typical expand-and-reduce practice. The first challenge is to efficiently and accurately evaluate 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 standard evaluation procedure involves including the new feature in the base feature set, retraining the machine learning model, and observing the

[1]Institute for Interdisciplinary Information Sciences (IIIS), Tsinghua University, Beijing, China [2]School of Data Science, The Chinese University of Hong Kong (Shenzhen), Shenzhen, China [3]Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, U.S. [4]Sea AI Lab, Singapore [5]Microsoft Research Asia, Beijing, China. Correspondence to: Jian Li <lijian83@mail.tsinghua.edu.cn>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

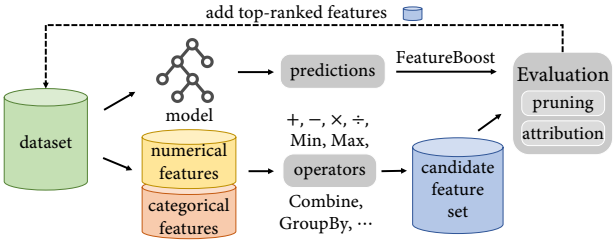

Figure 2: The overview of OpenFE.

change in the validation loss (Katz et al., 2016). However, retraining the model is usually time-consuming, making the standard evaluation procedure prohibitively expensive for large datasets. The second challenge is to calculate and evaluate the huge number of candidate features. There may be millions of candidate features for a dataset containing hundreds of features. Even with an efficient evaluation algorithm, computing all candidate features on the dataset is computationally costly and often impractical due to the enormous amount of memory required.

This paper presents OpenFE, which addresses the two challenges by a feature boosting algorithm, and a two-stage pruning algorithm. Regarding the first challenge, we argue that model retraining is not required for accurately evaluating the incremental performance of new features. Motivated by gradient boosting, we propose FeatureBoost, an efficient algorithm for evaluating the incremental performance of new features. FeatureBoost performs incremental training on top of the predictions yielded by the base feature set, which is substantially more efficient than retraining the model. For the second challenge, we propose a two-stage pruning algorithm to efficiently retrieve effective features from the vast pool of candidate features. Due to the fact that the effective features are usually sparse, the two-stage pruning algorithm performs feature pruning in a coarse-to-fine manner. We validate OpenFE on various datasets and Kaggle competitions, where OpenFE outperforms existing baseline methods by a large margin. In two famous Kaggle competition with thousands of data science teams participating (Kaggle, 2019; 2016), the baseline model with features generated by OpenFE beats 99.3% and 99.6% data science teams. More importantly, the features generated by OpenFE result in comparable or even greater performance improvement than those provided by the competition's first-place team, demonstrating that OpenFE is competitive against machine learning experts in feature generation.

In addition to proposing a novel method, this paper intends to address two important problems that hinder the research process of automated feature generation. The first problem is a lack of fair comparisons of existing methods. The majority of existing methods do not open-source their codes, and some previous studies evaluate their methods without using a held-out test set (Chen et al., 2019; Zhu et al., 2022; Li et al., 2023). We reproduce most existing methods and perform a large-scale benchmarking on existing methods to facilitate fair comparisons in future research. The second problem is the lack of evidence regarding the necessity of feature generation in the era of deep learning. In recent years, a variety of deep neural networks (DNNs) have been developed for modeling tabular data (Arık & Pfister, 2021; Gorishniy et al., 2021), and several of them have demonstrated their efficiency in feature interaction learning (Song et al., 2019; Wang et al., 2021). We conduct a large-scale empirical study to demonstrate that the generated features can further enhance the learning performance of several state-of-the-art DNNs. In addition to the empirical results, we provide a theoretical justification of our feature generation procedure by presenting a simple yet representative transductive learning setting in which feature generation can bring benefit to the learning algorithm provably. We summarize the contributions of our paper as follows:

- We propose OpenFE, a novel automated feature generation method that can efficiently and accurately identify useful new features to enhance learning performance. OpenFE achieves both efficiency and accuracy with a feature boosting method and a two-stage pruning algorithm.
- Extensive experiments show that OpenFE outperforms baselines on 10 benchmark datasets by a large margin. More importantly, the case study on two Kaggle competitions demonstrates that OpenFE can compete in feature generation against human experts.
- We present a theoretical model to show that the model using the generated features can be statistically more advantageous than that using only the base features.
- We facilitate future research by reproducing existing methods and performing a large-scale benchmarking on existing methods. The codes and datasets are available at `https://github.com/IIIS-Li-Group/OpenFE`

2. Problem Definition

For a given training dataset $\mathcal{D}$, we split it into a sub-training set $\mathcal{D}_{tr}$ and a validation set $\mathcal{D}_{vld}$. Assume $\mathcal{D}$ consists of 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 $\mathcal{L}(\mathcal{D}_{tr}, \mathcal{T}+\mathcal{S})$, and compute the evaluation metric $\mathcal{E}(\mathcal{L}(\mathcal{D}_{tr}, \mathcal{T}+\mathcal{S}), \mathcal{D}_{vld}, \mathcal{T}+\mathcal{S})$ to measure the model performance, with a larger value indicating better performance. Formally, the feature generation problem is:

$$\max_{\mathcal{S}\subseteq A(\mathcal{T})} \mathcal{E}(\mathcal{L}(\mathcal{D}_{tr}, \mathcal{T}+\mathcal{S}), \mathcal{D}_{vld}, \mathcal{T}+\mathcal{S}), \tag{1}$$

Algorithm 1 `OpenFE`

Input: $\mathcal{D}$: dataset, $\mathcal{T}$: feature set, $\mathcal{O}$: operators
Output: new feature set
Initialize order $= 1$.
while order $<$ predefined max order **do**
  ▷ Expansion step
  Initialize $A(\mathcal{T})$ by applying $\mathcal{O}$ on $\mathcal{T}$.
  ▷ Reduction step
  $\hat{\boldsymbol{y}} \leftarrow$ generate predictions with $\mathcal{T}$ on $\mathcal{D}$.
  $A(\mathcal{T}) = \texttt{SuccessivePruning}(A(\mathcal{T}), \mathcal{D}, \hat{\boldsymbol{y}})$.
  $A(\mathcal{T}) = \texttt{FeatureAttribution}(A(\mathcal{T}), \mathcal{D}, \hat{\boldsymbol{y}})$.
  $\mathcal{T} \leftarrow \mathcal{T} + \text{Top_k}(A(\mathcal{T}))$.
  order $\leftarrow$ order + 1.
end while
return $\mathcal{T}$.

Table 1: Examples of operators and their effects on features in Figure 1. All examples follow the Value$_{\text{Name}}$ format.

Type	Operator	Example
Unary	Freq	African$_{\text{Race}} \rightarrow 0.21$
Unary	Log	$1.0_{\#\text{Diagnose}} \rightarrow 0.0$
Binary	Min	$(18.0_{\#\text{Diagnose}}, 9.0_{\#\text{Medications}}) \rightarrow 9.0$
Binary	$\div$	$(16.0_{\#\text{Diagnose}}, 5.0_{\#\text{Medications}}) \rightarrow 3.2$

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 $A(\mathcal{T})$ that maximizes the evaluation metric.

3. OpenFE

OpenFE follows the expand-and-reduce framework (Kanter & Veeramachaneni, 2015; Lam et al., 2021; Kaul et al., 2017; Shi et al., 2020; Katz et al., 2016) for automated feature generation, in which we first expand the candidate feature space, and then reduce the space by removing ineffective features. The procedure of OpenFE is presented in Algorithm 1 and Figure 2. In the following we describe the expansion step and the reduction step in turn.

3.1. The Expansion Step

In the expansion step, we transform the base features by the operators to create a pool of candidate features. The operators are critical in the expansion step, as they determine the search space for new features. The operators are classified into unary operators (such as log, sigmoid, square), and binary operators, (such as $\times$, $\div$, min, max, GroupByThenMean) according to the number of features they operate on. We present several examples that use the operators to transform base features in Table 1. The details of all the operators are presented in Appendix D.2. The candidate feature space is expanded via enumeration. For each operator, we iterate over all the base features and transform the base features into new features by the operator. For instance, for the operator "$\div$", we iterate over all possible numerical feature pairs $\{(\tau_1, \tau_2)\}$ to generate a list of candidate features $\{\tau_1 \div \tau_2\}$. The candidate feature space contains all the first-order transformations of base features, where each transformation uses one operator. Given a dataset with m features, the number of candidate features is $O(dm^2)$, where d is the number of binary operators. There may be millions of candidate features for a dataset containing hundreds of features.

Algorithm 2 `FeatureBoost`

Input: $\mathcal{D}$: dataset, $\mathcal{T}'$: feature set, $\hat{\boldsymbol{y}}$: predictions on $\mathcal{T}$
Output: incremental performance of $\mathcal{T}'$
Initialize $L(f)$ as the objective function of f with $\mathcal{T}$.
Initialize a new model f'.
Optimize $L(f') = \sum_{i=1}^{n} l(y_i, \hat{y}_i + f'(\boldsymbol{x}_i[\mathcal{T}']))$
$\Delta \leftarrow L(f) - L(f')$
return Δ

3.2. The Reduction Step

After the expansion step, we obtain a vast pool of candidate features. Therefore, the reduction framework aims at efficiently and accurately identifying the effective features from the vast pool of candidate features. To this end, first we propose FeatureBoost, an efficient algorithm to evaluate the effectiveness of new features. Although FeatureBoost provides an efficient way to evaluate each candidate feature, it is still computationally expensive to compute and evaluate the huge number of candidate features. Therefore, we propose a two-stage pruning algorithm on top of FeatureBoost to further accelerate the reduction step.

3.2.1. FeatureBoost: Fast Incremental Performance Evaluation

One of the key challenges in automated feature generation is 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 in the base feature set, retraining the machine learning model, and observing changes in the objective function (Katz et al., 2016). In the following we formally describe the standard evaluation procedure. Given a dataset of n samples, we denote the dataset $\mathcal{D} = \{(\boldsymbol{x}_i[\mathcal{T}], y_i) \,|\, i = 1, 2, \cdots, n\}$, where $\boldsymbol{x}_i[\mathcal{T}] \in \mathbb{R}^{|\mathcal{T}|}$ denotes the i-th sample projected to the feature set $\mathcal{T}$. Given a trained machine learning model f using $\mathcal{T}$, we denote the predictions as $\hat{y}_i = f(\boldsymbol{x}_i[\mathcal{T}])$ and the objective function

Algorithm 3 `SuccessivePruning`

Input: $\mathcal{D}$: dataset, $\hat{\boldsymbol{y}}$: predictions on $\mathcal{T}$,
$A(\mathcal{T})$: candidate feature set, q: integer
Output: pruned new feature set
Divide $\mathcal{D}$ equally into 2^q data blocks.
$A_0(\mathcal{T}) \leftarrow A(\mathcal{T})$.
for $i = 0$ **to** q **do**
  ▷ Create a subset $\mathcal{D}_i$ with 2^i randomly selected data blocks
  for new feature $\tau \in A_i(\mathcal{T})$ **do**
    $\Delta_\tau = \texttt{FeatureBoost}(\mathcal{D}_i, \{\tau\}, \hat{\boldsymbol{y}})$.
  end for
  $A_i(\mathcal{T}) \leftarrow$ deduplicate $A_i(\mathcal{T})$.
  $A_{i+1}(\mathcal{T}) \leftarrow$ Take the top half of $A_i(\mathcal{T})$ based on Δ.
end for
for $\tau \in A_{q+1}(\mathcal{T})$ **do**
  if $\Delta_\tau \leq 0$ **then**
    $A_{q+1}(\mathcal{T}) \leftarrow A_{q+1}(\mathcal{T}) \backslash \{\tau\}$.
  end if
end for
return $A_{q+1}(\mathcal{T})$

as $L(f) = \sum_{i=1}^{n} l(y_i, f(\boldsymbol{x}_i[\mathcal{T}]))$, where $l(\cdot,\cdot)$ is the loss function (e.g., the mean squared error). When introducing a set of new features $\mathcal{T}'$, the standard procedure retrains a new model f' with the full feature set $\mathcal{T} + \mathcal{T}'$, and computes the objective function $L(f') = \sum_{i=1}^{n} l(y_i, f'(\boldsymbol{x}_i[\mathcal{T} + \mathcal{T}']))$. The loss improvement $L(f) - L(f')$ indicates the incremental performance of $\mathcal{T}'$, which is larger the better. However, retraining the model with the full feature set is usually time-consuming, making the standard evaluation procedure prohibitively expensive.

