AutoCross: Automatic Feature Crossing for Tabular Data in Real-World Applications

Yuanfei Luo
luoyuanfei@4paradigm.com
4Paradigm Inc., Beijing, China

Mengshuo Wang
wangmengshuo@4paradigm.com
4Paradigm Inc., Beijing, China

Hao Zhou
zhouhao@4paradigm.com
4Paradigm Inc., Beijing, China

Quanming Yao+
yaoquanming@4paradigm.com
4Paradigm Inc., Beijing, China

Wei-Wei Tu*
tuweiwei@4paradigm.com
4Paradigm Inc., Beijing, China

Yuqiang Chen
chenyuqiang@4paradigm.com
4Paradigm Inc., Beijing, China

Qiang Yang
qyang@cse.ust.hk
Hong Kong University of Science and Technology, Hong Kong

Wenyuan Dai
daiwenyuan@4paradigm.com
4Paradigm Inc., Beijing, China

ABSTRACT

Feature crossing captures interactions among categorical features and is useful to enhance learning from tabular data in real-world businesses. In this paper, we present AutoCross, an automatic feature crossing tool provided by 4Paradigm to its customers, ranging from banks, hospitals, to Internet corporations. By performing beam search in a tree-structured space, AutoCross enables efficient generation of high-order cross features, which is not yet visited by existing works. Additionally, we propose successive mini-batch gradient descent and multi-granularity discretization to further improve efficiency and effectiveness, while ensuring simplicity so that no machine learning expertise or tedious hyper-parameter tuning is required. Furthermore, the algorithms are designed to reduce the computational, transmitting, and storage costs involved in distributed computing. Experimental results on both benchmark and real-world business datasets demonstrate the effectiveness and efficiency of AutoCross. It is shown that AutoCross can significantly enhance the performance of both linear and deep models. [1]

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**;

KEYWORDS

AutoML, Feature Crossing, Tabular Data

ACM Reference Format:
Yuanfei Luo, Mengshuo Wang, Hao Zhou, Quanming Yao+, Wei-Wei Tu*, Yuqiang Chen, Qiang Yang, and Wenyuan Dai. 2019. AutoCross: Automatic Feature Crossing for Tabular Data in Real-World Applications. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, Anchorage, Alaska, USA, 12 pages. https://doi.org/10.1145/3292500.3330679

[1] Y. Luo and M. Wang make equal contribution to this work; Q. Yao and W.-W. Tu are the corresponding authors.

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330679

1 INTRODUCTION

Recent years have seen the emergence of the automated machine learning (AutoML) [40, 43] as a promising way to make machine learning techniques easier to use, so that the manpower heavily involved in the current applications could be spared, and greater social and commercial benefits could be made. In particular, AutoML aims to automate some parts or the whole of the machine learning pipeline, which usually comprises data preprocessing, feature engineering, model selection, hyper-parameter tuning, and model training. It has been well recognized that the performance of machine learning methods depends greatly on the quality of features [9, 28, 31]. Since raw features rarely lead to satisfying results, manual feature generation is often carried out to better represent the data and improve the learning performance. However, it is often a tedious and task-specific work. This motivates automatic feature generation [2, 5, 6, 19, 21, 26, 33, 34, 42], one major topic of AutoML, where informative and discriminative features are automatically generated. In 4Paradigm, a company with the aspiration to make machine learning techniques accessible to more people, we also make efforts on this topic. We provide simple yet powerful feature generation tools to organizations and companies without machine learning expertise, and enable them to better exploit their data.

The customers of 4Paradigm range from banks, hospitals, to various Internet Corporations. In their real-world businesses, e.g., fraud detection [4, 38], medical treatment [23], online advertising [10, 41] or recommendation [3], etc., the majority of gathered data is presented in the form of tables, where each row corresponds to an instance and each column to a distinct feature. Such data is often termed as *tabular data*. Furthermore, in such data, a considerable part of features are *categorical*, e.g., '`job`' and '`education`' to describe the occupation and education status of an individual, respectively. These features, indicating an entity or describing some important attributes, are very important and informative. In order to make better use of them, many feature generation methods have

	age (n)	job (c)	marital (c)	education (c)	balance (n)	housing (c)
0	30	unemployed	married	primary	1787	no
1	33	services	married	secondary	4789	yes
2	35	management	single	tertiary	1350	yes
3	30	management	married	tertiary	1476	yes
4	59	blue-collar	married	secondary	0	yes
5	35	management	single	tertiary	747	no

Figure 1: An example of tabular data (UCI-Bank). The letters embraced in the parentheses indicate the feature types, with 'n' standing for 'numerical' and 'c' for 'categorical'.

been proposed recently [5, 7, 14, 19, 26, 33, 34, 42]. In this paper, we also aim to increase learning performance by exploiting categorical features.

Feature crossing, taking cross-product of sparse features, is a promising way to capture the interaction among categorical features and is widely used to enhance learning from tabular data in real-world businesses [5, 7, 26, 34, 37]. The results of feature crossing are called *cross features* [26, 37], or conjunction features [5, 34]. Feature crossing represents the co-occurrence of features, which may be highly correlated with the target label. For example, the cross feature '`job` ⊗ `company`' indicates that an individual takes a specific job in a specific company, and is a strong feature to predict one's income. Feature crossing also adds nonlinearity to data, which may improve the performance of learning methods. For instance, the expressive power of linear models is restricted by their linearity, but can be extended by cross features [5, 34]. Last but not least, explicitly generated cross features are highly interpretable, which is an appealing characteristic in many real-world businesses, such as medical treatment and fraud detection.

However, to enumerate all cross features may lead to degraded learning performance, since they may be irrelevant or redundant, introduce noise, and increase the difficulty of learning. Hence, only useful and important cross features should be generated, but they are often task-specific [5, 26, 34, 37]. In traditional machine learning applications, human experts are heavily involved in feature engineering, striving to generate useful cross features for every task, with their domain-knowledge in a trial-and-error manner. Furthermore, even experienced experts may have trouble when the number of original features is large. The manpower requirement and difficulty of manual feature crossing greatly increase the total cost to apply machine learning techniques, and even hinder many companies and organizations to make good use of their data.

This raises the great demand for *automatic feature crossing*, the target of our work presented in this paper. In addition to our main target, i.e., to automate feature crossing with high effectiveness and efficiency, we need to consider several more requirements: 1) *simplicity requirement*: a tool with high simplicity is user-friendly and easy to use. The performance of most existing automatic feature generation methods greatly depends on some hyper-parameters. Examples are the thresholds in ExploreKit [21], subsampling ratio in [5], and network architectures in deep-learning-based methods [7, 14, 26, 33, 42]. These hyper-parameters should be properly determined or carefully tuned by the user for each specific task. Since no machine learning expertise of our customers is assumed, hyper-parameters that require careful setting or fine-tuning should be avoided. 2) *distributed computing*: the large amount of data and features in real-world businesses makes distributed computing a must. Feature crossing methods should take into consideration the corresponding computational, transmitting and storage costs. 3) *real-time inference requirement*: real-time inference is involved in many real-world businesses. In such cases, once an instance is inputted, the feature should be produced instantly and the prediction made. Latency requirement for real-time inference is rigorous [8, 13], which raises the need for fast feature producing.

