




DAFEE: A Scalable Distributed Automatic Feature Engineering Algorithm for Relational Datasets

Wenqian Zhao¹ , Xiangxiang Li¹ , Guoping Rong² , Mufeng Lin¹ ,
Chen Lin¹ , and Yifan Yang¹  

¹ Transwarp Technology (Shanghai) Co., Ltd., Shanghai, China

{wenqian.zhao,xiangxiang.li,mufeng.lin,chen.lin,yifan.yang}@transwarp.io

² The Joint Laboratory of Nanjing University and Transwarp on Data Technology,
Nanjing University, Nanjing, China
ronggp@nju.edu.cn

Abstract. Automatic feature engineering aims to construct informative features automatically and reduce manual labor for machine learning applications. The majority of existing approaches are designed to handle tasks with only one data source, which are less applicable to real scenarios. In this paper, we present a distributed automatic feature engineering algorithm, DAFEE, to generate features among multiple large-scale relational datasets. Starting from the target table, the algorithm uses a Breadth-First-Search type algorithm to find its related tables and constructs advanced high-order features that are remarkably effective in practical applications. Moreover, DAFEE implements a feature selection method to reduce the computational cost and improve predictive performance. Furthermore, it is highly optimized to process a massive volume of data. Experimental results demonstrate that it can significantly improve the predictive performance by 7% compared to SOTA algorithms.

Keywords: AutoML · Automatic feature engineering · Relational dataset · Big data · Feature selection · Machine learning

1 Introduction

Nowadays, as most businesses in the world embrace the opportunity of data science, people are dramatically impressed by the magic of data mining. An experienced data scientist is capable of generating and transforming useful features based on his or her high degrees of skill. However, this procedure, which is called *feature engineering*, is highly manual and typically requires domain knowledge. In addition, it is usually unrepeatable and non-scalable and occupies the majority of time in a machine learning task.

W. Zhao and X. Li—These authors contributed equally to this work.

As a result, automatically extracting informative features from data has become the forefront of both academia and industry. Although priory knowledge is essential, there are still some regular routines of generating informative features from the original attributes. However, since the actual structured data is typically residing in the relational database management systems (RDMS), being represented as a set of tables with relational links, the combinations of original features' mathematical transformations among relational tables may also be important, which makes the number of potential features grows exponentially.

Recently, researchers try to automatically extract the information among the relational datasets in two ways. One way is to use deep neural networks to implicitly complete the feature generation work, such as R2N [18]. However, since the created features are uninterpretable, this approach is hardly applicable in scenarios requiring strict justifications, such as fraud detection in banks. The other is to generate a tree representation for the connected datasets and then constructs features automatically when searching through the tree. Deep Feature Synthesis (DFS) [12] uses the Depth-First-Search algorithm. After finding out the farthest table from the target table, DFS sequentially applies different kinds of mathematical functions and generates features according to the relationship along the search path. To prevent generating too many features, DFS introduces a hyper-parameter called max-depth to control the farthest table it can reach. However, some information may be lost since not all tables are used to generate features. Furthermore, as DFS only supports limited mathematical functions, it can not extract complex features in practice.

Based on these observations, we propose the Distributed Automatic Feature Engineering (DAFEE) algorithm to improve the performance of automatic feature engineering. DAFEE uses a Breadth-First-Search type algorithm to search through the datasets. Each time the features are joined back to the target table, feature selection strategies would be applied to remove useless features. Additionally, DAFEE generates high-order features by applying sophisticated transformations upon multiple keys, which are both informative and interpretable. The major contributions of this paper are summarized as follows:

- DAFEE improves the DFS algorithm and can produce more useful features by increasing interaction among entities efficiently.
- DAFEE generates advanced features to improve model performance in ways that SOTA algorithms cannot, such as generating features based on the interest values, by connecting multiple keys simultaneously and so on.
- DAFEE is implemented with the Spark [31] distributed framework and utilizes its advantage to be better applied to large datasets.
- DAFEE reduces the complexity of feature explosion and improves the robustness by applying pruning and feature selection strategy to trim trivial features before the feature expansion process.