In this paper, we argue that it is unnecessary to retrain the model to estimate the incremental performance of new features. In response, we present FeatureBoost, a fast incremental performance evaluation algorithm. As illustrated in Algorithm 2, the input to FeatureBoost comprises the dataset $\mathcal{D}$, the feature set $\mathcal{T}'$, and the prediction $\hat{\boldsymbol{y}}$. Inspired by gradient boosting, FeatureBoost only trains a model with the new feature set $\mathcal{T}'$ to fit the residuals between $\hat{\boldsymbol{y}}$ and the target $\boldsymbol{y}$. Formally, it optimizes the new model f' to minimize $L(f') = \sum_{i=1}^{n} l(y_i, \hat{y}_i + f'(\boldsymbol{x}_i[\mathcal{T}']))$. Finally, $\Delta = L(f) - L(f')$ serves as an estimate of the incremental performance of $\mathcal{T}'$.

In general, the input feature set for FeatureBoost is the new features, which allows the algorithm to only train f' on few features to be efficient. This implementation is employed in the first stage of the pruning algorithm (introduced in the next subsection), which coarsely reduces the number of candidate features. However, in the second stage, we need to consider the interaction effects between the new features and the original features, which is more accurate when evaluating the incremental performance of the new features. In this case, the input $\mathcal{T}'$ is in fact the full feature set (i.e., $\mathcal{T} + \mathcal{T}'$) to allow the model to better rank the new features. It is worth noting that even when the input becomes full set, FeatureBoost is still superior to the standard evaluation because it exploits $\hat{\boldsymbol{y}}$ to converge faster.

Algorithm 4 `FeatureAttribution`

Input: $\mathcal{D}$: dataset, $\mathcal{T}$: feature set, $A'(\mathcal{T})$: candidate feature set, $\hat{\boldsymbol{y}}$: predictions on $\mathcal{T}$
Output: sorted $A'(\mathcal{T})$.
$\Delta = \texttt{FeatureBoost}(\mathcal{D}, \mathcal{T} + A'(\mathcal{T}), \hat{\boldsymbol{y}})$.
Attribute importance scores to each feature in $A'(\mathcal{T})$ according to their contributions to the loss reduction Δ.
$A'(\mathcal{T}) \leftarrow$ sort $A'(\mathcal{T})$ based on their importance scores.
return $A'(\mathcal{T})$

3.2.2. A Two-stage Pruning Algorithm

Although FeatureBoost provides an efficient way to estimate the incremental performance of candidate features, it is still computationally expensive to compute and evaluate the huge number of candidate features. Therefore, we propose a two-stage pruning algorithm on top of FeatureBoost to efficiently eliminate redundant new features. Motivated by the fact that effective features are usually sparse in the candidate feature space, we devise a two-stage pruning algorithm to perform feature pruning in a coarse-to-fine manner. Concretely, the first stage is to coarsely reduce the number of candidate features by examining the effectiveness of each feature alone, while the second stage takes into account the fine-grained interaction between features.

Stage I: Successive Featurewise Pruning We propose a successive featurewise pruning algorithm to quickly reduce the number of candidate features (Algorithm 3). The algorithm is motivated by the successive halving algorithm in multi-armed bandit problems (Even-Dar et al., 2006; Zhou et al., 2014), where successive halving dynamically allocates computing resources to promising arms. In our settings, first we split the dataset into 2^q data blocks, where each data block has $\lfloor \frac{n}{2^q} \rfloor$ samples and q is a hyperparameter. Then the algorithm proceeds iteratively to remove redundant candidate features. In the i-th iteration, first we create a subset $\mathcal{D}_i$ by randomly selecting 2^i data blocks. Then, for each candidate feature τ, we compute $\Delta_\tau = \texttt{FeatureBoost}(\mathcal{D}_i, \{\tau\}, \hat{\boldsymbol{y}})$ as the score of the candidate feature. In the end of each iteration, we keep the top half of candidate features and eliminate the rest according to the scores Δ. Finally, each candidate feature τ with a positive Δ_τ can be returned. Besides, if there are two features with exactly the same values, we remove one of them for deduplication. For example, $\max(\tau_1, \tau_2)$ and

$\max(\tau_1, \tau_3)$ are exactly the same when the minimum value of τ_1 is larger than the maximum value of both τ_2 and τ_3. In this case, we only keep $\max(\tau_1, \tau_2)$ as the candidate feature.

Stage II: Feature Importance Attribution Unlike the coarse stage I, stage II considers the fine-grained interaction effects between the candidate features and the base features (Algorithm 4). Let the candidate features after pruning be $A'(\mathcal{T})$. We use candidate features $A'(\mathcal{T})$ and base features $\mathcal{T}$ together as the inputs to FeatureBoost. $\Delta = \texttt{FeatureBoost}(\mathcal{D}, \mathcal{T} + A'(\mathcal{T}), \hat{\boldsymbol{y}})$ evaluates the incremental performance of the remaining candidate features $A'(\mathcal{T})$, which considers the interaction effects between $\mathcal{T}$ and $A'(\mathcal{T})$. We are interested in the contribution of each candidate feature in the loss reduction Δ. Feature importance attribution methods can attribute the importance scores to each feature (Lundberg et al., 2018). Popular methods include mean decrease in impurity (MDI) (Breiman, 2001), permutation feature importance (PFI) (Breiman, 2001), and SHAP (Lundberg et al., 2018). We rank the candidate features according to their importance scores. Finally, we only select the top-ranked candidate features to improve the generalization of our algorithm.

3.3. Implementation

In OpenFE, we use gradient boosting decision trees (GBDT) (Friedman, 2001) to model tabular data for two reasons: 1) GBDT is usually the best performing model on tabular data where features are individually meaningful (Gorishniy et al., 2021; Borisov et al., 2021). 2) GBDT can automatically handle missing values and categorical features, which is convenient for automation (Ke et al., 2017). We use the popular LightGBM implementation (Ke et al., 2017). We use MDI for feature importance attribution. We compare MDI, PFI, and SHAP in Appendix C.6. 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. Theoretical Advantage of Feature Generation

In this section, we study the advantage of feature generation from a theoretical perspective. We present a simple yet representative setting in which the test loss of empirical risk minimization using both base features and generated features converges to zero provably as the number of training samples increases, while the test loss for any learning model using only base features is at least a positive constant. In particular, we present a transductive learning setting, which can capture important characteristics of many datasets one may encounter frequently in data science applications, e.g., the IEEE-CIS Fraud Detection dataset (we also conduct experiments on this dataset in Section 5.5). Due to space limit, a formal and detailed description of the model can be found in Appendix A. We briefly introduce the high-level idea here.

Many tabular datasets contain both categorical and numerical attributes (i.e., features). A categorical feature partitions the dataset into groups (each associated with a distinct category). For a data point (X, Y). the target Y is correlated with not only the feature X, but also certain statistics of the group containing (X, Y). Datasets with such characteristics are abundant in data science applications. As a concrete example, suppose each training data point records a transaction, and one special categorical feature is user id (each user may have many transactions in the table. All transactions of the same user form a group). The target Y we want to predict about the transaction (e.g., probability of fraudulence) may depend on not only the features of this particular transaction, but also some statistics of this user (e.g., average size of his/her transactions). Hence, one can see that operations such as $\mathrm{GroupByThenMean}$ (group by the user id) can provide statistical information about the user by aggregating the information from all data points associated to this user.

We present a theoretical data model, which is a two-phase data generation model, to capture the above characteristics. Under fairly standard learning theoretic assumptions (i.e., bounded Rademacher complexity), we prove that empirical risk minimization augmented with feature generation (such as the $\mathrm{GroupByThenMean}$ operation) can achieve vanishing test loss as the sample size and group size increase.

Theorem 4.1. *(informal) Assume the data set is generated according to the two-phase process described in Appendix A. Denote the number of groups in the training set and test set by k_1 and k_2 respectively, and the number of data points in each group by h. There is a feature generation function H, such that the test loss of the empirical risk minimizer $\hat{f}$ can be bounded by*

$$L_{\mathcal{D}_{test}}(H, \hat{f}) \leq O\Bigg(\mathrm{Rad}_{k_1}(\mathcal{F}) + \sqrt{\ln(4\delta^{-1})/k_1} + \sqrt{d\ln(4d(k_1+k_2)\delta^{-1})/h}\Bigg)$$

with probability at least $1-\delta$. In particular, assuming the Rademacher complexity $\mathrm{Rad}_{k_1}(\mathcal{F}) \to 0$ as $k_1 \to \infty$, the test loss approaches to 0 when $k_1, h \to \infty$.

On the other hand, if we do not use any feature generation, we prove that any predictor f' (no matter how complicated f' is) incurs a non-vanishing constant test loss.

Theorem 4.2. *(informal) In case that we do not use any feature generation, there exists a problem instance such that, no matter how large k_1, k_2, and h are, for any predictor $f' : \mathcal{X} \to \mathcal{Y}$, the test loss $L_{\mathcal{D}_{test}}(f') \geq \frac{3}{64}$.*

Table 2: Properties of datasets used in our experiments. Notation: "RMSE" $\sim$ root-mean-square error, "AUC" $\sim$ area-under-curve, "Acc." $\sim$ accuracy.

Dataset	Regression			Classification						
	CA	MI	ME	TE	BR	DI	NO	VE	JA	CO
# samples (k)	20.6	1200	163	51.0	900	102	34.4	98.5	83.7	581
# numerical features	7	111	5	21	31	3	34	100	54	9
# categorical features	0	0	6	22	0	34	29	0	0	0
# ordinal features	1	25	0	14	27	10	55	0	0	45
# classes	–	–	–	2	2	2	2	2	4	7
metric	RMSE	RMSE	RMSE	AUC	AUC	AUC	AUC	AUC	Acc.	Acc.

Table 3: Comparisons between OpenFE and baseline methods. The results that demonstrate a significant improvement over others are highlighted in **bold**. We repeat each experiment 10 times and apply Welch's t-test with unequal variance and a p-value of 0.05 to assess the significance. ✗ denotes a failure due to exceeding the runtime limits (24 hours). – means that the method cannot run on multi-classification and regression datasets.