To summarize our business requirements, we need our automatic feature crossing tool to have high *effectiveness*, *efficiency* and *simplicity*, be optimized for *distributed computing*, and enable *fast inference*. To address these challenges, we present AutoCross, an automatic feature crossing tool especially designed for tabular data with considerable categorical features. The major contributions of this paper are summarized as follows:

- We propose an efficient AutoML algorithm to explicitly search for useful cross features in an extensive search space. It enables to construct *high-order* cross features, which can further improve the learning performance but is not yet visited by existing works.
- AutoCross features high simplicity with minimized exposure of hyper-parameters. We propose successive mini-batch gradient descent and multi-granularity discretization. They improve the efficiency and effectiveness of feature crossing while avoiding careful hyper-parameter setting.
- AutoCross is fully optimized for distributed computing. By design, the algorithms can reduce the computational, transmitting, and storage costs.
- Extensive experimental results on both benchmark and real-world business datasets are reported to verify the effectiveness and efficiency of AutoCross. It can significantly improve the performance of generalized linear models, while keeping a low inference latency. It is also shown that AutoCross can accompany deep models, by which means we can combine the advantage of explicit and implicit feature generation and further improve the learning performance.

With the these characteristics, AutoCross enables our customers to make better use of their data in real-world businesses with little learning and using costs.

The remaining of this paper is organized as follows: in Section 2, we demonstrate what motivates us to develop our own feature crossing tool; next, in Section 3, the overall system is introduced; the techniques employed in each component, as well as the choices made in designing the system, are detailed in Section 4; experimental results are presented in Section 5; Section 6 reviews some related works and Section 7 concludes this paper.

2 MOTIVATION

Cross features are constructed by vectorizing the cross-product (⊗) of original features:

$$\mathbf{c}_{i,j,\cdots,k} = \mathsf{vec}\left(\mathbf{f}_i \otimes \mathbf{f}_j \otimes \cdots \otimes \mathbf{f}_k\right), \tag{1}$$

Table 1: Comparison between AutoCross and other feature generation methods for tabular data.

Method	High-order Feature Cross	Simplicity	Fast Inference	Interpretability
Search-based methods (e.g., [5, 34])	×	medium	√	√
Implicit deep-learning-based methods (e.g., [33, 42])	×	low	×	×
Explicit deep-learning-based methods (e.g., [26, 37])	×	low	×	√
AutoCross	√	high	√	√

where $\mathbf{f}_i$'s are binary feature vectors (generated by one-hot encoding or hashing trick) and $\mathrm{vec}(\cdot)$ vectorizes a tensor. A cross feature is also a binary feature vector. If a cross feature uses three or more original features, we denote it as a *high-order* cross feature. In this section, we state motivation of our work: why we consider high-order cross features and why existing works do not suit our purpose.

While most early works of automatic feature generation focus on second-order interactions of original features [5, 6, 19, 21, 34], trends have appeared to consider higher-order (i.e., with order higher than two) interactions to make data more informative and discriminative [2, 26, 33, 42]. High-order cross features, just like other high-order interactions, can further improve the quality of data and increase predictive power of learning algorithms. For example, a third-order cross feature '`item ⊗ time ⊗ region`' can be a strong feature to recommend regionally preferred food during certain festivals. However, explicit generation of high-order cross features is not yet visited in existing works. In the remaining of this section, we demonstrate that existing feature generation approaches either do not generate high-order cross features or cannot fulfill our business requirements.

On the one hand, *search-based feature generation methods* employ explicit search strategies to construct useful features or feature sets. Many such methods focus on numerical features [11, 20, 21, 35, 36], and do not generate cross features. As for existing feature crossing methods [5, 34], they are not designed, and are hence inefficient, to perform high-order feature crossing.

On the other hand, there are *deep-learning-based feature generation approaches*, where interactions among features are implicitly or explicitly represented by specific networks. Variants of deep learning models are proposed to deal with categorical features (e.g., Factorisation-machine supported neural networks [42] and Product-based neural networks [33]). Efforts are also made to accompany deep learning models with additional structures that incorporate: 1) manually designed features (e.g., Wide & Deep [7]); 2) factorization machines (e.g., DeepFM [14] and xDeepFM [26]), and/or 3) other feature construction components (e.g., Deep & Cross Network [37] and xDeepFM [26]). Especially, xDeepFM [26], proven superior to other deep-learning-based approaches mentioned above, proposed a compressed interaction network (CIN) to explicitly capture high-order interactions among embedded features. This is done by performing entry-wise product on them:

$$\mathbf{e}_{i,j,\cdots,k} = \mathbf{W}_i\mathbf{f}_i \circ \mathbf{W}_j\mathbf{f}_j \circ \cdots \circ \mathbf{W}_k\mathbf{f}_k, \tag{2}$$

where $\circ$ denotes the entry-wise product (Hadamard product) and $\mathbf{W}_i$'s the embedding matrices so that $\mathbf{Wf} \in \mathbb{R}^D$. Different embedding matrices lead to different interaction $\mathbf{e}$'s. However, as stated in the following proposition, the resulting features in Equation (2) is only a special case of embedded high-order cross features.

Proposition 1. *There exist infinitely many embedding matrices* $\mathbf{C}$*'s with* D *rows so that: there do not exist any embedding matrices* $\mathbf{A}$ *and* $\mathbf{B}$ *that satisfy the following equation:*

$$\mathbf{Ax} \circ \mathbf{By} = \mathbf{Cz}, \tag{3}$$

for all binary vectors $\mathbf{x}$, $\mathbf{y}$ *and their crossing* $\mathbf{z}$.

The proof can be found in Appendix A. Furthermore, deep models are relatively slow in inference , which makes model compression [17] or other accelerating techniques necessary to deploy them in many real-time inference systems.

Motivated by the usefulness of high-order cross features and the limitation of existing works, in this paper, we aim to propose a new automatic feature crossing method that is efficient enough to generate high-order cross features, while satisfying our business requirements, i.e., to be simple, optimized for distributed computing, and enable fast inference. Table 1 compares AutoCross and other existing methods.

3 SYSTEM OVERVIEW

Figure 2 gives an overview of AutoCross, comprising three parts: 1) the general work flow; 2) the component algorithms; and 3) the underlying infrastructure.

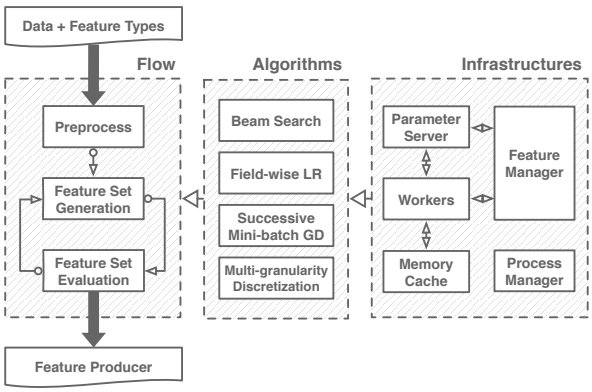

Figure 2: System overview of AutoCross.

From the users' perspective, AutoCross is a black box that takes as input the training data and feature types (i.e., categorical, numerical, time series, etc.), and outputs a feature producer. The feature producer can fast perform crossing learned by AutoCross to transform the input data, which is then used by the learning algorithm

in model training, or the learned model in inference. It employs hashing trick [39] to improve the accelerate feature producing. Compared with deep-learning-based methods, the feature producer takes significantly less computation resources, and is hence especially suitable for real-time inference.

Inside the black box ('flow' part in Figure 2), the data will first be preprocessed, where hyper-parameters are determined, missing values filled and numerical features discretized. Afterwards, useful feature sets are iteratively constructed in a loop consisting of two steps: 1) *feature set generation*, where candidate feature sets with new cross features are generated; and 2) *feature set evaluation*, where candidate feature sets are evaluated and the best is selected as a new solution. This iterative procedure is terminated once some conditions are met.