The remaining part of this paper is organized as follows: Sect. 2 reviews some related work; Sect. 3 introduces our research motivation, DAFEE algorithm and its implementation in detail; Sects. 4 and 5 present the datasets and experiment results; Sect. 6 summarizes the work.

2 Related Work

Most automatic feature engineering works contain both automatic feature generation and feature selection. Some of them explicitly expand the dataset with all transformed features and apply a feature selection afterward. ExploreKit [13] is such a framework that generates a large number of candidate features with identified common operators and conducts meta-feature based ranking and the greedy-search evaluating approaches. AutoLearn [14] proposes a regression-based feature learning algorithm which constructs regression models to study the feature correlations. These correlation scores are deemed as new features to conduct feature selections after all features are generated.

On the other hand, FEature DIScovery (FEADIS) [5] employs a wide range of feature combinations and includes constructed features greedily. Cognito [17] expresses feature generation as a directed acyclic graph, which is called a Transformation Tree. Each non-root node in the tree represents a transformed dataset with newly generated features and is associated with a score that measures the importance. Several heuristics tree traversal strategies such as depth-first and balanced traversal could be done with it. [29] introduces Genetic Programming (GP) into feature generation and selection of high-dimensional classification problems, where feature selection selects the leaf nodes used to construct the GP tree. [30] presents AutoCross to automatically generate cross features for categorical features. AutoCross also ensures efficiency, effectiveness, and simplicity with mini-batch gradient descent and multi-granularity discretization.

Some novel techniques such as meta-learning and reinforcement learning are also applied on automatic feature generation. Learning Feature Engineering (LFE) [26] proposes a meta-learning method to evaluate the features' score on the Transformation Tree. It trains a classifier to recommend a set of useful transformations from historical feature engineering experiences on other datasets. [16] derives a strategy for efficiently searching the Transformation Tree by reinforcement learning (RL). Under a budget constraint, RL could be used for performance-driven exploration of the tree.

All researches above only consider single table, Data Science Machine (DSM) [12] is the first end-to-end system that automates feature engineering for relational tables. The core of DSM is Deep Feature Synthesis (DFS), which automatically generates features from interconnected tables. In addition to the DFS-based feature generation method, DSM autotunes an entire machine learning pipeline, including data preprocessing, Truncated SVD based feature selection, and automatic model selection with Bayesian Copula Process for Bayesian hyperparameter optimization. One Button Machine [19] extends DSM to handle large datasets while compensating for the disadvantage of DSM to handle unstructured data. Neural networks are also taken into account as [18] proposes a novel Relational Recurrent Neural Network (R2N) that maps the relational tree to a target value. However, the generated features are not interpretable.

3 Method

Let \mathcal{D} be a set of relational tables, where each entry is related to at least another table in \mathcal{D} . Assume any relationship is treated as an edge, and one or more connected edges started from target table constitute a path p . The path set \mathcal{P} includes all possible paths. Suppose $g_p^{(t)}(\cdot)$ is an arbitrary mapping function that takes any features X as input and uses the transformation t along the path p . Then the result feature set F could be written as $\mathcal{F} = \{F : F = g_p^{(t)}(X), \forall \text{ path } p, \text{ transformation } t\}$. To simplify the setting, we assume each table could be split into the training data set \mathcal{D}_{train} and validation data set $\mathcal{D}_{validate}$ by a universal criterion. Assume $\ell(\cdot)$ is the loss function (such as RMSE or F1-score) and $L(\cdot)$ is a learning algorithm. Our target is to derive:

$$F^* = \arg \min_{F \subseteq \mathcal{F}} \ell(L(\mathcal{D}_{train}, F), \mathcal{D}_{validate}, F) \quad (1)$$

In this manuscript we propose DAFEE, an automatic scalable feature engineering algorithm that is implemented on the distributed computation engine Spark [31]. Based on the relationship of tables, it connects all the tables and convey the information of each table to the target table through joining any two related tables in an ordered manner. Finally, DAFEE applies a feature selection after each time joining other tables to the target table, in order to reduce the number of features. In this section, we explain the motivation for DAFEE and introduce the algorithm in detail.