Method	CA ↓	MI ↓	ME ↓	TE ↑	BR ↑	DI ↑	NO ↑	VE ↑	JA ↑	CO ↑
Base	$0.432_{\pm 0.004}$	$0.744_{\pm 0.000}$	$1128.4_{\pm 1.639}$	$0.671_{\pm 0.002}$	$0.756_{\pm 0.004}$	$0.731_{\pm 0.001}$	$0.996_{\pm 0.000}$	$0.925_{\pm 0.000}$	$0.721_{\pm 0.002}$	$0.969_{\pm 0.001}$
NN	$0.479_{\pm 0.001}$	$0.750_{\pm 0.000}$	$1413.7_{\pm 2.425}$	$0.661_{\pm 0.002}$	$0.748_{\pm 0.004}$	$0.717_{\pm 0.001}$	$0.992_{\pm 0.000}$	$0.924_{\pm 0.001}$	$0.720_{\pm 0.001}$	$0.966_{\pm 0.000}$
FCTree	$0.432_{\pm 0.003}$	$0.744_{\pm 0.000}$	$1088.9_{\pm 1.072}$	$0.670_{\pm 0.001}$	$0.750_{\pm 0.004}$	$0.731_{\pm 0.001}$	$0.996_{\pm 0.000}$	$0.926_{\pm 0.000}$	$0.719_{\pm 0.001}$	$0.971_{\pm 0.001}$
SAFE	–	–	–	$0.673_{\pm 0.001}$	$0.750_{\pm 0.004}$	$0.730_{\pm 0.002}$	$0.996_{\pm 0.000}$	$0.925_{\pm 0.001}$	–	–
AutoFeat	$0.444_{\pm 0.002}$	$0.744_{\pm 0.000}$	$1172.2_{\pm 2.235}$	$0.672_{\pm 0.002}$	$0.750_{\pm 0.005}$	$0.732_{\pm 0.001}$	$0.996_{\pm 0.000}$	$0.925_{\pm 0.000}$	$0.721_{\pm 0.001}$	$0.968_{\pm 0.001}$
AutoCross	–	–	–	$0.651_{\pm 0.002}$	$0.765_{\pm 0.001}$	$0.732_{\pm 0.001}$	$0.993_{\pm 0.000}$	$0.921_{\pm 0.000}$	–	–
FETCH	$0.430_{\pm 0.002}$	✗	$1130.6_{\pm 1.045}$	$0.673_{\pm 0.002}$	✗	$0.731_{\pm 0.002}$	$0.996_{\pm 0.000}$	$0.927_{\pm 0.000}$	$0.720_{\pm 0.001}$	✗
OpenFE	$\mathbf{0.421}_{\pm 0.002}$	$\mathbf{0.738}_{\pm 0.000}$	$\mathbf{982.0}_{\pm 1.745}$	$\mathbf{0.680}_{\pm 0.002}$	$\mathbf{0.786}_{\pm 0.002}$	$\mathbf{0.888}_{\pm 0.002}$	$\mathbf{0.997}_{\pm 0.000}$	$0.928_{\pm 0.000}$	$\mathbf{0.729}_{\pm 0.001}$	$\mathbf{0.974}_{\pm 0.000}$

5. Experiments

5.1. Datasets and Evaluation Metrics

We collect a diverse set of ten public datasets. Most datasets are frequently used in previous studies (Gorishniy et al., 2021; 2022; Grinsztajn et al., 2022; Strack et al., 2014; Siebert, 1987; Candillier & Lemaire, 2012). Each dataset has exactly one train-validation-test split, and all methods use the same split. The datasets we collect include: California Housing (CA), Microsoft (MI), Medical (ME), Diabetes (DI), Nomao (NO), Vehicle (VE), Broken Machine (BR), Telecom (TE), Jannis (JA), Covertype (CO). We summarize the dataset properties in Table 2. See more detailed descriptions in the appendix.

5.2. Baseline Methods for Comparisons

The baseline methods for comparisons include: **Base**, the base feature set without feature generation, **FCTree** (Fan et al., 2010), **SAFE** (Shi et al., 2020), **AutoFeat** (Horn et al., 2019), **AutoCross** (Luo et al., 2019), and **FETCH** (Li et al., 2023). We provide another baseline that uses the last hidden layer of DCN-V2 as new features. DCN-V2 (Wang et al., 2021) is a DNN architecture developed for modeling tabular data and claims to be able to capture feature interactions automatically. We denote this baseline as **NN**. Most existing automated feature engineering methods do not open-source their codes. We reproduce FCTree, SAFE, and AutoCross according to the descriptions in these papers. We validate our reproduction by comparing the reproduced results with those in the corresponding papers (see Appendix C.4). 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 crucial for generalization in learning-based methods, however previous research did not describe which datasets were utilized for training (Nargesian et al., 2017; Khurana et al., 2018). We run OpenFE on their datasets and compare our results with the numbers reported in their papers in Appendix C.1.

5.3. Comparison Results

We compare OpenFE with other baseline methods. We use LightGBM (Ke et al., 2017) as the learning algorithms to evaluate the effectiveness of the new features generated by different methods. Hyperparameter tuning follows a standard benchmarking study (Gorishniy et al., 2021), and we tune the hyperparameters using the base feature set. For each dataset, we generate first-order features and include the same number of new features for different methods for a fair comparison. We discuss generating high-order features in Appendix E.2. We present the comparison results in Table 3. We can observe that OpenFE has clear advantages over baseline methods. When baseline methods fail to generate

Table 4: The effect of OpenFE on a variety of DNNs. We report the results on the test set over 10 random seeds (see Appendix C.2 for standard deviations). Statistically significant improvements are marked in **bold**.

Model	Features	CA ↓	MI ↓	NO ↑	VE ↑	JA ↑
AutoInt	Base	0.474	0.750	0.993	0.927	0.720
	OpenFE	**0.462**	**0.745**	**0.994**	**0.929**	**0.724**
DCN-V2	Base	0.483	0.749	0.993	0.924	0.716
	OpenFE	**0.472**	**0.743**	**0.994**	**0.926**	0.716
TabNet	Base	0.510	0.751	0.993	0.924	0.724
	OpenFE	**0.501**	**0.746**	**0.995**	**0.926**	0.724
NODE	Base	0.464	0.745	0.994	0.927	0.727
	OpenFE	**0.457**	**0.740**	**0.995**	**0.929**	**0.731**
FT-T	Base	0.459	0.746	0.993	0.927	0.733
	OpenFE	**0.453**	**0.741**	**0.995**	**0.929**	**0.738**

effective new features, they include uninformative features in the base feature set, which brings additional noise and does harm to the generalization of the learning model.

5.4. Feature Generation for Neural Networks

We show that the new features 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), **DCN-V2** (Wang et al., 2021), **NODE** (Popov et al., 2019), **TabNet** (Arık & Pfister, 2021), **FT-Transformer** (Gorishniy et al., 2021). We follow the same implementations and hyperparameter tuning in Gorishniy et al. (2021). We present the results in Table 4. The features generated by OpenFE greatly enhance the performance of different models in most cases.

5.5. Compare OpenFE with Kaggle Experts

The first Kaggle competition is IEEE-CIS Fraud Detection (Kaggle, 2019), 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 on Kaggle, with 6,351 data science teams participating. The competition's first place team made public the features they generated after the competition ended (Deotte, 2019), which we refer to as Expert. A baseline model of XGBoost (Chen & Guestrin, 2016) without feature generation ranks at 2286 among 6351 teams on the private leaderboard. The baseline model with features generated by the team ranks at 76/6351. The same baseline model with features generated by OpenFE ranks at 42/6351, which outperforms the features generated by the first-place team. We present the results in Table 5. The first-place team only shared one of the XGBoost models they used. The actual model they used to achieve the first rank is an ensemble of many learning models (including LightGBM, CatBoost (Prokhorenkova et al., 2018), and XGBoost). But the team withheld the details of these models (Deotte, 2019), which is the reason their reproduced result does not rank first.

Table 5: Results of OpenFE and Expert (feature generation by the 1-st place team in IEEE and the 8-th place team in BNP) in two Kaggle competitions. Notation: pub. ∼ public score, pri. ∼ private score.

Feature	Order	IEEE ↑		BNP ↓	
		pub.	pri.	pub.	pri.
Base	–	0.946	0.918	0.438	0.436
Expert	first	0.960	0.933	0.433	0.431
	all	0.960	0.932	**0.432**	**0.430**
OpenFE	first	**0.962**	**0.936**	0.435	**0.432**
	all	**0.962**	**0.936**	0.432	**0.430**

The second competition is BNP Paribas Cardif Claims Management (Kaggle, 2016), where the goal is to predict whether an insurance claim should be approved. The competition's 8-th place team made public their generated features after the competition ended (KOHEI, 2020) (the winners with higher rankings did not share their codes). We evaluate the performance of OpenFE in a similar way. A baseline model using Catboost (Prokhorenkova et al., 2018) without feature generation ranks at 31 among 2920 teams. The baseline model with features generated by experts ranks at 12/2920, while the baseline model with features generated by OpenFE also ranks at 12/2920.

Table 6: Results of the ablation study.

	CA ↓	MI ↓	DI ↑	NO ↑	VE ↑	CO ↑
OpenFE	0.421	0.738	0.888	0.997	0.928	0.974
OpenFE-MI	0.428	0.744	0.887	0.996	0.926	0.971
w.o. Successive	0.428	0.744	0.885	0.995	0.927	0.965

5.6. Analysis and Discussion

Ablation Study We conduct an ablation study to justify the design choices of OpenFE. We name different variants of OpenFE as follows: 1) **OpenFE-MI**. Instead of using FeatureBoost to evaluate the effectiveness of new features, we use mutual information between the feature and the target as the scoring criterion. 2) ***w.o.* Successive**. OpenFE without the successive featurewise pruning. We subsample the data so that generating all the features can fit in the memory. We present the results in Table 6. One can see from the results that: 1) Mutual information is worse than FeatureBoost in evaluating the effectiveness of new features. 2) Directly subsampling the data usually hurts the performance.

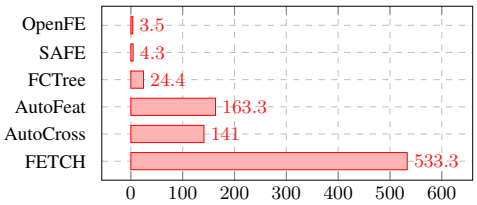

Figure 3: Runtime comparisons in minutes. The x-axis is the average runtime of TE, DI, NO, VE datasets.

Runtime Comparisons We compare the runtime of different methods on the TE, DI, NO, VE datasets. The results are presented in Figure 3. One can see that OpenFE is significantly more efficient than AutoFeat, AutoCross, and FETCH. OpenFE only runs a few minutes to generate features on most medium-scale datasets. We provide the runtime on different datasets in Appendix C.5. Complexity analysis is provided in Appendix F.

Discussion of High-order Features During the experiments, we find that generating second-order features is hardly effective for most benchmarking datasets. Table 5 presents the Kaggle results using only first-order features and both first-order and high-order features (all) separately. Generating high-order features does not seem to be helpful for Expert or OpenFE in the IEEE competition. In the BNP competition, generating high-order features provides a slight improvement in the test score. First-order features are usually more important than high-order features in feature generation. See more detailed discussions in Appendix E.2.

Is Feature Generation Effective for All Datasets? In order to answer this question, we conduct a large-scale empirical study using the OpenML CC18 benchmark (Bischl et al., 2021) with 68 dataset (see results in Table 7). We found that OpenFE did not result in significant improvements for 19 of those datasets, and neither did other feature generation methods. It is possible that some datasets do not necessitate feature generation. While we do not fully understand which datasets benefit most from feature generation, we would like to highlight that OpenFE is highly efficient, allowing researchers to quickly obtain preliminary results and assess the impact of feature generation for their dataset.