From the implementation perspective ('infrastructures' part in Figure 2), the foundation of AutoCross is a distributed computing environment based on the well-known parameter server (PS) architecture [25]. Data is cached in memory by blocks, where each block contains a small subset of the training data. Workers visit the cached data blocks, generate corresponding features, and evaluate them. A feature manager takes control over the feature set generation and evaluation. A process manager controls the whole procedure of feature crossing, including hyper-parameter adaptation, data preprocessing, work flow control, and program termination.

The algorithms, that bridge the work flow and infrastructures, are the main focus of this paper ('algorithms' part of Figure 2). Each algorithm corresponds to a part in the work flow: we employ beam search for feature set generation to explore an extensive search space (Section 4.2), field-wise logistic regression and successive mini-batch gradient descent for feature set evaluation (Section 4.3), and multi-granularity discretization for data preprocessing (Section 4.4). These algorithms are chosen, designed, and optimized with the considerations of simplicity and costs of distributed computing, as will be detailed in the next section.

4 METHOD

In this section, we detail the algorithms used in AutoCross. We focus on the binary classification problem. It is not only the subject of most existing works [5, 7, 21, 26, 34], but also the most widely considered problem in real-world businesses [3, 4, 10, 23, 38, 41].

4.1 Problem Definition

For the ease of discussion, first we assume that all the original features are categorical. The data is represented in the *multi-field* categorical form [26, 37, 42], where each *field* is a binary vector generated from a categorical feature by encoding (one-hot encoding or hashing trick). Given training data $\mathcal{D}_{TR}$, we split it into a sub-training set $\mathcal{D}_{tr}$ and a validation set $\mathcal{D}_{vld}$. Then, we represent $\mathcal{D}_{tr}$ with a feature set $\mathcal{S}$, and with learning algorithm $\mathcal{L}$ learn a model $\mathcal{L}(\mathcal{D}_{tr}, \mathcal{S})$. To evaluate this model, we represent the validation set $\mathcal{D}_{vld}$ with the same feature set $\mathcal{S}$ and calculate a metric $\mathcal{E}\left(\mathcal{L}(\mathcal{D}_{tr}, \mathcal{S}), \mathcal{D}_{vld}, \mathcal{S}\right)$, which should be maximized.

Now, we formally define the *feature crossing problem* as:

$$\max_{\mathcal{S} \subseteq A(\mathcal{F})} \mathcal{E}\left(\mathcal{L}(\mathcal{D}_{tr}, \mathcal{S}), \mathcal{D}_{vld}, \mathcal{S}\right), \tag{4}$$

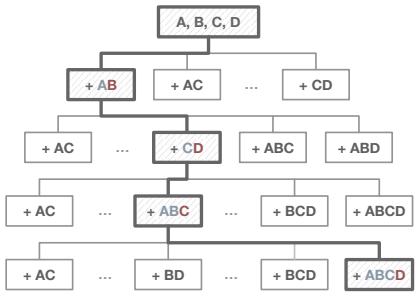

Figure 3: An illustration of the search space and beam search strategy employed in AutoCross. In beam search, only the best node (bold stroke) at each level is expanded. We use two colors to indicate the two features that are used to construct the new cross feature.

where $\mathcal{F}$ is the original feature set of $\mathcal{D}_{TR}$, and $A(\mathcal{F})$ is the set of all original features and possible cross features generated from $\mathcal{F}$.

4.2 Feature Set Generation

In this subsection, we introduce the feature set generation method in AutoCross, which also determines the main search strategy.

We consider the feature crossing problem (Problem (4)). Assume the size of the original feature set is d, which is also the highest order of cross features. The size of $A(\mathcal{F})$ is:

$$\text{card}\left(A(\mathcal{F})\right) = \sum_{k=1}^{d} C(d,k) = 2^d - 1, \tag{5}$$

and the number of all possible feature sets is $2^{(2^d-1)}$, a double exponential function of d. Obviously, it is impractical to search for an globally optimal feature set in such an extensive space. In order to find a moderate solution with limited time and computational resources, we employ a greedy approach to *iteratively construct a locally optimal feature set.*

In AutoCross, we consider a tree-structured space $\mathcal{T}$ depicted in Figure 3, where each node corresponds to a feature set and the root is the original feature set $\mathcal{F}$. [2] For simplicity, in this example, we denote the crossing of two features `A` and `B` as `AB`, and higher-order cross features in similar ways. For a node (a feature set), its each child is constructed by adding to itself one *pair-wise* crossing of its own elements. The pair-wise interactions between cross features (or a cross feature and an original feature) will lead to *high-order* feature crossing. The new space $\mathcal{T}$ considers all possible features in $A(\mathcal{F})$, but excludes part of its subsets. With $\mathcal{T}$, to search for a feature set is equivalent to identifying a *path* from the root of $\mathcal{T}$ to a specific node. This can be done by iteratively adding cross features into a maintained feature set. However, the size of $\mathcal{T}$ is $O\left((d^2/2)^k\right)$ where k is the maximum number of generated cross features. It grows exponentially with k. Hence, it will be extremely expensive to exhaustively visit all possible solutions, or in other

[2] In Figure 3 only one node at each level is expanded. This is because we use beam search strategy. It should be noted that the search space $\mathcal{T}$ is a fully expanded tree.

Algorithm 1 Feature Set Search Strategy in AutoCross.

Require: original feature set $\mathcal{F}$.
Ensure: solution $\mathcal{S}^*$.
1: initialize current node $\mathcal{S}^* \leftarrow \mathcal{F}$;
2: **while** procedure not terminated **do**
3: **Feature Set Generation**: expand $\mathcal{S}^*$, generate its children node set `children`($\mathcal{S}^*$) by adding to itself different pair-wise crossing of its elements;
4: **Feature Set Evaluation**: evaluate all candidate feature sets in `children`($\mathcal{S}^*$) and identify the best child $\mathcal{S}'$;
5: visit $\mathcal{S}'$: $\mathcal{S}^* \leftarrow \mathcal{S}'$
6: **end while**
7: **return** $\mathcal{S}^*$.

words, to traverse $\mathcal{T}$. To address this issue, we employ beam search to further improve the efficiency.

Beam search [29] is a greedy strategy to explore a graph with low computation and memory costs. The basic idea is to only expand the most promising nodes during search. First we generate all children nodes of the root, evaluate their corresponding feature sets and choose the best performing one to visit next. In the process that follows, we expand the current node and visit its most promising child. When the procedure is terminated, we end at a node that is considered as the solution. By this means, we only consider $O\left(kd^2\right)$ nodes in a search space with size $O\left((d^2/2)^k\right)$, and the cost grows linearly with k, the maximal number of cross features. It enables us to efficiently construct high-order cross features. This feature set generation method leads to the main feature set search strategy in AutoCross, as described in Algorithm 1. Figure 3 highlights a search path that begins from the original feature set `{A, B, C, D}` and ends at `{A, B, C, D, AB, CD, ABC, ABCD}`, the solution.

4.3 Feature Set Evaluation

A vital step in Algorithm 1 is to evaluate the performance of candidate feature sets (Step 4). Here, the performance of a candidate set $\mathcal{S}$ is expressed as $\mathcal{E}\left(\mathcal{L}(\mathcal{D}_{tr}, \mathcal{S}), \mathcal{D}_{vld}, \mathcal{S}\right)$ (see Problem (4)), denoted as $\mathcal{E}(\mathcal{S})$ for short. To directly estimate it, we need to learn a model with algorithm $\mathcal{L}$ on the training set represented by $\mathcal{S}$ and evaluate its performance on the validation set. Though highly accurate, direct evaluation for feature sets is often rather expensive. In real-world business scenarios, training a model to convergence may take great computational resources. Such direct evaluations are often too expensive to be invoked repetitively in the feature generation procedure. In order to improve the evaluation efficiency, we proposed field-wise logistic regression and successive mini-batch gradient descent in AutoCross.