3.1 Motivation

Most of the researches on feature engineering [7, 24, 25, 28], especially automatic feature engineering [13, 17, 20, 30], only consider single table. However, real-world data is usually much more complicated. In most cases, features are derived from multiple tables. Consider the customer anomaly detection scenario in banking industry as an example. In order to predict whether a customer is abnormal, we need to fully utilize different aspects of information, such as customer profile and transactions. However, these data are usually collected in separate tables, making it impossible for automatic feature engineering algorithms mentioned above to handle. Besides, generating useful features, especially interaction features extracted from multiple tables requires domain knowledge and is time-consuming. In this manuscript, we propose DAFEE to tackle these problems.

3.2 Combination Strategy

To search and join related tables, a revised version of deep feature synthesis (DFS) [12] is used. In DFS, a table is called an *entity*, which is capable to handle numeric, categories, timestamps and free text features. We use the same concepts and symbols introduced in DFS.

A *forward* relationship could be simply regarded as a one-to-many or one-to-one relationship, while a *backward* relationship is just the opposite.

Direct features (DFEAT) are features directly delivered through *forward* relationships. Relational features (RFEAT) are generated by applying aggregation functions like MAX, MIN and SUM on a *backward* relationship. Different from direct features and relational features, Entity features (EFEAT) are created by taking transformations on the entity’s current features, regardless of any *forward* or *backward* relationship.

DFS uses a search strategy similar to depth-first-search, which starts from the target entity, traverses through the relation paths and stops searching when finding any leaf. On each leaf entity, entity features would be generated at first. Then direct features or relational features are generated on the leaf and joined to its parent according to the relationship between them. These operations would be taken sequentially until all features are joined back to the target entity.

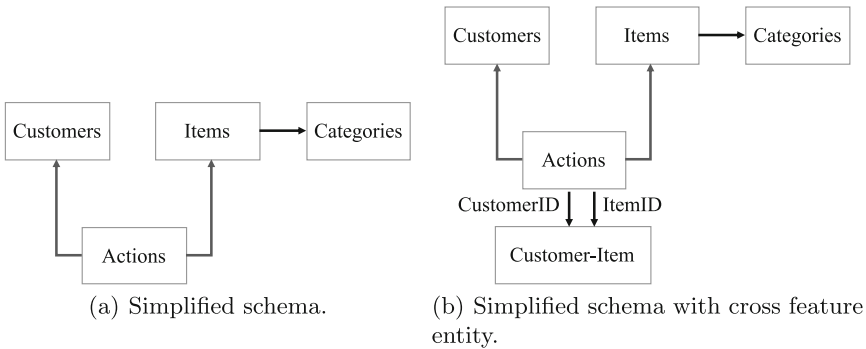


Fig. 1. Simplified Schema for e-commerce scenario.

However, this strategy is unable to generate some complicated but useful features. Figure 1(a) shows a simplified schema with four entities targeting for predicting actions of customers in an e-commerce scene. **Customers** entity contains personal information like age, gender, registration date, and so on. **Items** and **Categories** entities hold features describing items and categories. **Actions** entity is the entity with labels that records customers’ actions, such as browse and purchase. The direction of arrows represents the direction of relationships illustrated above. Complicated features, such as “*the number of products whose sales ranks 1st last month in its category that customer A bought in a week*”, are often used to describe the customer but could not be generated by DFS.

To solve this problem, we introduce **Combine** functions to join *forward* entity back to its father and create new features after generating relational features. With **Combine** functions, we can first construct relational features like “*whether the sales of this product ranks 1st last month in its category*” and then construct the feature illustrated in the last paragraph based on that. Besides, since a **Combine** function merges the visited *forward* entity into current entity, the entity tree is simplified as its depth reduces. In addition, we introduce more types of

relational features, such as Rank and Proportion features. Conditional features filtered by Time Windows and Interesting Values are also revised to make them more applicable.