6. Related Work

Expand-and-reduce is arguably the most popular framework in automated feature generation. Most existing feature generation methods employ statistical methods to identify and remove redundant features. For example, Data Science Machine (DSM) (Kanter & Veeramachaneni, 2015) uses the f-value and One Button Machine (OneBM) (Lam et al., 2021) uses the Chi-square test between the feature and the target to remove redundant features. DSM and OneBM focus on feature generation in relational databases. FCTree (Fan et al., 2010) uses information gain and 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 evaluate a new feature. LFE (Nargesian et al., 2017) and ExploreKit (Katz et al., 2016) design meta features based on statistical methods and use a meta learning approach to determine the effectiveness of new features.

Although statistical methods are widely employed, there are many studies showing that statistically significant features do not always translate into good predictors (Lo et al., 2015; Ward et al., 2010; Welch & Goyal, 2008; Zhang et al., 2021). The effectiveness of a new feature may be encompassed by the base feature set, even if the new feature is significantly correlated with the target. Another line of works employs the standard approach to evaluate the incremental performance of new features. Neural Feature Search (Chen et al., 2019) utilizes a recurrent neural network controller to search for new features. Khurana et al. (2018) uses reinforcement learning to traverse a transformation graph for high-order features. FETCH (Li et al., 2023) built a feature engineering pipeline based on the Markov decision process.

7. Conclusion & Limitations

We propose OpenFE, a powerful automated feature generation tool with FeatureBoost and a two-stage pruning algorithm. Extensive experiments show that OpenFE achieves SOTA on ten benchmark datasets and is competitive against human experts in feature generation. We provide theoretical evidence that feature generation is crucial in modeling tabular data. We open-source the codes and datasets.

OpenFE currently has some limitations: 1) OpenFE cannot handle time series data. For example, we may leverage statistical ideas developed in time series literature, and the new features should satisfy the time series constraint (they should be computed from the past data). OpenFE cannot handle such constraints. 2) Even though OpenFE is highly efficient and can handle large-scale datasets with millions of samples and hundreds of features, in some cases where the dataset is exceptionally large or computing resources are limited, one could perform feature selection or use a fraction of the base features to generate candidate features.

8. Acknowledgement

Tianping Zhang, Zheyu Zhang, Zhiyuan Fan, and Jian Li are supported in part by the National Natural Science Foundation of China Grant 62161146004, Turing AI Institute of Nanjing and Xi'an Institute for Interdisciplinary Information Core Technology.

Table 7: Additional comparison results on 49 datasets from OpenML CC18 benchmark where feature generation can yield a significant improvement. The metric is accuracy for all datasets. OpenFE outperforms all baseline methods significantly on 40 out of these 49 datasets and provides an average accuracy improvement of 1.9% over the base features across these 49 datasets. The results are presented in descending order based on the performance gap between OpenFE and Base, from largest to smallest (see more detailed experimental settings in Appendix C.3). A dash ('–') indicates that the method is not applicable to multi-classification datasets. ✗ denotes a failure due to exceeding the runtime limits (24 hours) or bugs in the python package of AutoFeat. The results that demonstrate a significant improvement over others are highlighted in **bold**. We repeat each experiment 10 times and apply Welch's t-test with unequal variance and a p-value of 0.05 to assess the significance. n is the number of samples and p is the number of features.

Dataset	n	p	OpenFE	Base	FCTree	SAFE	AutoFeat	AutoCross	FETCH
balance-scale	625	4	**0.990**$_{\pm 0.003}$	0.860$_{\pm 0.005}$	0.919$_{\pm 0.002}$	–	0.850$_{\pm 0.003}$	–	0.965$_{\pm 0.004}$
jungle_chess	44819	6	**0.985**$_{\pm 0.000}$	0.864$_{\pm 0.000}$	0.950$_{\pm 0.000}$	–	0.864$_{\pm 0.000}$	–	0.950$_{\pm 0.000}$
cylinder-bands	540	37	**0.865**$_{\pm 0.014}$	0.798$_{\pm 0.007}$	0.845$_{\pm 0.013}$	0.805$_{\pm 0.021}$	✗	0.822$_{\pm 0.014}$	0.733$_{\pm 0.022}$
vehicle	846	18	0.800$_{\pm 0.009}$	0.738$_{\pm 0.005}$	0.725$_{\pm 0.008}$	–	0.745$_{\pm 0.007}$	–	0.798$_{\pm 0.005}$
eucalyptus	736	19	**0.722**$_{\pm 0.013}$	0.675$_{\pm 0.011}$	0.703$_{\pm 0.008}$	–	0.665$_{\pm 0.010}$	–	0.703$_{\pm 0.007}$
mfeat-morphological	2000	6	**0.767**$_{\pm 0.002}$	0.730$_{\pm 0.000}$	0.732$_{\pm 0.000}$	–	0.732$_{\pm 0.000}$	–	0.739$_{\pm 0.002}$
steel-plates-fault	1941	27	**0.804**$_{\pm 0.005}$	0.770$_{\pm 0.008}$	0.786$_{\pm 0.007}$	–	0.766$_{\pm 0.006}$	–	0.800$_{\pm 0.005}$
madelon	2600	500	**0.870**$_{\pm 0.004}$	0.842$_{\pm 0.004}$	0.848$_{\pm 0.007}$	0.860$_{\pm 0.003}$	0.845$_{\pm 0.006}$	✗	✗
climate-model	540	18	0.969$_{\pm 0.004}$	0.943$_{\pm 0.006}$	0.942$_{\pm 0.004}$	0.972$_{\pm 0.000}$	0.943$_{\pm 0.006}$	0.943$_{\pm 0.004}$	**0.981**$_{\pm 0.003}$
analcatdata_dmft	797	4	**0.206**$_{\pm 0.000}$	0.181$_{\pm 0.000}$	0.181$_{\pm 0.000}$	–	0.181$_{\pm 0.000}$	–	0.181$_{\pm 0.000}$
credit-g	1000	20	0.747$_{\pm 0.011}$	0.726$_{\pm 0.012}$	0.736$_{\pm 0.012}$	0.712$_{\pm 0.005}$	0.734$_{\pm 0.007}$	0.723$_{\pm 0.010}$	0.746$_{\pm 0.011}$
blood-transfusion	748	4	**0.800**$_{\pm 0.000}$	0.780$_{\pm 0.000}$	0.773$_{\pm 0.002}$	0.751$_{\pm 0.009}$	0.780$_{\pm 0.000}$	0.755$_{\pm 0.011}$	0.762$_{\pm 0.007}$
credit-approval	690	15	**0.879**$_{\pm 0.006}$	0.859$_{\pm 0.008}$	0.871$_{\pm 0.004}$	0.873$_{\pm 0.006}$	0.874$_{\pm 0.006}$	0.867$_{\pm 0.008}$	0.839$_{\pm 0.006}$
mfeat-fourier	2000	76	**0.845**$_{\pm 0.004}$	0.825$_{\pm 0.008}$	0.817$_{\pm 0.003}$	–	0.822$_{\pm 0.005}$	–	0.815$_{\pm 0.003}$
GesturePhase	9873	32	**0.688**$_{\pm 0.002}$	0.668$_{\pm 0.002}$	0.656$_{\pm 0.003}$	–	0.666$_{\pm 0.003}$	–	0.659$_{\pm 0.005}$
pc4	1458	37	0.925$_{\pm 0.004}$	0.907$_{\pm 0.004}$	0.912$_{\pm 0.005}$	0.896$_{\pm 0.004}$	0.908$_{\pm 0.003}$	0.915$_{\pm 0.006}$	0.924$_{\pm 0.003}$
pc1	1109	21	**0.941**$_{\pm 0.003}$	0.923$_{\pm 0.000}$	0.924$_{\pm 0.001}$	0.927$_{\pm 0.002}$	0.923$_{\pm 0.000}$	0.921$_{\pm 0.005}$	0.924$_{\pm 0.002}$
breast-w	699	9	**0.991**$_{\pm 0.005}$	0.975$_{\pm 0.004}$	0.969$_{\pm 0.005}$	0.979$_{\pm 0.004}$	0.968$_{\pm 0.004}$	0.971$_{\pm 0.003}$	0.966$_{\pm 0.003}$
phoneme	5404	5	0.917$_{\pm 0.002}$	0.901$_{\pm 0.003}$	0.913$_{\pm 0.003}$	0.916$_{\pm 0.002}$	0.907$_{\pm 0.002}$	0.866$_{\pm 0.004}$	0.904$_{\pm 0.002}$
qsar-biodeg	1055	41	**0.854**$_{\pm 0.005}$	0.840$_{\pm 0.005}$	0.837$_{\pm 0.004}$	0.838$_{\pm 0.004}$	0.841$_{\pm 0.006}$	0.845$_{\pm 0.003}$	0.838$_{\pm 0.005}$
pc3	1563	37	0.905$_{\pm 0.004}$	0.892$_{\pm 0.006}$	0.894$_{\pm 0.006}$	0.894$_{\pm 0.005}$	0.891$_{\pm 0.006}$	0.904$_{\pm 0.004}$	0.895$_{\pm 0.005}$
mfeat-factors	2000	216	**0.960**$_{\pm 0.003}$	0.948$_{\pm 0.003}$	0.942$_{\pm 0.004}$	–	0.942$_{\pm 0.002}$	–	✗
car	1728	6	**0.997**$_{\pm 0.000}$	0.986$_{\pm 0.000}$	0.886$_{\pm 0.006}$	–	0.989$_{\pm 0.001}$	–	0.970$_{\pm 0.004}$
ilpd	583	10	0.717$_{\pm 0.011}$	0.706$_{\pm 0.012}$	0.691$_{\pm 0.008}$	0.715$_{\pm 0.015}$	0.713$_{\pm 0.008}$	0.708$_{\pm 0.016}$	0.680$_{\pm 0.012}$
electricity	45312	8	**0.945**$_{\pm 0.001}$	0.934$_{\pm 0.001}$	0.940$_{\pm 0.001}$	0.931$_{\pm 0.001}$	0.934$_{\pm 0.001}$	0.930$_{\pm 0.001}$	0.939$_{\pm 0.001}$
kc2	522	21	**0.800**$_{\pm 0.004}$	0.790$_{\pm 0.009}$	0.794$_{\pm 0.009}$	0.776$_{\pm 0.005}$	0.789$_{\pm 0.009}$	0.789$_{\pm 0.009}$	0.786$_{\pm 0.009}$
spambase	4601	57	**0.959**$_{\pm 0.001}$	0.949$_{\pm 0.001}$	0.947$_{\pm 0.002}$	0.954$_{\pm 0.001}$	0.948$_{\pm 0.001}$	0.951$_{\pm 0.002}$	0.955$_{\pm 0.002}$
ozone-level-8hr	2534	72	**0.954**$_{\pm 0.002}$	0.945$_{\pm 0.002}$	0.946$_{\pm 0.002}$	0.944$_{\pm 0.002}$	0.942$_{\pm 0.002}$	0.942$_{\pm 0.001}$	0.949$_{\pm 0.001}$
semeion	1593	256	**0.945**$_{\pm 0.003}$	0.937$_{\pm 0.003}$	0.939$_{\pm 0.003}$	–	0.937$_{\pm 0.003}$	–	✗
Bioresponse	3751	1776	**0.791**$_{\pm 0.005}$	0.784$_{\pm 0.004}$	0.784$_{\pm 0.006}$	0.784$_{\pm 0.004}$	0.784$_{\pm 0.004}$	✗	✗
wilt	4839	5	**0.986**$_{\pm 0.001}$	0.979$_{\pm 0.001}$	0.981$_{\pm 0.001}$	0.984$_{\pm 0.001}$	0.981$_{\pm 0.001}$	0.973$_{\pm 0.001}$	0.984$_{\pm 0.002}$
analcatdata_authorship	841	70	**0.989**$_{\pm 0.002}$	0.983$_{\pm 0.002}$	0.980$_{\pm 0.003}$	–	0.983$_{\pm 0.002}$	–	0.982$_{\pm 0.000}$
mfeat-karhunen	2000	64	**0.959**$_{\pm 0.002}$	0.953$_{\pm 0.003}$	0.950$_{\pm 0.002}$	–	0.955$_{\pm 0.002}$	–	0.946$_{\pm 0.002}$
MiceProtein	1080	77	**0.991**$_{\pm 0.002}$	0.985$_{\pm 0.003}$	0.988$_{\pm 0.003}$	–	0.983$_{\pm 0.002}$	–	0.980$_{\pm 0.003}$
segment	2310	16	**0.940**$_{\pm 0.002}$	0.934$_{\pm 0.001}$	0.935$_{\pm 0.001}$	–	0.933$_{\pm 0.002}$	–	0.935$_{\pm 0.001}$
churn	5000	20	0.961$_{\pm 0.001}$	0.956$_{\pm 0.001}$	0.959$_{\pm 0.001}$	0.957$_{\pm 0.001}$	0.956$_{\pm 0.001}$	0.944$_{\pm 0.001}$	**0.969**$_{\pm 0.001}$
connect-4	67557	42	**0.864**$_{\pm 0.001}$	0.859$_{\pm 0.001}$	0.858$_{\pm 0.001}$	–	0.859$_{\pm 0.001}$	–	✗
bank-marketing	45211	16	**0.913**$_{\pm 0.001}$	0.909$_{\pm 0.001}$	0.909$_{\pm 0.001}$	0.909$_{\pm 0.001}$	0.910$_{\pm 0.001}$	0.912$_{\pm 0.001}$	✗
first-order-theorem-proving	6118	51	**0.608**$_{\pm 0.003}$	0.604$_{\pm 0.004}$	0.605$_{\pm 0.002}$	–	0.601$_{\pm 0.003}$	–	0.600$_{\pm 0.003}$
dna	3186	180	**0.959**$_{\pm 0.002}$	0.956$_{\pm 0.002}$	0.957$_{\pm 0.001}$	–	0.956$_{\pm 0.002}$	–	✗
numerai28.6	96320	21	**0.517**$_{\pm 0.001}$	0.514$_{\pm 0.001}$	0.516$_{\pm 0.001}$	0.514$_{\pm 0.001}$	0.515$_{\pm 0.001}$	0.515$_{\pm 0.001}$	✗
pendigits	10992	16	**0.994**$_{\pm 0.000}$	0.991$_{\pm 0.001}$	0.993$_{\pm 0.000}$	–	0.991$_{\pm 0.000}$	–	0.993$_{\pm 0.001}$
jm1	10885	21	0.815$_{\pm 0.003}$	0.813$_{\pm 0.002}$	**0.820**$_{\pm 0.002}$	0.818$_{\pm 0.001}$	0.814$_{\pm 0.002}$	0.812$_{\pm 0.002}$	0.819$_{\pm 0.002}$
kr-vs-kp	3196	36	**0.996**$_{\pm 0.001}$	0.994$_{\pm 0.001}$	0.992$_{\pm 0.001}$	0.994$_{\pm 0.001}$	0.994$_{\pm 0.001}$	0.992$_{\pm 0.001}$	0.994$_{\pm 0.001}$
sick	3772	28	**0.989**$_{\pm 0.000}$	0.987$_{\pm 0.001}$	0.986$_{\pm 0.002}$	0.982$_{\pm 0.002}$	0.988$_{\pm 0.001}$	0.982$_{\pm 0.002}$	0.976$_{\pm 0.001}$
wall-robot-navigation	5456	24	**1.000**$_{\pm 0.000}$	0.998$_{\pm 0.000}$	0.999$_{\pm 0.000}$	–	0.999$_{\pm 0.000}$	–	0.999$_{\pm 0.000}$
adult	48842	14	**0.875**$_{\pm 0.001}$	0.874$_{\pm 0.001}$	0.873$_{\pm 0.001}$	0.872$_{\pm 0.001}$	0.873$_{\pm 0.001}$	0.873$_{\pm 0.001}$	✗
banknote-authentication	1372	4	**1.000**$_{\pm 0.000}$	0.999$_{\pm 0.002}$	0.996$_{\pm 0.000}$	0.996$_{\pm 0.000}$	0.995$_{\pm 0.002}$	0.996$_{\pm 0.002}$	0.998$_{\pm 0.002}$
isolet	7797	617	**0.961**$_{\pm 0.001}$	0.960$_{\pm 0.001}$	0.959$_{\pm 0.002}$	–	✗	–	✗