4.3.1 Field-wise Logistic Regression. Our first effort to accelerate feature set evaluation is *field-wise logistic regression* (field-wise LR). Two approximations are made. First, we use logistic regression (LR) trained with mini-batch gradient descent to evaluate candidate feature sets, and use the corresponding performance to approximate the performance of the learning algorithm $\mathcal{L}$ that actually follows. We choose logistic regression since, as a generalized linear model, it is the most widely used model in large scale machine learning. It is simple, scalable, fast for inference, and makes interpretable predictions [5, 7, 34].

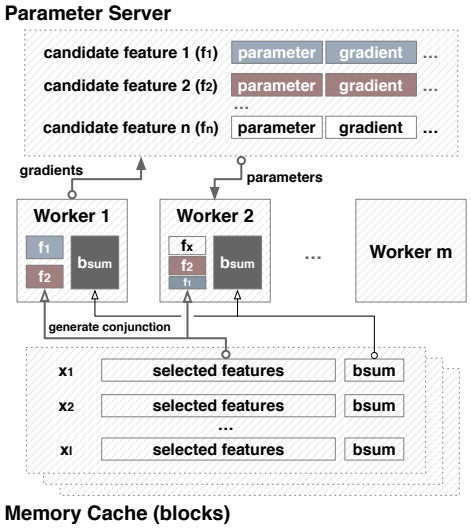

Figure 4: Illustration of field-wise logistic regression for feature evaluation based on a parameter server architecture.

The second approximation is that, during model training, we only learn the weights of the newly added cross feature, while other weights are fixed. Hence, the training is 'field-wise'. For example, assume the current solution feature set is $\mathcal{S}^*$ = `{A, B, C, D}`, and we want to evaluate a candidate set $\mathcal{S}$ = `{A, B, C, D, AB}`. Only the weights of `AB` is updated in training. Formally, denote an instance as $\mathbf{x} = [\mathbf{x}_s^{\mathrm{T}}, \mathbf{x}_c^{\mathrm{T}}]^{\mathrm{T}}$, where $\mathbf{x}_s$ corresponds to all features in the current solution and $\mathbf{x}_c$ the newly added cross feature. Their corresponding weights are $\mathbf{w} = [\mathbf{w}_s^{\mathrm{T}}, \mathbf{w}_c^{\mathrm{T}}]^{\mathrm{T}}$. An LR model makes prediction:

$$P(y = 1|\mathbf{x}) = s(\mathbf{w}^{\mathrm{T}}\mathbf{x}) = s(\mathbf{w}_s^{\mathrm{T}}\mathbf{x}_s + \mathbf{w}_c^{\mathrm{T}}\mathbf{x}_c) = s(\mathbf{w}_c^{\mathrm{T}}\mathbf{x}_c + b_{sum}), \quad (6)$$

where $s(\cdot)$ is the sigmoid function. In field-wise LR, we only update $\mathbf{w}_c$, and since we fix $\mathbf{w}_s$, b_{sum} is a constant scalar during training. We cache the values of b_{sum} in the memory so that they can be directly fetched by the workers.

Figure 4 shows how field-wise LR works on the parameter server architecture. Field-wise LR improves the efficiency of feature set evaluation from several aspects: 1) *Storage*: the workers store only $\mathbf{x}_c$ (in sparse format where only the hashed values are stored) and b_{sum}, rather than full representation of instances; there is a negligible overhead to store b_{sum} in memory cache; 2) *Transmitting*: the contents of transmission between the memory cache and workers are b_{sum} and the hashed values of the features that are used to construct $\mathbf{x}_c$. Transmission of full instance representation is therefore spared; 3) *Computation*: only $\mathbf{w}_c$ is updated, which reduces the computation burden of workers and parameter servers; all workers directly fetch the stored b_{sum}, so that the latter need not to be repetitively calculated for every mini-batch at each worker.

Once the field-wise LR finishes, we estimate the performance of the resulting model on the validation set $\mathcal{D}_{vld}$. We use the resulting metrics $\mathcal{E}'(\mathcal{S})$, such as Area-Under-Curve (AUC), accuracy, or negative log-loss, to evaluate the quality of $\mathcal{S}$. Obviously, $\mathcal{E}'(\mathcal{S})$ is an

approximation of $\mathcal{E}(\mathcal{S})$, with accuracy traded for higher efficiency. However, since the purpose of feature set evaluation is to *identify the most promising candidate*, rather than *to accurately estimate the performance of candidates*, a degraded accuracy is acceptable if only it can recognize the *best* candidate with high probability. Experimental results reported in Section 5 demonstrate the effectiveness of field-wise LR.

After a candidate is selected to replace the current solution $\mathcal{S}^*$ (Step 6, Algorithm 1), we train an LR model with the new $\mathcal{S}^*$, evaluate its performance, and update b_{sum} for data blocks that will be used in the next iteration. Details will be discussed immediately.

4.3.2 Successive Mini-batch Gradient Descent. In AutoCross, we use a successive mini-batch gradient descent method to further accelerate field-wise LR training. It is motivated by the successive halving algorithm [18] which was originally proposed for multi-arm bandit problems. Successive halving features an efficient allocation of computing resources and high simplicity. In our case, we consider each candidate feature set as an arm, and a pull of the arm is to train the corresponding field-wise LR model with a data block. The instant reward of pulling an arm is the resulting validation AUC of the partially trained model. The training data is equally split into $N \geq \sum_{k=0}^{\lceil \log_2 n \rceil - 1} 2^k$ data blocks, where n is the number of candidates. Then we invoke Algorithm 2 to identify the best candidate feature set. Successive mini-batch gradient descent allocates more resources to more promising candidates. The only hyper-parameter N, namely the number of data blocks, is adaptively chosen according to the size of data set and the working environment. Users do not need to tune the mini-batch size and sample ratios that are critical for vanilla subsampling.

Algorithm 2 Successive Mini-batch Gradient Descent (SMBGD).

Require: set of candidate feature sets $\mathbb{S} = \{\mathcal{S}_i\}_{i=1}^{n}$, training data equally divided into $N \geq \sum_{k=0}^{\lceil \log_2 n \rceil - 1} 2^k$ data blocks.
Ensure: best candidate $\mathcal{S}'$.

1: **for** $k = 0, 1, \cdots, \lceil \log_2 n \rceil - 1$ **do**
2: use additional 2^k data blocks to update the field-wise LR models of all $\mathcal{S} \in \mathbb{S}$, with warm-starting;
3: evaluate the models of all $\mathcal{S}$'s with validation AUC;
4: keep the top half of candidates in $\mathbb{S}$: $\mathbb{S} \leftarrow$ `top_half`($\mathbb{S}$) (rounding down);
5: break if $\mathbb{S}$ contains only one element;
6: **end for**
7: **return** $\mathcal{S}'$ (the singleton element of $\mathbb{S}$).

4.4 Preprocessing

In AutoCross, we use *discretization* in the data preprocessing step to enable feature crossing between numerical and categorical features. Discretization has been proven useful to improve predicting capability of numerical features [5, 24, 27]. The most simple and widely-used discretization method is equal-width discretization, i.e., to split the value range of a feature into several equal-width intervals. However, in traditional machine learning applications, the number of intervals, named as *granularity* in our work, has a great impact on the learning performance and should be carefully determined by human experts.