Algorithm 1: Naive features generation

```

1 Function DAFEE( $E^i, E_M, E_V$ ):
2    $E_V \leftarrow E_V \cup E^i$ ;
3    $F^i \leftarrow F^i \cup \text{EFEAT}(E^i)$ ;
4    $E_F \leftarrow \text{Forward}(E^i, E_M)$ ;
5   while  $E_F \neq \text{null}$  do
6     for  $E^j \in E_F$  do
7        $F^j \leftarrow F^j \cup \text{RFEAT}(E^j, E^i) \cup \text{EFEAT}(E^j)$ ;
8        $F^i \leftarrow F^i \cup \text{DFEAT}(E^i, E^j)$ ;
9        $\text{Combine}(E^i, E^j)$ ;
10     $E_F \leftarrow \text{Forward}(E^i, E_M)$ ;
11   $E_B \leftarrow \text{Backward}(E^i, E_M)$ ;
12  for  $E^j \in E_B$  do
13    if  $E^j \cup E_V$  then
14      continue;
15    DAFEE( $E^j, E_M, E_V$ );
16     $F^i \leftarrow F^i \cup \text{RFEAT}(E^i, E^j)$ ;

```

Here E and F are entity and feature set corresponding. A^i indicates the i -th element of set A . A_V represents the subset of A annotated by V . For example, E^j presents the j -th entity while E_V stands for the set of visited entities.

3.3 Cross Features Generation

Another common scenario is the multi-key connection scenario, such as two entities connected by two separated keys. In the sample case illustrated in Fig. 1(b), the relationship between Actions entity and Customer-Item entity is a typical multi-key connection relationship since the two entities are connected both by CustomerID and ItemID. As DFS only supports one foreign key in each connection of entities, Customer-Item can not be connected to Actions directly. This causes information lost of the Actions entity in this case.

To support multi-key connection, we introduce a new relationship called *cross* relationship in DAFEE. Operations that are used to create relational features, such as MAX and MIN, are also used to create cross features upon a cross relationship. Besides, some more complicated operations that utilizes both backward and cross relationships, e.g. “*the proportion of milk A purchased by customer B among all milk brands that B purchases in a week*”, are also supplied. With cross features, the procedure of automated feature engineering is more close to manual feature engineering.

In general, we can create more meaningful features using DAFEE. However, at the same time, we also generate more useless features. To deal with such side effect, we introduce a trim algorithm to be explained in Sect. 3.4 and a feature selection module to be illustrated in Sect. 3.5.

3.4 Trim Strategy

A heuristic algorithm with two principle rules is used to trim the useless features. Both rules are introduced according to experts’ experiences, some useful features may be dropped, but with a negligible chance:

- The first rule discards those numeric transformations such as SUM or AVG on a date related feature. For example, “sum of the months of transaction dates” is a useless feature, which is an encryption over the month of transaction dates without further useful information.
- The second rule aims to trim cross features. If one entity has a *forward* entity and a *cross* entity that share one common foreign key, then the features generated through DFEAT from the *forward* entity will be passed directly to the *cross* entity. For example, in Fig. 1(b) the features in the Actions entity that are inherited from the Customer entity will be joined to Customer-Item entity directly without any RFEAT operations.

3.5 Feature Selection

Feature selection has been proven to be an effective and efficient way to reduce dimension and achieve a better result to solve many machine learning problems [22]. In general, feature selection algorithms could be categorized into three types: filter, wrapper and embedded [27]. Filter methods select features according to the intrinsic property of dataset, without using any classifier. Typical filter methods include similarity based methods [6, 9], information theoretical based methods [2, 21] and statistical based methods [4, 23]. Wrapper methods iteratively learn and predict feature scores by using certain machine learning algorithm and select part of them as outputs. One such commonly used wrapper method is recursive feature elimination (RFE) [8], which recursively ranks features by the importance score. Embedded methods integrate the feature selection process with the learner training process, and complete both processes in the same optimization stage.