References

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

Bischl, B., Casalicchio, G., Feurer, M., Gijsbers, P., Hutter, F., Lang, M., Mantovani, R. G., van Rijn, J. N., and Vanschoren, J. Openml benchmarking suites. In Vanschoren, J. and Yeung, S. (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021.

Blackard, J. A. and Dean, D. J. 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.

Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., and Kasneci, G. Deep neural networks and tabular data: A survey. *arXiv preprint arXiv:2110.01889*, 2021.

Breiman, L. Random forests. *Machine learning*, 45(1): 5–32, 2001.

Candillier, L. and Lemaire, V. 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.

Chen, T. and Guestrin, C. 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.

Chen, X., Lin, Q., Luo, C., Li, X., Zhang, H., Xu, Y., Dang, Y., Sui, K., Zhang, X., Qiao, B., 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.

Deotte, C. Xgb fraud with magic - [0.9600], 2019. URL https://www.kaggle.com/code/cdeotte/xgb-fraud-with-magic-0-9600.

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

Erickson, N., Mueller, J., Shirkov, A., Zhang, H., Larroy, P., Li, M., and Smola, A. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

Even-Dar, E., Mannor, S., Mansour, Y., and Mahadevan, S. Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *Journal of machine learning research*, 7(6), 2006.

Fan, W., Zhong, E., Peng, J., Verscheure, O., Zhang, K., Ren, J., Yan, R., and Yang, Q. 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.

Friedman, J. H. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

Gorishniy, Y., Rubachev, I., Khrulkov, V., and Babenko, A. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34, 2021.

Gorishniy, Y., Rubachev, I., and Babenko, A. On embeddings for numerical features in tabular deep learning. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=pfI7u0eJAIr.

Grinsztajn, L., Oyallon, E., and Varoquaux, G. Why do tree-based models still outperform deep learning on typical tabular data? In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=Fp7__phQszn.

Guyon, I., Sun-Hosoya, L., Boullé, M., Escalante, H. J., Escalera, S., Liu, Z., Jajetic, D., Ray, B., Saeed, M., Sebag, M., et al. Analysis of the automl challenge series. *Automated Machine Learning*, pp. 177, 2019.

Horn, F., Pack, R., and Rieger, M. 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.

Kaggle, V. C. Bnp paribas cardif claims management, 2016. URL https://www.kaggle.com/competitions/bnp-paribas-cardif-claims-management/.

Kaggle, V. C. Ieee-cis fraud detection, 2019. URL https://www.kaggle.com/competitions/ieee-fraud-detection/overview.

Kanter, J. M. and Veeramachaneni, K. 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.

Katz, G., Shin, E. C. R., and Song, D. Explorekit: Automatic feature generation and selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 979–984. IEEE, 2016.

Kaul, A., Maheshwary, S., and Pudi, V. Autolearn—automated feature generation and selection. In *2017 IEEE International Conference on data mining (ICDM)*, pp. 217–226. IEEE, 2017.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

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

KOHEI. xfeat + cudf + lightgbm + catboost (wip), 2020. URL https://www.kaggle.com/code/confirm/xfeat-cudf-lightgbm-catboost-wip.

Lam, H. T., Buesser, B., Min, H., Minh, T. N., Wistuba, M., Khurana, U., Bramble, G., Salonidis, T., Wang, D., and Samulowitz, H. Automated data science for relational data. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 2689–2692. IEEE, 2021.

Li, L., Wang, H., Zha, L., Huang, Q., Wu, S., Chen, G., and Zhao, J. Learning a data-driven policy network for pre-training automated feature engineering. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=688hNNMigVX.

Lo, A., Chernoff, H., Zheng, T., and Lo, S.-H. Why significant variables aren't automatically good predictors. *Proceedings of the National Academy of Sciences*, 112 (45):13892–13897, 2015.

Lu, Y. An end-to-end automl solution for tabular data at kaggledays. *Google AI Blog*, 2019. URL https://ai.googleblog.com/2019/05/an-end-to-end-automl-solution-for.html.

Lundberg, S. M., Erion, G. G., and Lee, S.-I. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888*, 2018.

Luo, Y., Wang, M., Zhou, H., Yao, Q., Tu, W.-W., Chen, Y., Dai, W., and Yang, Q. 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.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. *Foundations of machine learning*. MIT press, 2018.

Nargesian, F., Samulowitz, H., Khurana, U., Khalil, E. B., and Turaga, D. S. Learning feature engineering for classification. In *Ijcai*, pp. 2529–2535, 2017.

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

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

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. Catboost: unbiased boosting with categorical features. *Advances in neural information processing systems*, 31, 2018.

Qin, T. and Liu, T.-Y. Introducing letor 4.0 datasets. *arXiv preprint arXiv:1306.2597*, 2013.

Shi, Q., Zhang, Y.-L., Li, L., Yang, X., Li, M., and Zhou, J. Safe: Scalable automatic feature engineering framework for industrial tasks. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pp. 1645–1656. IEEE, 2020.

Siebert, J. P. Vehicle recognition using rule based methods. 1987.

Song, W., Shi, C., Xiao, Z., Duan, Z., Xu, Y., Zhang, M., and Tang, J. 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.

Strack, B., DeShazo, J. P., Gennings, C., Olmo, J. L., Ventura, S., Cios, K. J., and Clore, J. N. Impact of hba1c measurement on hospital readmission rates: analysis of 70,000 clinical database patient records. *BioMed research international*, 2014, 2014.

Wang, R., Shivanna, R., Cheng, D., Jain, S., Lin, D., Hong, L., and Chi, E. 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.

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

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

Zhang, T., Feng, H., Chen, W., Chen, Z., Zheng, W., Luo, X.-N., Huang, W., and Tung, A. K. Chartnavigator: an interactive pattern identification and annotation framework for charts. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

Zhou, Y., Chen, X., and Li, J. Optimal pac multiple arm identification with applications to crowdsourcing. In *International Conference on Machine Learning*, pp. 217–225. PMLR, 2014.

Zhu, G., Xu, Z., Yuan, C., and Huang, Y. DIFER: differentiable automated feature engineering. In Guyon, I., Lindauer, M., van der Schaar, M., Hutter, F., and Garnett, R. (eds.), *International Conference on Automated Machine Learning, AutoML 2022, 25-27 July 2022, Johns Hopkins University, Baltimore, MD, USA*, volume 188 of *Proceedings of Machine Learning Research*, pp. 17/1–17. PMLR, 2022. URL https://proceedings.mlr.press/v188/zhu22a.html.

A. Theoretical Results

Problem Setting

In this section, we study the advantage of feature generation from a theoretical perspective. In particular, we present a simple yet representative setting in which empirical risk minimization augmented with feature generation can achieve smaller test loss than any learning model without feature generation.