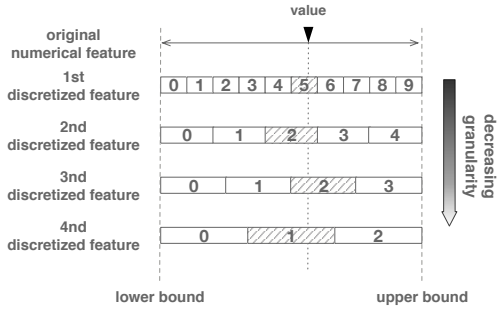

Figure 5: An illustration of multi-granularity discretization. Shade indicates the value taken by each discretized feature.

In order to automate discretization and spare its dependence on human experts, we propose a *multi-granularity discretization* method. The basic idea is simple: instead of using a fine-tuned granularity, we discretize each numerical feature into several, rather than only one, categorical features, each with a different granularity. Figure 5 gives an illustration of discretizing a numerical feature with four levels of granularity. Since more levels of granularity are considered, it is more likely to get a promising result.

In order to avoid the dramatic increase in feature number caused by discretization, once these features are generated, we use field-wise LR (without considering b_{sum}) to evaluate them and keep only the best half. A remaining problem is how to determine the levels of granularity. For an experienced user, she can set a group of potentially good values. If no values are specified, AutoCross will use $\{10^p\}_{p=1}^{P}$ as default values, where P is an integer determined by a rule-based mechanism that considers the available memory, data size and feature numbers.

In addition, AutoCross will invoke a tuning procedure in the preprocessing step to find optimal hyper-parameters for LR models. They will be used in all LR models involved subsequently.

4.5 Termination

Three kinds of termination conditions are used in AutoCross: 1) *runtime condition*: the user can set a maximal runtime of AutoCross. When the time elapses, AutoCross terminates outputs the current solution $\mathcal{S}^*$. Additionally, the user can always interrupt the procedure and get the result of the time; 2) *performance condition*: after a new feature set is generated (Step 6, Algorithm 1), an LR model is trained with all its features. If, compared with the former set, the validation performance degrades, the procedure is terminated; 3) *maximal feature number*: the user can give a maximal cross feature number so that AutoCross stops when the number is reached.

5 EXPERIMENTS

In this section, we demonstrate the effectiveness and efficiency of AutoCross. First, by comparing AutoCross with several reference methods on both benchmark and real-world business datasets, we show that with feature crossing it can significantly improve the performance of both linear and deep models, and that high-order cross features are useful. Then we report the time costs of feature crossing with AutoCross. Finally, we show the advantage of AutoCross in real-time inference.

5.1 Setup

Datasets: we test AutoCross with both benchmark and real-world business datasets, gathered from different applications. Table 2 summarizes these datasets[3]. All the datasets are for binary classification. The real-world business datasets are provided by the customers of 4Paradigm with sanitization.

Table 2: Characteristics of datasets used in the experiments. 'Num.' and 'Cate.' indicate numerical and categorical features respectively. '# Val.' indicates the number of different values taken by the categorical features. 'H.R.' is short for 'human resource' and 'Adv.' for 'advertising'.

Benchmark Datasets						
Name	# Samples		# Features			Domain
	Training	Testing	# Num.	# Cate.	# Val.	
Bank	27,459	13,729	10	10	63	Banking
Adult	32,561	16,281	6	8	42	Social
Credit	100,000	50,000	10	0	0	Banking
Employee	29,493	3,278	0	9	7,518	H. R.
Criteo	41,256 K	4,584 K	13	26	33,762 K	Adv.
Real-World Business Datasets						
Name	# Samples		# Features			Domain
	Training	Testing	# Num.	# Cate.	# Val.	
Data1	2,641,185	719,998	34	28	4,181,854	Sports
Data2	1,888,366	1,119,778	8	19	109,180	Talkshow
Data3	2,340,209	1,059,016	55	21	3,174,081	Social
Data4	2,848,746	688,481	7	19	455,778	Video
Data5	11,802,126	2,058,424	8	18	436,361	News

Methods: in order to demonstrate the effectiveness of AutoCross, we compare the following methods;

- **AC+LR**: logistic regression with cross features generated by AutoCross;
- **AC+W&D**: Wide & Deep method [7] whose wide part uses cross features generated by AutoCross;
- **LR (base)**: our self-developed logistic regression with only the original features. It is used as the baseline;
- **CMI+LR**: logistic regression with cross features generated by the method proposed in [5], where conditional mutual information (CMI) is used as the metric to evaluate features. This method only considers second-order feature crossing;
- **Deep**: a deep model with embedding layers to deal with categorical features. It implicitly generate feature interactions;
- **xDeepFM**: the method proposed in [26], which explicitly generates features with a compressed interaction network. It is the state-of-the-art of deep-learning-based method.

In these methods, **AC+LR** and **AC+W&D** use the cross features generated by AutoCross, and demonstrate its effectiveness to enhance linear and deep models. **CMI+LR** uses a representative search-based feature crossing method. **xDeepFM** is the state-of-the-art method following the Wide & Deep framework. We choose it as a reference method since it outperforms other existing deep-learning-based methods, as reported in [26]. We also consider **Deep** to test how a bare-bone deep model performs. All these methods are designed to handle tabular data with categorical features.

[3]Availability of data sets are in Appendix B.3.

Reproducibility: the features and models are learned with training and validation data (20% of the training data, if needed), and the resulting AUCs on the testing data indicate the performance of different methods. *More information about the settings of methods under test can be found in Appendix B.*

5.2 Results

5.2.1 Effectiveness. Table 3 reports the resulting test AUCs on the benchmark and real-world business datasets. We did not run **CMI+LR** on real-world business datasets because there are multi-value categorical features that cannot be handled by CMI. In the table, we highlighted the top-two methods for each dataset. As can be easily observed, **AC+LR** ranks top-two in most cases, and often outperforms deep-learning-based methods (**Deep** and **xDeepFM**). **AC+W&D** also shows competitive performance, demonstrating the capability of AutoCross to enhance deep models. In most cases, **AC+LR** and **AC+W&D** show better results than **xDeepFM**. According to Proposition 1, xDeepFM only generates a special case of embedded cross feature. This results show the effectiveness to directly and explicitly generate high-order cross features.

Table 3: Experimental results (test AUC) on benchmark and real-world business datasets.

Benchmark Datasets					
Method	Bank	Adult	Credit	Employee	Criteo
LR (base)	0.9400	0.9169	0.8292	0.8655	0.7855
AC+LR	**0.9455**	**0.9280**	**0.8567**	**0.8942**	0.8034
AC+W&D	0.9420	**0.9260**	**0.8623**	**0.9033**	**0.8068**
CMI+LR	**0.9431**	0.9153	0.8336	0.8901	0.7844
Deep	0.9418	0.9130	0.8369	0.8745	0.7985
xDeepFM	0.9419	0.9131	0.8358	0.8746	**0.8059**
Real-World Business Datasets					
Method	Data1	Data2	Data3	Data4	Data5
LR (base)	0.8368	0.8356	0.6960	0.6117	0.5992
AC+LR	**0.8545**	**0.8536**	**0.7065**	**0.6276**	0.6393
AC+W&D	**0.8531**	**0.8552**	**0.7026**	**0.6260**	**0.6547**
Deep	0.8479	0.8463	0.6936	0.6207	0.6509
xDeepFM	0.8504	0.8515	0.6936	0.6241	**0.6514**

As has been discussed in the papers of Wide & Deep [7] and DeepFM [14], in online recommendation scenarios, small improvement (0.275% in [7] and 0.868% in [14], compared with LR) in off-line AUC evaluation can lead to a significant increase in online CTR and hence great commercial benefits. Table 4 shows the test AUC improvement brought by AutoCross. It can be observed that both **AC+LR** and **AC+W&D** achieve significant improvement over **LR (base)**, and **AC+W&D** also considerably improve the performance of deep model. These results demonstrate that by generating cross features, AutoCross can make the data more informative and discriminative, and improve the learning performance. The promising results achieved by AutoCross also demonstrate the capability of field-wise LR to identify useful cross features.