Since the label information is included, and embedded methods usually take less time than wrapper methods, an embedded methods based on XGBoost [3] is used. XGBoost is an end-to-end tree boosting system that is widely used in different machine learning tasks. It also designs mechanisms to achieve high scalability, which is essential in handling the large dataset scenario. Moreover, its feature importance method using boosting trees has already been proven to be able to get good performance in industry [10]. Such feature importance is used to rank and output the top K (or p percent) of the features.

In order to control the number of features, feature selection is applied when the current entity is the target entity. Breadth-First-Search (BFS) allows more control on feature selections, which is also one of the reasons why a BFS type strategy is preferred to search forward entities to a Depth-First-Search alike strategy that are used in DFS. The final feature generation algorithm with trim strategy and feature selection is presented in Algorithm 2.

Algorithm 2: Feature generating

```

1 Function DAFEE( $E^i, E_M, E_V, target$ ):
2    $E_V \leftarrow E_V \cup E^i$ ;
3    $F^i \leftarrow F^i \cup \text{EFEAT}(E^i)$ ;
4    $E_F \leftarrow \text{Forward}(E^i, E_M)$ ;
5   while  $E_F \neq \text{null}$  do
6     for  $E^j \in E_F$  do
7        $F^j \leftarrow F^j \cup \text{RFEAT}(E^j, E^i) \cup \text{EFEAT}(E^j)$ ;
8        $F^i \leftarrow F^i \cup \text{DFEAT}(E^i, E^j)$ ;
9        $\text{Combine}(E^i, E^j)$ ;
10      if  $target == i$  then
11         $F^i \leftarrow \text{FeatureSelect}(F^i)$ ;
12     $E_F \leftarrow \text{Forward}(E^i, E_M)$ ;
13   $E_B \leftarrow \text{Backward}(E^i, E_M)$ ;
14  for  $E^j \in E_B$  do
15    if  $E^j \cup E_V$  then
16       $\text{continue}$ ;
17    DAFEE( $E^j, E_M, E_V, target$ );
18     $F^i \leftarrow F^i \cup \text{RFEAT}(E^i, E^j)$ ;
19    if  $target == i$  then
20       $F^i = \text{FeatureSelect}(F^i)$ ;
21   $E_C \leftarrow \text{CrossRelation}(E^i, E_M)$ ;
22  for  $E^j \in E_C$  do
23    if  $E^j \cup E_V$  then
24       $\text{continue}$ ;
25    DAFEE( $E^j, E_M, E_V$ );
26     $F^i \leftarrow F^i \cup \text{CFEAT}(E^i, E^j)$ ;
27    if  $target == i$  then
28       $F^i = \text{FeatureSelect}(F^i)$ ;
29  if  $target == i$  then
30     $F^i = \text{FeatureSelect}(F^i)$ ;

```

3.6 Implementation

Since the amount of data grows rapidly recently, it is necessary to implement algorithms on a scalable framework. Our system is based on Spark for the capability of distributed computation. Beside the high performance, Spark provides a module called Spark SQL [1], which offers sufficient commonly used operations, e.g. SUM, MIN, MAX, SELECT and JOIN, to process relational data. On top of Spark, we have developed additional operations like MODE and MEDIAN to fulfill the needs. All codes are optimized following the Spark code optimization skills, such as caching the intermediate results. Otherwise, the whole DAG may be computed multiple times.

Moreover, the mechanism to filter data based on some specified condition is implemented. Time Window and Interesting Value are two typical filters. A Time Window is a period of time, such as “1 day” and “3 months”. After specifying the time column and cutoff time, one can use the Time Window to filter data by time. For example, if one wants to generate “the number of transactions of each customer in 1 week”, one can filter “the number of transactions of each

customer” by Time Window “1 week”. Since data scientists usually generate the same features with different Time Windows, some commonly used Time Windows are implemented into a `TimeWindows` object in a convenient manner.