Consider a regression problem with both numerical and categorical features in a transductive learning setting. Denote $\mathcal{X}^{\text{num}} \subseteq \mathbb{R}^d$ as the numerical feature space, $\mathcal{X}^{\text{cat}} \subseteq \mathbb{N}$ the categorical feature space, and $\mathcal{Y} \subseteq [0, 1]$ as the target domain. We assume $\mathcal{X}^{\text{num}}$ is d-dimensional and convex. The training data $\mathcal{D}_{\text{train}}$ consists of n training data $\{(X_i, Y_i)\}_{i=1,\ldots,n}$ where $X_i = (x_{i0}, x_{i1}, \ldots, x_{id})$, $x_{i0} \in \mathcal{X}^{\text{cat}}$ is the categorical feature value, $(x_{i1}, \ldots, x_{id}) \in \mathcal{X}^{\text{num}}$ are d numerical feature values. As a concrete example, one may think each training data as a transaction, where x_{i0} as the user id and $(x_{i1}, \ldots, x_{id})$ are some features about the user and the transaction, and Y is the target we want to predict about the transaction (e.g., lateness of payment, probability of fraudulence). The test data $\mathcal{D}_{\text{test}}$ consists of m data points $\{(X_i, Y_i)\}_{i=n+1,\ldots,m}$, but these Y_i are the *unknown* targets that we try to predict.

The data model: The training set $\mathcal{D}_{\text{train}}$ and test set $\mathcal{D}_{\text{test}}$ are generated by a two-phase process. Let $\Delta(\mathcal{X}^{\text{cat}})$ be the space all probability distributions defined over $\mathcal{X}^{\text{cat}}$, and $\mathcal{Q}$ be a probability distribution over $\Delta(\mathcal{X}^{\text{cat}})$. $\mathcal{D}_{\text{train}}$ is generated by repeating the following k_1 times: in the i-th iteration, we first take a sample $\mathcal{P}_i \in \Delta(\mathcal{X}^{\text{cat}})$ from $\mathcal{Q}$ (note that $\mathcal{P}_i$ is a distribution over $\mathcal{X}^{\text{cat}}$). Then, we sample a group of h training data points, each with its categorical feature value being $c_i \in \mathcal{X}^{\text{cat}}$ and its d numerical feature values being an i.i.d., sample from $\mathcal{P}$. Hence, all these h training data $X_{ih+1}, \ldots, X_{(i+1)h}$ have the same categorical value $c_i \in \mathcal{X}^{\text{cat}}$, and they form the i-th group G_i (e.g., the set of transactions of the i-th user). $\mathcal{D}_{\text{train}}$ contains k_1 groups $G_1, \ldots, G_{k_1}$. For the i-th data point (X_i, Y_i), we use $\mathrm{g}(i)$ to denote the index of the group that contains (X_i, Y_i). Hence, $(X_i, Y_i) \in G_{\mathrm{g}(i)}$. Let $\mathcal{F}$ be the hypothesis class and the true hypothesis is $f^* : \mathcal{X}^{\text{num}} \times \mathcal{X}^{\text{cat}} \to \mathcal{Y}$ such that $Y_i = f^*(x_{i1}, \ldots, x_{id}, Z_i)$ where $Z_i = \mathbb{E}_{X \sim \mathcal{P}_{\mathrm{g}(i)}}[X]$ (Z_i is a d-dimensional vector) (e.g., the target depends not only on the numerical feature values of this particular transaction, but also the mean of the statistics of the user).

The test dataset $\mathcal{D}_{\text{test}}$ is generated in the same way and it contains k_2 groups $G_{k_1+1}, \ldots, G_{k_1+k_2}$. We assume the categorical feature values $c_1, \ldots, c_{k_1}, c_{k_1+1} \ldots, c_{k_1+k_2}$ are all distinct (e.g., a user id that appears in the test set does not appear in the training set).

Feature generation: Here we are interested in features generation using operations such as $\mathrm{GroupByThenMean}$. In our setting, the groupby operation is based on the categorical feature $\mathcal{X}^{\text{cat}}$. So, for a data point X_i, we can generate a set of new features $\hat{X}_i$ which can be computed from all data points in the same group $G_{\mathrm{g}(i)}$. Formally, let $\mathcal{H}$ be the set of possible feature generation function and the new feature $\hat{X}_i$ is computed as follows:

$$\hat{X}_i = H(G_{\mathrm{g}(i)}) \text{ for some } H \in \mathcal{H}.$$

For a feature generation function H and predictor f, the loss on the data point (X_i, Y_i) is

$$L(H, f, X_i, Y_i)] = \|Y_i - f(X_i, \hat{X}_i)\|^2 = \|Y_i - f(X_i, H(G_{\mathrm{g}(i)}))\|^2.$$

Our goal is to find a feature generation function $H \in \mathcal{H}$ and a predictor $\hat{f} \in \mathcal{F}$ such that the test loss is minimized

$$L_{\mathcal{D}_{\text{test}}}(H, \hat{f}) = \mathop{\mathbb{E}}_{(X_i,Y_i)\in\mathcal{D}_{\text{test}}}[L(H, \hat{f}, X_i, Y_i)] = \frac{1}{m} \sum_{(X_i,Y_i)\in\mathcal{D}_{\text{test}}} \|Y_i - f(X_i, H(G_{\mathrm{g}(i)}))\|^2.$$

If we do not use any feature generation, the loss of the predictor f' [1] is simply

$$L_{\mathcal{D}_{\text{test}}}(f') = \mathop{\mathbb{E}}_{(X_i,Y_i)\in\mathcal{D}_{\text{test}}}[\|Y_i - f'(X_i)\|^2]$$

In the following, we show that under nature conditions, we can achieve vanishing test loss by using feature generation (as the number of training samples n and the minimum size of each group h become larger). See Theorem A.1. But if we only use the raw feature X_i for predicting Y_i, the expected test loss of any predictor is at least a positive constant. See Theorem A.2.

[1] Note that the input dimension for the predictor f' here is different since there is no generated feature. But this is not an issue for our following argument.

Learnability with Feature Generation

For a particular feature generation function H, we use $\mathrm{Rad}_k(\mathcal{F})$ to denote the empirical Rademacher complexity of $\mathcal{F}$ over k random samples:

$$\mathrm{Rad}_k(\mathcal{F}) = \mathop{\mathbb{E}}_{\{(X_i,Y_i)\}_{i=1}^k} \frac{1}{k} \mathop{\mathbb{E}}_{\sigma} \left[\sup_{f\in\mathcal{F}} \sum_{i=1}^k \sigma_i L(H, f, X_i, Y_i) \right]$$

where $\sigma = (\sigma_1, \cdots, \sigma_k)$ are independent Rademacher variables and X_i is an i.i.d. sample from $\mathcal{P}_i$, which is an i.i.d. sample from $\mathcal{Q}$, and $Y_i = f^*(X_i, Z_i)$ where $Z_i = \mathbb{E}_{X\sim\mathcal{P}_i}[X]$. We assume that $\mathrm{Rad}_{k_1}(\mathcal{F}) \to 0$ as $k_1 \to \infty$ (for many hypothesis classes, Rad_{k_1} scales as $O(\sqrt{1/k_1})$) (Mohri et al., 2018). We also assume any function $f(\cdot,\cdot) \in \mathcal{F}$ is Lipschitz on z: There exists constant $C_{\mathcal{F}}$ such that $|f(X, Z_1) - f(X, Z_2)| \le C_{\mathcal{F}}\|Z_1 - Z_2\|$ for any $z_1, z_2, x \in \mathcal{X}$ and $f \in \mathcal{F}$. We further assume there exists constant $B_{\mathcal{X}}$ such that $\sup_{x\in\mathcal{X}} \|x\| \le B_{\mathcal{X}}$.

Theorem A.1. *There is a feature generation function H, such that the test loss of the empirical risk minimizer $\hat{f}$ can be bounded by*

$$L_{\mathcal{D}_{test}}(H, \hat{f}) \le 2\mathrm{Rad}_{k_1}(\mathcal{F}) + \sqrt{2\ln(4\delta^{-1})/k_1} + 6B_{\mathcal{X}}C_{\mathcal{F}}\sqrt{2d\ln(4d(k_1+k_2)\delta^{-1})/h}$$

with probability at least $1-\delta$. In particular, the test loss approaches to 0 when $k_1, h \to \infty$.

Proof. We fix the feature generation function to be GroupByThenMean. In concrete, we have $\hat{X}_i = H(G_{\mathrm{g}(i)}) = \frac{1}{|G_{\mathrm{g}(i)}|}\sum_{j\in G_{\mathrm{g}(i)}} X_j$. The algorithm then find f that minimizes the emprical risk:

$$\hat{f} = \arg\min_{f\in\mathcal{F}} L_{\mathcal{D}_{\text{train}}}(H, f).$$

According to Hoeffding's inequality and union bound for each dimension, with probability at least $1 - \delta/(2(k_1+k_2))$, we have

$$\|\widehat{X}_i - Z_i\| = \left\| \frac{1}{|G_{\mathrm{g}(i)}|} \sum_{j\in G_{\mathrm{g}(i)}} X_j - \mathop{\mathbb{E}}_{X\sim\mathcal{P}_{\mathrm{g}(i)}}[X] \right\| \le B_{\mathcal{X}}\sqrt{2d\ln(4d(k_1+k_2)\delta^{-1})/h},$$

for some concrete $\mathrm{g}(i)$. By applying union bound on all groups, this statement holds for all i with probability at least $1-\delta/2$.

In this case, for any function $f \in \mathcal{F}$, we have

$$\begin{aligned}
&\left|(f(X_i, Z_i) - Y_i)^2 - (f(X_i, \hat{X}_i) - Y_i)^2\right| \\
\le\ & |f(X_i, Z_i) - f(X_i, \hat{X}_i)| \cdot \left|f(X_i, Z_i) - Y_i + f(X_i, \hat{X}_i) - Y_i\right| \\
\le\ & 2B_{\mathcal{X}}C_{\mathcal{F}}\sqrt{2d\ln(4d(k_1+k_2)\delta^{-1})/h}
\end{aligned} \tag{2}$$

where the last inequality holds since $|f(X_i, Z_i) - f(X_i, \hat{X}_i)| \le C_{\mathcal{F}}\|\widehat{z}_i - z_i\|$ and $\mathcal{Y} \subseteq [0,1]$. Define H^* be the function $H^*(G_{\mathrm{g}(i)}) = Z_i = \mathbb{E}_{X\sim\mathcal{P}_{\mathrm{g}(i)}}[X]$. We note this statement further implies $\left|L_{\mathcal{D}_{\text{train}}}(H^*, f) - L_{\mathcal{D}_{\text{train}}}(H, f)\right| \le \sqrt{2d\ln(4d(k_1+k_2)\delta^{-1})/h}$ and $\left|L_{\mathcal{D}_{\text{test}}}(H^*, f) - L_{\mathcal{D}_{\text{test}}}(H, f)\right| \le \sqrt{2d\ln(4d(k_1+k_2)\delta^{-1})/h}$ according to the definition of the loss function.