5.2.2 The effect of high-order features. With the above reported results, we have demonstrated the effect of AutoCross. Figure 6 shows the number of second/high-order cross features generated for each dataset, where the latter take a considerable proportion. Besides, in Table 5, we compare the performance improvements brought

Table 4: Test AUC improvement v.s. LR (base) and Deep.

AC+LR v.s. **LR (base)**					
Bank	Adult	Credit	Employee	Criteo	**Average**
0.585%	1.211%	3.316%	3.316%	2.279%	2.141%
Data1	Data2	Data3	Data4	Data5	**Average**
2.115%	2.154%	1.509%	2.599%	6.692%	3.014%
AC+W&D v.s. **LR (base)**					
Bank	Adult	Credit	Employee	Criteo	**Average**
0.213%	0.992%	3.992%	4.367%	2.712%	2.455%
Data1	Data2	Data3	Data4	Data5	**Average**
1.948%	2.346%	0.948%	2.338%	9.546%	3.368%
AC+W&D v.s. **Deep**					
Bank	Adult	Credit	Employee	Criteo	**Average**
0.021%	1.424%	3.035%	3.293%	1.039%	1.763%
Data1	Data2	Data3	Data4	Data5	**Average**
0.6133%	1.0516%	1.2976%	0.8539%	0.5361%	0.880%

by **CMI+LR**, that only generates second-order cross features, and **AC+LR** that considers high-order feature crossing. We can see that **AC+LR** stably and constantly outperforms **CMI+LR**. This result demonstrates the usefulness of high-order cross features.

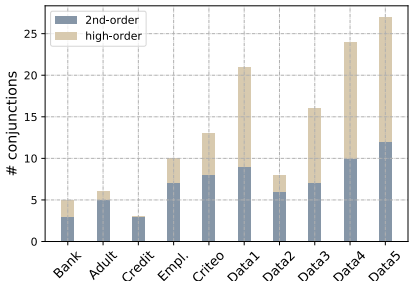

Figure 6: The number of second/high-order cross features generated for each dataset.

Table 5: Test AUC improvement: second v.s. high order features on benchmark datasets.

v.s. **LR(base)**	Bank	Adult	Credit	Employee	Criteo	**Average**
CMI+LR	0.330%	-0.175%	0.531%	2.842%	-0.140%	0.678%
AC+LR	0.585%	1.211%	3.316%	3.316%	2.279%	2.141%

5.2.3 Time costs of feature crossing. Table 6 reports the feature crossing time of AutoCross on each dataset. Figure 7 shows the validation AUC (**AC+LR**) versus runtime on real-world business datasets. Such curves are visible to the user and she can terminate AutoCross at any time to get the current result. It is notable that due to the high simplicity of AutoCross, no hyper-parameter needs to be fine-tuned, and the user does not need to spend any extra time to get it work. In contrast, if deep-learning-based methods are used, plenty of time will be spent on the network architecture design and hyper-parameter tuning.

5.2.4 Inference Latency. In many real-world businesses, the application scenario of a feature generation tool comprises three stages: 1) off-line feature generation; 2) off-line/online model training; 3) online inference. In this scenario, the off-line generation stage is invoked the least frequently, for instance, features can be generated

Table 6: Cross feature generation time (unit: hour).

Benchmark Datasets				
Bank	Adult	Credit	Employee	Criteo
0.0267	0.0357	0.3144	0.0507	3.0817
Real-World Business Datasets				
Data1	Data2	Data3	Data4	Data5
0.9327	0.7973	1.5206	2.7572	5.1861

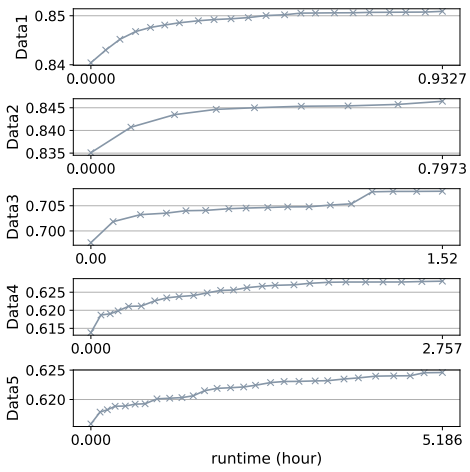

Figure 7: Validation AUC curves in real-business datasets.

weekly or even monthly. In contrast, within every millisecond, hundreds or thousands of inferences may sequentially take place, which makes high efficiency a must. Online inference consists of two major steps: 1) feature producing to transform the input data, and 2) inference to make prediction. Deep-learning method combines these steps. In Table 7, we report the inference time of **AC+LR**, **AC+W&D**, **Deep** and **xDeepFM**.

Table 7: Inference latency comparison (unit: millisecond).

Benchmark Datasets					
Method	Bank	Adult	Credit	Employee	Criteo
AC+LR	0.00048	0.00048	0.00062	0.00073	0.00156
AC+W&D	0.01697	0.01493	0.00974	0.02807	0.02698
Deep	0.01413	0.01142	0.00726	0.02166	0.01941
xDeepFM	0.08828	0.05522	0.04466	0.06467	0.18985
Real-World Business Datasets					
Method	Data1	Data2	Data3	Data4	Data5
AC+LR	0.00367	0.00111	0.00185	0.00393	0.00279
AC+W&D	0.03537	0.01706	0.04042	0.02434	0.02582
Deep	0.02616	0.01348	0.03150	0.01414	0.01406
xDeepFM	0.32435	0.11415	0.40746	0.12467	0.13235

It can be easily observed that **AC+LR** is orders of magnitude faster than other methods in inference. This demonstrates that, AutoCross can not only improve the model performance, but also ensure fast inference with its feature producer.

6 RELATED WORKS

In this section, we briefly review works that are loosely related to AutoCross and demonstrate why they do not suit our purpose.

Factorization machines seek low-dimensional embeddings of original features, and capture their interactions [2, 6, 19]. Such interactions, however, are not explicitly constructed. Furthermore, they may over-generalize [7] and/or introduce noise since they enumerate all possible interactions regardless of their usefulness [26].

There are also some embedded feature selection/generation methods, such as group lasso [30] and gradient boost machine [12], that intrinsically identify or implicitly construct useful features along model training. However, these methods often struggle to deal with large scale problems with high-dimensional sparse data generated from categorical features, and/or have computational issues when the number of features is large, which happens when high-order feature crossing is considered.

Finally, itemsets [1] have been well studied in data mining communities. Like cross features, they also represent the co-occurrence of attributes. However, the difference is that the elements in an itemset are often of a same kind, e.g., all being commodities. Also, itemsets are mostly used in rule-based machine learning techniques such as frequent patterns [16] and association rules [32]. These techniques may have trouble to generalize, and are slow in inference when the number of rules is large, due to great retrieving costs [15].

7 CONCLUSION

In this paper, we present AutoCross, an automatic feature crossing method for tabular data in real-world applications. It captures useful interactions among categorical features and increases the predictive power of learning algorithms. It employs beam search to efficiently construct cross features, which enables the consideration of high-order feature crossing, which is not yet visited by existing works. Successive mini-batch gradient descent and multi-granularity discretization are proposed to further improve the efficiency and effectiveness while keeping high simplicity. All the algorithms are designed for distributed computing to deal with big data in real-world businesses. Experimental results show that AutoCross can significantly enhance learning from tabular data, outperforming other search-based and deep-learning-based feature generation methods dedicated to the same topic.