Interesting Value is another widely used filter. Data scientists may be interested in studying the conditions that one certain column’s value is equal to a specific value. One example is “the number of actions that a user put ABC items into shopping cart within 1 week”. If there is a column called “action type” with value “1” indicating “put into shopping cart”, then the interesting Value is the “1” value of column “action type”. One side effect is that the number of newly generated features grows dramatically as the number of interesting value increases. For example, if we have two Interesting Values, then the total number of generated features is nearly doubled. To solve this problem, we introduce a hyper-parameter called **target columns** for Interesting Values to restrict the columns that interesting values filters applied on.

4 Evaluation

We conducted experiments to demonstrate the performance and scalability of our automatic feature engineering approach. We mainly compared DAFEE with the DFS algorithm and its Python implementation `Featuretools`. The latter is the SOTA algorithm implementation and widely used by data scientists. To evaluate fairly, a four-node Spark cluster (2×8 cores E5-2620 V4 with 8×16 GB ram memory) is set up, where DAFEE and DFS are both ran in the yarn mode. The software environment is based on Linux 3.10.0-957 with Java (1.8.0-131), Hadoop (2.7.2) and Spark (2.4.3).

To justify the performance of our method, we test our implementation on four datasets with different sizes and configurations: **HI GUIDES**¹, **KDD Cup 2014**², **Coupon Purchase** (See footnote 2) and **IJCAI 2015**³. The completed machine learning pipeline is introduced in the following sections. For scalability, we compare the speedup on four different datasets mentioned above between our framework and `Featuretools` on Spark, which is the implementation of DFS based on Spark. All the experiments described in the following paragraphs are repeatedly conducted for three times to reduce the randomness. Average scores are taken as the final results.

4.1 Dataset Overview

Four datasets used in experiments will be described briefly in the chapter.

HI GUIDES predicts whether users will purchase quality travel services in the short term. There are five related tables: **UserProfile** describes users’ profile information; **Action** contains user behavioral information; **OrderHistory** includes

¹ <http://www.dcjingsai.com>.

² <http://www.kaggle.com>.

³ <http://ijcai15.org>.

users’ historical order information; **OrderFuture** contains user id and order type for training dataset; **UserComment** includes users’ evaluation and comments.

KDD 2014 Cup helps donorschoose.org to find the exciting projects that are more likely to be funded by website users. There are five related tables: **Projects** includes information regarding project itself and it’s corresponding teacher and school; **Essays** includes a short description and full essay of each project; **Donations**: contains information about donations on each project; **Resources**: contains requested resources of each project; **Outcomes**: contains outcome of each project with the label column “*is_exciting*”. Table projects, essays and resources provide both training and testing data, while donations and outcomes only give information regarding projects in training set.

Coupon Purchase predicts the coupons that a customer will buy in a certain period of time. The dataset contains a whole year’s information. There are five tables: **User_list** includes user profile and registration information; **Coupon_list** includes coupon information; **Coupon_visit** includes browsing log of coupons; **Coupon_detail** includes purchase log and more details of coupons; **Coupon_area** includes area information of coupons.

IJCAI 2015 predicts the repeated buyers for merchants in Tmall. The data could be extracted into four different table: **Users** includes user profile, including age range and gender; **Merchant** includes merchant ID; **Log** includes merchant id, and action information between users and items; **User_merchant** includes indicates whether a user repeatedly buy a merchant.

4.2 Preparation

To be comparable to DFS, we applied the same machine learning pipeline on DFS and DAFEE, except for the feature generation part. Figure 2 shows the full procedure. As DFS’s effectiveness highly relies on its parameter settings such as the choice of aggregation functions and interesting values, we carefully follow Featuretools’ guidance to ensure that the result reasonably reflects the true performance of DFS.