Note that each data point is not an i.i.d. sample (since two points in one group are correlated), we need to define the following group Rademacher complexity to bound the generalization error. We define *group Rademacher complexity* to be Rademacher complexity but defined over groups:

$$\mathrm{Rad}_{k_1}^G(\mathcal{F}) := \mathop{\mathbb{E}}_{\mathcal{D}_{\text{train}}} \frac{1}{k_1} \mathop{\mathbb{E}}_{\sigma} \left[\sup_{f\in\mathcal{F}} \sum_{i=1}^{k_1} \sigma_i \cdot \frac{1}{h} \sum_{j\in G_i} L(H, f, X_j, Y_j) \right],$$

where $\sigma = (\sigma_1, \cdots, \sigma_{k_1})$ are independent Rademacher variables. Note that if we view each group as one random sample, these groups are i.i.d. samples. Hence, we can apply the classical generalization bound via Rademacher complexity (Mohri et al., 2018), which asserts that with probability at least $1-\delta/2$ for any function $f \in \mathcal{F}$ the following holds

$$\left|L_{\mathcal{D}_{\text{test}}}(H, f) - L_{\mathcal{D}_{\text{train}}}(H, f)\right| \le 2\mathrm{Rad}_{k_1}^G(\mathcal{F}) + \sqrt{2\ln(4\delta^{-1})/k_1}. \tag{3}$$

Moreover, one can see the group Rademacher complexity can be upper bounded by the ordinary Rademacher complexity for i.i.d. samples:

$$\mathrm{Rad}_{k_1}^{G}(\mathcal{F}) \leq \frac{1}{h}\sum_{i=1}^{h} \mathop{\mathbb{E}}_{\mathcal{D}_{\text{train}}} \frac{1}{k_1} \mathop{\mathbb{E}}_{\sigma}\left[\sup_{f\in\mathcal{F}} \sum_{j=1}^{k_1} \sigma_j \cdot L(H, f, X_{G_{i,j}}, Y_{G_{i,j}})\right] = \mathrm{Rad}_{k_1}(\mathcal{F}) \tag{4}$$

where $G_{i,j}$ is the j-th element in G_i.

According to union bound, both (2) and (3) are satisfied for all $f \in \mathcal{F}$ with probability at least $1-\delta$. In this case, since f^* is the ground truth, its empirical risks satisfies $L_{\mathcal{D}_{\text{train}}}(H^*, f^*) = 0$. In this case, we have

$$\begin{aligned} L_{\mathcal{D}_{\text{train}}}(H, \hat{f}) &\leq L_{\mathcal{D}_{\text{train}}}(H, f^*) \\ &\leq \left|L_{\mathcal{D}_{\text{train}}}(H, f^*) - L_{\mathcal{D}_{\text{train}}}(H^*, f^*)\right| + L_{\mathcal{D}_{\text{train}}}(H^*, f^*) \\ &\leq 2B_{\mathcal{X}}C_{\mathcal{F}}\sqrt{2d\ln(4d(k_1+k_2)\delta^{-1})/h} \end{aligned} \tag{5}$$

where the first inequality dues to empirical risk minimization and the last inequality dues to (2). As a result, the population loss can then be bounded by

$$\begin{aligned} L_{\mathcal{D}_{\text{test}}}(H, \hat{f}) \leq& \left|L_{\mathcal{D}_{\text{test}}}(H, \hat{f}) - L_{\mathcal{D}_{\text{test}}}(H^*, \hat{f})\right| + \left|L_{\mathcal{D}_{\text{test}}}(H^*, \hat{f}) - L_{\mathcal{D}_{\text{train}}}(H^*, \hat{f})\right| \\ &+ \left|L_{\mathcal{D}_{\text{train}}}(H^*, \hat{f}) - L_{\mathcal{D}_{\text{train}}}(H, \hat{f})\right| + L_{\mathcal{D}_{\text{train}}}(H, \hat{f}) \\ \leq& 2B_{\mathcal{X}}C_{\mathcal{F}}\sqrt{2d\ln(4d(k_1+k_2)\delta^{-1})/h} + 2\mathrm{Rad}_{k_1}^{G}(\mathcal{F}) + \sqrt{2\ln(4\delta^{-1})/k_1} \\ &+ 2B_{\mathcal{X}}C_{\mathcal{F}}\sqrt{2d\ln(4d(k_1+k_2)\delta^{-1})/h} + 2B_{\mathcal{X}}C_{\mathcal{F}}\sqrt{2d\ln(4d(k_1+k_2)\delta^{-1})/h} \\ \leq& 2\mathrm{Rad}_{k_1}(\mathcal{F}) + \sqrt{2\ln(4\delta^{-1})/k_1} + 6B_{\mathcal{X}}C_{\mathcal{F}}\sqrt{2d\ln(4d(k_1+k_2)\delta^{-1})/h} \end{aligned}$$

where the second inequality holds due to (3) and (5) while the last inequality follows from (4). This proves the desired statement. □

Remarks: We only consider the case where $\mathcal{H}$ only contain one particular operation $\mathrm{GroupByThenMean}$. In fact, it is possible to extend the above setting to one with multiple feature generation functions. Here we require that such feature generation functions be statistics of the corresponding group and can be estimated using i.i.d. samples. In our two-phase generative process, each group contains exactly h data points. We can easily extend our theorem to the setting where the size of each group is also a random variable as long as it takes value at least h with high probability. The main idea is the same but the notations would become very tedious. Since our goal here is to illustrate the statistical advantage of feature generation, we choose to present a simplified yet representative setting.

Without Feature Generation

Theorem A.2. *In case that we do not use any feature generation, there exists a problem instance such that, no matter how large k_1 (number of groups in the training set), k_2 ((number of groups in the test set)), and h (the size of each group) are, for any function $f' : \mathcal{X} \to \mathcal{Y}$, the test loss is at least*

$$L_{\mathcal{D}_{test}}(f') \geq \frac{3}{64}.$$

Proof. Consider a problem instance with $\mathcal{X} = \mathcal{Y} = [0, 1]$. The data distribution is generated in the following way: Each group contains h data points and is generated in the following way: The distribution of i-th group $\mathcal{P}_{\mathbf{g}(i)}$ is random between $B(\frac{3}{4})$ and $B(\frac{1}{4})$ with equal probability where $B(p)$ is the Bernoulli distribution such that $\Pr[B(p) = 1] = p = 1 - \Pr[B(p) = 0]$. Thus, $Z_i = \mathbb{E}_{X\sim\mathcal{P}_{\mathbf{g}(i)}}[X]$ is either $\frac{3}{4}$ or $\frac{1}{4}$ with equal probability for each group. X_i is 0/1 random variable, generated i.i.d. from either $B(\frac{3}{4})$ if $Z_i = \frac{3}{4}$ and from either $B(\frac{1}{4})$ if $Z_i = \frac{1}{4}$. We assume the target of data point is given by $Y_i = f^*(X_i, Z_i) = Z_i$.

When $X_i = 1$, the probability of $Y_i = \frac{1}{4}$ is given by

$$\Pr\left[Y_i = \frac{1}{4}\middle|X_i = 1\right] = \Pr\left[Z_i = \frac{1}{4}\middle|X_i = 1\right] = \frac{\Pr[Z_i = \frac{1}{4}, X_i = 1]}{\sum_z \Pr[Z_i = z]\Pr[X_i = 1|Z_i = z]} = \frac{1}{4}.$$

Similarly, the probability of $Y_i = \frac{3}{4}$ is $\Pr[Y_i = \frac{3}{4}|X_i = 1] = \frac{3}{4}$. So for any predictor $f' : \mathcal{X} \to \mathcal{Y}$, we have

$$\mathop{\mathbb{E}}_{X_i=1}[\|Y_i - f'(X_i)\|^2] = \frac{1}{4} \cdot \left(f'(X_i) - \frac{1}{4}\right)^2 + \frac{3}{4} \cdot \left(f'(X_i) - \frac{3}{4}\right)^2 \geq \frac{3}{64}.$$

The last inequality holds because the left hand side is a quadratic function. With the same reasoning, we have $\mathbb{E}_{X_i=0}[\|Y_i - f'(X_i)\|^2] \geq \frac{3}{64}$. As a result, the test loss of for any predictor f' is at least

$$L_{\mathcal{D}_{\text{test}}}(f') = \mathop{\mathbb{E}}_{(X_i,Y_i)\in\mathcal{D}_{\text{test}}}[\|Y_i - f'(X_i)\|^2] \geq \frac{3}{64}.$$

The above argument does not depend on how large k_1, k_2 and h are. □

B. Data

Table 8: Datasets description

Name	Abbr	# Train	# Validation	# Test	# Num	# Cat	# Ord	Task type
California Housing (Pace & Barry, 1997)	CA	13209	3303	4128	7	0	1	Regression
Microsoft (Qin & Liu, 2013)	MI	723412	235259	241521	111	0	25	Regression
Medical	ME	104361	26091	32613	5	6	0	Regression
Telecom	TE	32669	8168	10210	23	22	12	Binclass
Broken Machine	BR	576000	144000	180000	58	0	0	Binclass
Diabetes (Strack et al., 2014)	DI	65129	16283	20354	3	34	10	Binclass
Nomao (Candillier & Lemaire, 2012)	NO	22465	6000	6000	34	29	55	Binclass
Vehicle (Siebert, 1987)	VE	60000	18528	20000	100	0	0	Binclass
Jannis (Guyon et al., 2019)	JA	53588	13398	16747	54	0	0	Multiclass
Covertype (Blackard & Dean, 1999)	CO	371847	92962	116203	9	0	45	Multiclass

We describe the details of the datasets in Table 8. The reason we did not perform evaluation on the datasets used by previous studies (Katz et al., 2016; Chen et al., 2019; Zhu et al., 2022; Li et al., 2023) is that, these datasets only contain a few hundreds to a few thousand data points with a few dozens of features. Experimental results on toy datasets are hardly convincing for the real progress in feature generation. The datasets used in our benchmark range in size from moderate to big, with some having millions of samples and hundreds of features.

C. Additional Results

C.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.

Table 9: Comparison between OpenFE and other baselines. The results of baseline methods are from the corresponding papers. 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**

- **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 9. OpenFE also surpasses other baseline methods in these datasets.

Table 10: Results in Table 4 with standard deviations.

Model	Features	CA ↓	MI ↓	NO ↑	VE ↑	JA ↑
AutoInt	Base	$0.474_{\pm 0.004}$	$0.750_{\pm 0.001}$	$0.993_{\pm 0.000}$	$0.927_{\pm 0.001}$	$0.720_{\pm 0.002}$
AutoInt	OpenFE	$\mathbf{0.462}_{\pm 0.005}$	$\mathbf{0.745}_{\pm 0.000}$	$\mathbf{0.994}_{\pm 0.000}$	$\mathbf{0.929}_{\pm 0.001}$	$\mathbf{0.724}_{\pm 0.003}$
DCN-V2	Base	$0.483_{\pm 0.002}$	$0.749_{\pm 0.001}$	$0.993_{\pm 0.000}$	$0.924_{\pm 0.001}$	$0.716_{\pm 0.002}$
DCN-V2	OpenFE	$\mathbf{0.472}_{\pm 0.004}$	$\mathbf{0.743}_{\pm 0.000}$	$\mathbf{0.994}_{\pm 0.000}$	$\mathbf{0.926}_{\pm 0.000}$	$0.716_{\pm 0.001}$
TabNet	Base	$0.510_{\pm 0.005}$	$0.751_{\pm 0.001}$	$0.993_{\pm 0.000}$	$0.924_{\pm 0.001}$	$0.724_{\pm 0.004}$
TabNet	OpenFE	$\mathbf{0.501}_{\pm 0.009}$	$\mathbf{0.746}_{\pm 0.001}$	$\mathbf{0.995}_{\pm 0.000}$	$\mathbf{0.926}_{\pm 0.001}$	$0.724_{\pm 0.004}$
NODE	Base	$0.464_{\pm 0.002}$	$0.745_{\pm 0.000}$	$0.994_{\pm 0.000}$	$0.927_{\pm 0.000}$	$0.727_{\pm 0.002}$
NODE	OpenFE	$\mathbf{0.457}_{\pm 0.002}$	$\mathbf{0.740}_{\pm 0.000}$	$\mathbf{0.995}_{\pm 0.001}$	$\mathbf{0.929}_{\pm 0.000}$	$\mathbf{0.731}_{\pm 0.001}$
FT-T	Base	$0.459_{\pm 0.003}$	$0.746_{\pm 0.000}$	$0.993_{\pm 0.000}$	$0.927_{\pm 0.001}$	$0.733_{\pm 0.002}$
FT-T	OpenFE	$\mathbf{0.453}_{\pm 0.003}$	$\mathbf{0.741}_{\pm 0.001}$	$\mathbf{0.995}_{\pm 0.000}$	$\mathbf{0.929}_{\pm 0.000}$	$\mathbf{0.738}_{\pm 0.002}$

C.2. Results with Standard Deviations

We present the results of the effect of OpenFE on DNNs with standard deviations in Table 10.