REFERENCES

[1] R. Agrawal, T. Imieliński, and A. Swami. 1993. Mining association rules between sets of items in large databases. In *ACM Sigmod Record*, Vol. 22. ACM, 207–216.

[2] M. Blondel, A. Fujino, N. Ueda, and M. Ishihata. 2016. Higher-order factorization machines. In *Advances in Neural Information Processing Systems*. 3351–3359.

[3] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. 2013. Recommender systems survey. *Knowledge-Based Systems* 46 (2013), 109–132.

[4] R. Bolton and D. Hand. 2002. Statistical fraud detection: A review. *Statistical science* (2002), 235–249.

[5] O. Chapelle, E. Manavoglu, and R. Rosales. 2015. Simple and scalable response prediction for display advertising. *ACM Transactions on Intelligent Systems and Technology (TIST)* 5, 4 (2015), 61.

[6] C. Cheng, F. Xia, T. Zhang, I. King, and M. Lyu. 2014. Gradient boosting factorization machines. In *ACM Conference on Recommender systems*. 265–272.

[7] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, and M. Ispir. 2016. Wide & deep learning for recommender systems. In *Workshop on Deep Learning for Recommender Systems*. 7–10.

[8] D. Crankshaw, X. Wang, G. Zhou, M. Franklin, J. Gonzalez, and I. Stoica. 2017. Clipper: A low-latency online prediction serving system.. In *USENIX Symposium on Networked Systems Design and Implementation*. 613–627.

[9] P. Domingos. 2012. A few useful things to know about machine learning. *Commun. ACM* 55, 10 (2012), 78–87.

[10] D. Evans. 2009. The online advertising industry: Economics, evolution, and privacy. *Journal of Economic Perspectives* 23, 3 (2009), 37–60.

[11] W. Fan, E. Zhong, J. Peng, O. Verscheure, K. Zhang, J. Ren, R. Yan, and Q. Yang. 2010. Generalized and heuristic-free feature construction for improved accuracy. In *SIAM International Conference on Data Mining*. 629–640.

[12] J. Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[13] H. Guo and W. Hsu. 2002. A survey of algorithms for real-time Bayesian network inference. In *Join Workshop on Real Time Decision Support and Diagnosis Systems*.

[14] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. In *International Joint Conference on Artificial Intelligence*. 1725–1731.

[15] J. Han, J. Pei, and M. Kamber. 2011. *Data mining: concepts and techniques*. Elsevier.

[16] J. Han, J. Pei, and Y. Yin. 2000. Mining frequent patterns without candidate generation. In *ACM Sigmod Record*, Vol. 29. 1–12.

[17] S. Han, H. Mao, and W. Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*.

[18] K. Jamieson and A. Talwalkar. 2016. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*. 240–248.

[19] Y. Juan, Y. Zhuang, W.-S. Chin, and C.-J. Lin. 2016. Field-aware factorization machines for CTR prediction. In *ACM Conference on Recommender Systems*. 43–50.

[20] J. Kanter and K. Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *IEEE International Conference on Data Science and Advanced Analytics*. 1–10.

[21] G. Katz, E. Shin, and D. Song. 2016. Explorekit: Automatic feature generation and selection. In *International Conference on Data Mining*. 979–984.

[22] D. Kingma and J. Ba. 2014. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

[23] I. Kononenko. 2001. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in medicine* 23, 1 (2001), 89–109.

[24] S. Kotsiantis and D. Kanellopoulos. 2006. Discretization techniques: A recent survey. *GESTS International Transactions on Computer Science and Engineering* 32, 1 (2006), 47–58.

[25] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. Andersen, and A. Smola. 2013. Parameter server for distributed machine learning. In *Big Learning NIPS Workshop*, Vol. 6. 2.

[26] J. Lian, X. Zhou, F. Zhang, Z. Chen, X. Xie, and G. Sun. 2018. xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems. In *International Conference on Knowledge Discovery & Data Mining*.

[27] H. Liu, F. Hussain, C. Tan, and M. Dash. 2002. Discretization: An enabling technique. *Data mining and knowledge discovery* 6, 4 (2002), 393–423.

[28] H. Liu, H. sand Motoda. 1998. *Feature extraction, construction and selection: A data mining perspective*. Vol. 453. Springer Science & Business Media.

[29] M. Medress, F. Cooper, J. Forgie, C. Green, D. Klatt, M. O'Malley, E. Neuburg, A. Newell, and B. Reddy, D Ritea. 1977. Speech understanding systems: Report of a steering committee. *Artificial Intelligence* 9, 3 (1977), 307–316.

[30] L. Meier, S. Van De Geer, and P. Bühlmann. 2008. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 70, 1 (2008), 53–71.

[31] T. Mitchell. 1997. *Machine learning*. Springer Science & Business Media.

[32] R. Ng, L. Lakshmanan, J. Han, and A. Pang. 1998. Exploratory mining and pruning optimizations of constrained associations rules. In *ACM Sigmod Record*, Vol. 27. ACM, 13–24.

[33] Y. Qu, H. Cai, K. Ren, W. Zhang, Y. Yu, Y. Wen, and J. Wang. 2016. Product-based neural networks for user response prediction. In *IEEE International Conference on Data Mining*. IEEE, 1149–1154.

[34] R. Rosales, H. Cheng, and E. Manavoglu. 2012. Post-click conversion modeling and analysis for non-guaranteed delivery display advertising. In *ACM International Conference on Web Search and Data Mining*. 293–302.

[35] M. Smith and L. Bull. 2005. Genetic programming with a genetic algorithm for feature construction and selection. *Genetic Programming and Evolvable Machines* 6, 3 (2005), 265–281.

[36] B. Tran, B. Xue, and M. Zhang. 2016. Genetic programming for feature construction and selection in classification on high-dimensional data. *Memetic Computing* 8, 1 (2016), 3–15.

[37] R. Wang, B. Fu, G. Fu, and M. Wang. 2017. Deep & cross network for ad click predictions. In *KDD Workshop*. ACM, 12.

[38] S. Wang. 2010. A comprehensive survey of data mining-based accounting-fraud detection research. In *Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on*, Vol. 1. IEEE, 50–53.

[39] K. Weinberger, A. Dasgupta, J. Attenberg, J. Langford, and A. Smola. 2009. Feature hashing for large scale multitask learning. In *International Conference on Machine Learning*.

[40] Q. Yao, M. Wang, Y. Chen, W. Dai, Y. Hu, Y. Li, W.-W. Tu, Q. Yang, and Y. Yu. 2018. *Taking Human out of Learning Applications: A Survey on Automated Machine Learning*. Technical Report. arXiv preprint.

[41] R. Zeff and B. Aronson. 1999. *Advertising on the Internet*. John Wiley & Sons, Inc.

[42] W. Zhang, T. Du, and J. Wang. 2016. Deep learning over multi-field categorical data. In *European conference on information retrieval*. Springer, 45–57.

[43] Y. Zhang, Q. Yao, W. Dai, and L. Chen. 2019. *AutoKGE: Searching Scoring Functions for Knowledge Graph Embedding*. Technical Report. arXiv preprint arXiv:1904.11682.

A PROOF OF PROPOSITION 1

Since high-order interactions can be represented by pair-wise interaction of lower-order ones, without loss of generality, in Proposition 1 we only consider second-order feature crossing $\mathbf{c}$ (Equation 1) and second-order entry-wise product of embedded vectors $\mathbf{e}$ (Equation 2).

Proof of Proposition 1. First, we consider a weak version of Proposition 1:

Proposition 2. *There exist at least one embedding matrix* $\mathbf{C}$ *with* D *rows so that: there do not exist any embedding matrices* $\mathbf{A}$ *and* $\mathbf{B}$ *that satisfy the following equation:*

$$\mathbf{Ax} \circ \mathbf{By} = \mathbf{Cz}, \tag{7}$$

for all binary vectors $\mathbf{x}$, $\mathbf{y}$ *and their crossing* $\mathbf{z}$.

We proof Proposition 2 by contradiction. Consider its opposite proposition:

Proposition 3 (Opposite of Proposition 2). *For all embedding matrices* $\mathbf{C}$ *with* D *rows, there exist embedding matrices* $\mathbf{A}$ *and* $\mathbf{B}$ *that satisfy the following equation:*

$$\mathbf{Ax} \circ \mathbf{By} = \mathbf{Cz}, \tag{8}$$

for all binary vectors $\mathbf{x}$, $\mathbf{y}$ *and their crossing* $\mathbf{z}$.

For simplicity, here we consider the cases where both $\mathbf{x}$ and $\mathbf{y}$ only have one hot bit, i.e., only one entry is '1' and others are '0'. Proposition 3 is correct in other cases unless it is true in this case.

Let the i-th bit of $\mathbf{x}$ and k-th bit of $\mathbf{y}$ are '1'. To ease discussion, we denote the hot bit of the resulting crossing $\mathbf{c}$ as its ik-th bit. Further, denote $\mathbf{a}_i$ as the i-th column of $\mathbf{A}$, $\mathbf{b}_k$ the k-th column of $\mathbf{B}$, and $\mathbf{c}_{ik}$ the ik-th column of $\mathbf{C}$. Proposition 3 necessarily leads to that, for any $\mathbf{C}$, we can find $\mathbf{A}$ and $\mathbf{B}$ to satisfy:

$$\mathbf{a}_i \circ \mathbf{b}_k = \mathbf{c}_{ik}, \tag{9}$$

for all i and k. Now, consider two instances of $\mathbf{x}$, with the i- and j-th bits set as '1', respectively, and similarly two instances of $\mathbf{y}$, with the k- and l-th bits set as '1', respectively. We have four resulting equations:

$$\mathbf{a}_i \circ \mathbf{b}_k = \mathbf{c}_{ik}, \quad \mathbf{a}_i \circ \mathbf{b}_l = \mathbf{c}_{il},$$
$$\mathbf{a}_j \circ \mathbf{b}_k = \mathbf{c}_{jk}, \quad \mathbf{a}_j \circ \mathbf{b}_l = \mathbf{c}_{jl}.$$

Since $\mathbf{C}$ is an arbitrary embedding matrix, we can easily make it satisfy:

$$\mathbf{c}_{ik}/\mathbf{c}_{jk} \neq \mathbf{c}_{il}/\mathbf{c}_{jl}, \tag{10}$$

where / denotes the entry-wise division of vectors. Equation (10) leads to:

$$\mathbf{c}_{ik}/\mathbf{c}_{jk} = (\mathbf{a}_i \circ \mathbf{b}_k)/(\mathbf{a}_j \circ \mathbf{b}_k) = \mathbf{a}_i/\mathbf{a}_j$$
$$\neq \quad \mathbf{c}_{il}/\mathbf{c}_{jl} = (\mathbf{a}_i \circ \mathbf{b}_l)/(\mathbf{a}_j \circ \mathbf{b}_l) = \mathbf{a}_i/\mathbf{a}_j.$$

It leads to $\mathbf{a}_i/\mathbf{a}_j \neq \mathbf{a}_i/\mathbf{a}_j$ which apparently does not hold. Hence, Proposition 3 is false, which proofs that its opposite, namely Proposition 2, is true.

Furthermore, we can construct infinitely many $\mathbf{C}$'s to satisfy Equation 10. Every such $\mathbf{C}$ falsifies Proposition 3, and hence makes Proposition 2 true. This verifies Proposition 1. □

B DETAILS OF EXPERIMENTAL SETUP

B.1 Experimental Environment

All experiments are carried out on a workstation with Intel(R) Xeon(R) CPU (E5-2630 v4 @ 2.20GHz, 24 cores), 256G memory and 8T hard disk.

B.2 Setup

AutoCross Setup. The hyper-parameters of AutoCross are the number of data blocks N in successive mini-batch gradient descent, the levels of granularity for multi-granularity discretization, and the termination condition. The first two are determined by a rule-based mechanism. As a result, N are set to $2\sum_{k=0}^{\lceil \log_2 n \rceil - 1} 2^k$ for relatively small datasets, namely `Bank`, `Adult`, `Credit` and `Employee`. This indicates that at most 50% of the training data will be used in successive mini-batch gradient descent. For other datasets, $N = 5\sum_{k=0}^{\lceil \log_2 n \rceil - 1} 2^k$, which corresponds to a maximum of 20% sample ratio. With respect to the levels of granularity, AutoCross uses $\{10^i\}_{i=1}^{3}$ for all datasets. As for the termination condition, we only invoke the performance condition, i.e., AutoCross terminates only if newly added cross feature leads to a performance degradation.

Logistic Regression Setup: logistic regression model is used in **LR (base)**, **AC+LR**, and **CMI+LR** methods. The feature set evaluation of AutoCross also uses LR models. We use our self-developed LR method in the experiments. There are only three hyper-parameters: learning rate $\alpha \in [0.005, 1]$, L1 penalty $\lambda_1 \in [1e-4, 10]$, and L2 penalty $\lambda_2 \in [1e-4, 10]$. In our experiment, as well as the real-world application of AutoCross, we invoke a hyper-parameter tuning procedure before feature generation. Log-grid search is used to find the optimal hyper-parameters. They are used in **LR (base)**, **AC+LR**, and **CMI+LR** methods, as well as the LRs in AutoCross.

CMI+LR Setup: we only test **CMI+LR** on benchmark datasets since CMI cannot handle multi-value categorical features. For its feature generation method [5], we use the multi-granularity discretization to convert numerical features. In order to ensure the accuracy of feature evaluation, we use all training data to estimate CMI. An exception is the `Criteo` dataset, for which we set the subsample ratio to 10%. We set the maximal cross feature number to 15.

Deep Model Setup: we use the open-source implementation of xDeepFM (*https://github.com/Leavingseason/xDeepFM*) in **xDeepFM** method, and use the deep component in **AC+W&D** and **Deep** methods. Hyper-parameters are set as the xDeepFM paper [26] suggested. To be more specific, we use 0.001 as the learning rate, Adam [22] with mini-batch size 4096 as the optimization method, 0.0001 as the L2 regularization penalty, 400 layers for deep component, 200/100 layers for compressed interaction network for `Criteo`/other datasets, and 10 as the dimension of field embedding. Since validation data is not need in **xDeepFM** and **Deep**, we do not split the training set and use all of it in model training.

B.3 Data Sets Availability

- adult:
  https://archive.ics.uci.edu/ml/datasets/adult
- Bank:

https://www.kaggle.com/brijbhushannanda1979/bank-data
- Credit:
  https://www.kaggle.com/c/GiveMeSomeCredit/data
- Employee:
  https://www.kaggle.com/c/amazon-employee-access-challenge/data
- Criteo:
  https://www.kaggle.com/c/criteo-display-ad-challenge/data

Due to secrecy agreement, real-world business datasets are not public available.