C.3. Additional Experiments on OpenML CC18 Benchmark

We expand our evaluation using the curated OpenML CC18 benchmarking suites (Bischl et al., 2021), which consist of 68 tabular datasets. The Three conclusions can be drawn from the additional experimental results:

1. Feature generation leads to statistically significant improvements over base features in learning performance on 49 out of 68 datasets.
2. OpenFE outperforms all baseline methods significantly on 40 out of these 49 datasets.

Table 11: Comparison between the results of FCTree, SAFE, and AutoCross from their paper and our reproduced results to verify the reproduction.

	Metric	FCTree (from paper)	FCTree (reproduced)
Transfusion	Accuracy	0.752	0.793 ± 0.017
Nuclear	AUC	0.629	0.625 ± 0.009
	Metric	SAFE (from paper)	SAFE (reproduced)
Magic	AUC	0.9288	0.9370 ± 0.0009
Spambase	AUC	0.9846	0.9837 ± 0.0012
	Metric	AutoCross (from paper)	AutoCross (reproduced)
Bank	AUC	0.9455	0.9456 ± 0.0008
Adult	AUC	0.9280	0.9251 ± 0.0003
Credit	AUC	0.8567	0.8624 ± 0.0002

3. OpenFE provides an average accuracy improvement of 1.9% over the base features across these 49 datasets.

We present the results in Table 7.

Regarding the experimental setup: For each dataset, we use the same 0.6/0.2/0.2 split (train/val/test) for all methods. The evaluation metric across all datasets is accuracy. LightGBM is chosen as the modeling algorithm for the tabular data, and we tune its hyperparameters using the validation set and the base features. We repeat each experiment 10 times and apply Welch's t-test with unequal variance and a p-value of 0.05 to assess the significance.

Regarding standard deviation: The standard deviation of results are generally small because LightGBM training is typically robust to different random seeds, ensuring consistent performance across multiple trials. In the table below, the standard deviation shown as 0.000 indicates that it is less than 0.0005.

C.4. 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 reproduced methods with the ones in their paper. We present the results in Table 11. We can see that most of the results of the reproduced methods closely match or even outperform the results from the papers.

Table 12: The running time of different methods in minutes. IEEE: IEEE-CIS Fraud Detection, BNP: BNP Paribas Cardif Claims Management.

	CA	MI	ME	TE	BR	DI	NO	VE	JA	CO	IEEE	BNP
FCTree	2.3	1378	6.9	5.7	330	6.8	11	74	40	160	-	-
SAFE	-	-	-	5	6.0	0.9	1.3	10	-	-	-	-
AutoFeat	0.2	23	20.6	32.1	537	37	49	535	354	1284	-	-
AutoCross	-	-	-	101	1078	169	148	146	-	-	-	-
FETCH	98.1	-	1202	150	-	241	325	1417	528	-	-	-
OpenFE	0.1	31	1.5	2.1	4.7	4	4.7	3.3	3.5	20	92	0.9

C.5. Running Time

We present the running time of different methods in Table 12. The experimental environment is the same for all the methods. We can see that OpenFE is a fast method that terminates within a reasonable amount of time even for large datasets.

Table 13: Comparisons between MDI, permutation, and SHAP in feature importance attribution.

	MDI	permutation	SHAP
Rank	$2.07_{\pm 0.93}$	$1.79_{\pm 0.91}$	$2.14_{\pm 0.69}$
Time	0s	25min	42s

C.6. 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 benchmarking datasets to rank these methods. The training model is LightGBM. We present the results in Table 13. We also present the running time of each method on the Microsoft dataset, 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 after the training process of LightGBM, while permutation and SHAP may require longer running time, depending on the sizes of datasets.

D. Implementation Details

D.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 are performed on one or more NVidia Tesla V100 16Gb.

Table 14: Unary operators.

Numerical	Categorical
$\text{Freq}, \text{Abs}, \text{Log},$ $\text{Sqrt}, \text{Sigmoid},$ $\text{Round}, \text{Residual}$	Freq

Table 15: Binary operators. (N refers to Numerical, C refers to Categorical)

N $\times$ N	N $\times$ C	C $\times$ C
$\min, \max,$ $+, -, \times, \div$	$\text{GroupByThenMin}, \text{GroupByThenMax},$ $\text{GroupByThenMean}, \text{GroupByThenMedian},$ $\text{GroupByThenStd}, \text{GroupByThenRank}$	$\text{Combine},$ $\text{CombineThenFreq},$ $\text{GroupByThenNUnique}$

D.2. Operators and Feature Transformations

All the operators are classified into unary operators (Table 14) and binary operators (Table 15), where unary operators act on one feature and binary operators act on two features. Then the operators are further classified according to the type of features they act 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 transformations. 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 of 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 enumerate all the first-order transformations to form the candidate feature set. A first-order transformation uses one operator once to transform base features. For example, weight/height is a first-order transformation of base features weight and height. BMI = weight/height^2 is a second-order transformation.

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

- $\mathrm{Freq}(f)$. The frequency of feature f.
- $\mathrm{Freq}(c)$. The frequency of feature c.
- $\mathrm{Abs}(f)$. Element-wise absolute value.
- $\mathrm{Log}(f)$. Element-wise logarithm.
- $\mathrm{Sqrt}(f)$. Element-wise square root.
- $\mathrm{Sigmoid}(f)$. Element-wisely apply function $x \mapsto \frac{1}{1+e^{-x}}$.
- $\mathrm{Round}(f)$. Element-wise rounding.
- $\mathrm{Residual}(f)$. Element-wisely take decimal part.
- $\mathrm{Min}(f_1, f_2)$. Element-wise minimum.
- $\mathrm{Max}(f_1, f_2)$. 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.
- $\mathrm{GroupByThenMin}(f, c)$. The minimum value of f in each category of feature c.
- $\mathrm{GroupByThenMax}(f, c)$. The maximum value of f in each category of feature c.
- $\mathrm{GroupByThenMean}(f, c)$. 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 data point, "vocation" is "doctor" and "hobby" is "football", then the value for $\mathrm{Combine}(\text{vocation}, \text{hobby})$ of this data point is a new 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.

Table 16: The parameters of OpenFE for each dataset. IEEE: IEEE-CIS Fraud Detection. BNP: BNP Paribas Cardif Claims Management.

	CA	MI	ME	TE	BR	DI	NO	VE	JA	CO	IEEE	BNP
# data blocks	1	64	1	1	1	8	32	16	1	8	32	8
# new features	10	10	10	10	10	10	10	10	50	50	600	200

D.3. Parameters

We present the hyperparameters of OpenFE for each dataset in Table 16. When the number of data blocks is one, we use all the data to calculate and evaluate new features in successive featurewise pruning, and eliminate new features with negative reduction in loss. For most 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. For a fair comparison, all the baseline methods include the same number of new features, except for the **NN** baseline. For the **NN** baseline, the number of new features is the number of hidden units in the last hidden layer, which is determined by hyperparameter tuning.

In OpenFE, the default parameter of LightGBM in successive featurewise pruning has 1000 number of estimators, 0.1 learning rate, 16 leaves, and 3 early stopping rounds. The default parameter in feature importance attribution is the same except for 50 early stopping rounds.

D.4. Feature Importance Attribution Methods

MDI is the gain importance embedded in LightGBM. For permutation feature importance, we randomly shuffle the values of features on the validation set, and observe the change in validation loss. For SHAP feature importance, we average the SHAP values of each sample for each feature.

E. Discussion

E.1. Discovered Feature Transformations

We discuss the effective feature transformations discovered by OpenFE for the DI (Diabetes) dataset. The DI dataset collects 10 years (1999-2008) of clinical care at 130 US hospitals, where the goal is to predict the readmission (Strack et al., 2014). One can see from Table 3 that OpenFE outperforms other baseline methods and improves the learning performance by a great margin. The top-1 feature discovered by OpenFE is freq(patient_id), which is the number of times the patient has been admitted to the hospital and is highly predictive of whether the patient will be readmitted to the hospital. However, other methods may fail to find this new feature since the feature patient_id itself is not useful. For example, SAFE (Shi et al., 2020) only generates candidate features based on the base features that are used as splits in XGBoost. However, since patient_id itself is not useful, it will not be used for splits in XGBoost. This example also demonstrates that, reducing the number of candidate features by heuristic assumptions may risk missing useful candidate features.

E.2. High-order Features

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 previous methods argue that high-order features are useful by directly searching for high-order features (Chen et al., 2019; Khurana et al., 2018). 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 their 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. In a word, searching for high-order features in a hierarchical manner is more appropriate than directly searching for high-order features in two aspects: 1) evaluating the effectiveness of high-order features in a more accurate way. 2) generating low-order features with better interpretability.

Whether it is necessary to generate high-order features is case-by-case. Because the search space of high-order features is usually incredibly huge (even if we limit the order), none of the existing methods can enumerate all the high-order features within reasonable computational resources. We can hardly claim that high-order features are not useful for a dataset. However, we do not find that generating high-order features is useful for all the benchmarking datasets in our experiments. In the IEEE competition, generating high-order features does not seem to be useful for neither Expert nor OpenFE. In the BNP competition, generating high-order features provides a small improvement in the test score. We may conclude that first-order features are usually more important than high-order ones in feature generation.

F. Complexity Analysis

Complexity of Generating Base Predictions. Let n be the number of samples, m be the number of base features of dataset, and k be the number of folds. Complexity of generating base predictions is k times of GBDT[2] training, hence

$$O\left(kTdnm\right),$$

where each GBDT contains T trees with a maximum depth d.

Complexity of STAGE I. Suppose we split the dataset into 2^q data blocks, the number of candidate features is m^2. There are $2^{-q}m^2$ features remaining after successive featurewise halving. The complexity is $\sum_{i=0}^{q} 2^{(i-q)} \cdot m^2 \cdot C(2^{-i}n, 1)$, where $C(n, m)$ is the complexity of GBDT training with data shape $n \times m$. If the GBDT contains T_1 trees with a maximum depth of d_1, we have the time complexity

$$O\left(2^{-q}qT_1d_1nm^2\right).$$

Complexity of STAGE II. The complexity of stage II is dominated by a single GBDT training. Suppose the GBDT has T_2 trees with a maximum depth of d_2, then the time complexity of stage II is

$$O\left(2^{-q}T_2d_2nm^2\right).$$

Overall Complexity. In the implementation of OpenFE, the number of trees and the maximum depth of GBDT are predefined fixed constant. If we regard T and d as constant, the overall complexity of OpenFE is

$$O(2^{-q}qnm^2).$$

[2]In complexity analysis, we refer to the implementation of LightGBM. The dominant term in complexity is building histogram, i.e., $O(nm)$ per depth per tree.