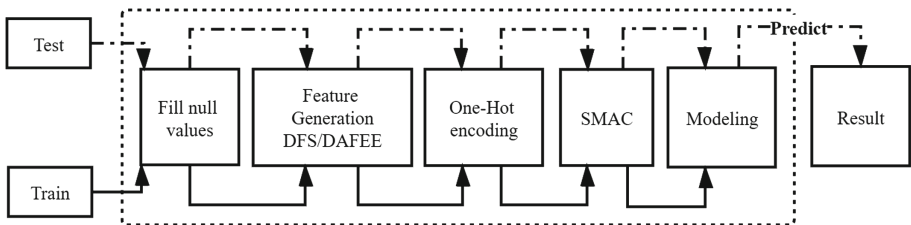


Fig. 2. Machine learning pipeline for effectiveness experiment.

In the preprocessing part, null values are filled with zeros at the beginning. One-hot encoding is put after feature generation to avoid the curse of dimensionality. In modeling section, we choose different model for different task. We use

random forest classifier for IJCAI 15 task while applying LightGBM [15] for other three tasks. In order to get the best models, we tuned their hyper-parameters automatically using SMAC [11], which is a widely used hyper-parameter optimization algorithm based on Bayesian optimization. The hyper-parameters we choose to tune are the same as defined in paper of DFS. Specifically, we set $n \in [50, 500]$ as the number of decision trees with default value 50; $m_d \in [1, 20]$ as the maximum depth of the decision trees without any default value; $\beta \in [1, 100]$ as the maximum percentage of features used in decision trees with default value 50; $rr \in [1, 10]$ as the ratio to re-weight underrepresented classes with default value 1.

5 Experimental Results and Analysis

In order to prove that our method is superior to DFS, we compared them on both the aspect of predictive performance and scalability. In this testing phase, DAFEE and DFS treat the four datasets in the same view (Fig. 3). The sample size and configuration information is shown in Table 1. The entity relationship of HI GUIDES and KDD Cup 2014 remain unchanged as they were illustrated originally, while those of the other two’s are revised slightly for a better performance. In Coupon Purchase, all features of `userCoupon` entity are discarded except for `user_list`, `coupon_list`, and `coupon_visit`. `userCouponPurchase` and `userPurchase` entities are generated from the `coupon_detail` table. By the way, cross features are only generated upon IJCAI15.

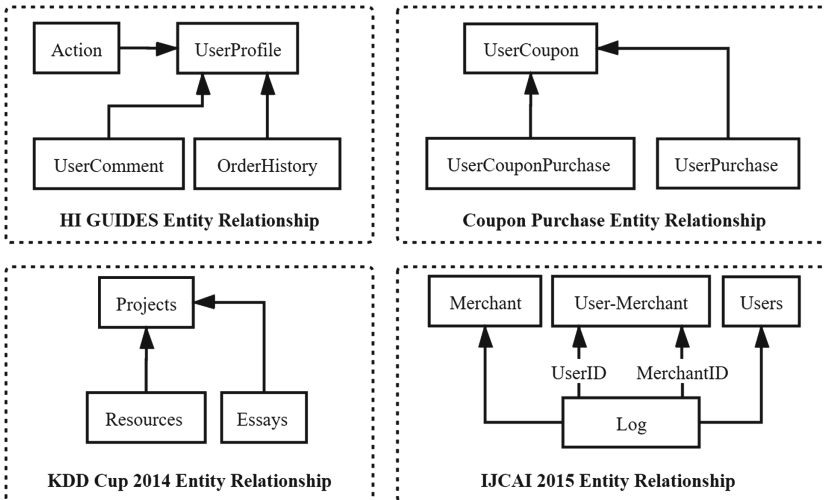


Fig. 3. The entity relationship of four datasets.

Table 1. The number of observations and features per entity in each experiment.

HI GUIDES			IJCAI 2015		
Entity	Rows	Features	Entity	Rows	Features
Action	1,666,060	3	Log	38,743,689	7
OrderHistory	27,512	7	Merchant	1,993	1
UserComment	12,337	5	User-Merchant	260,864	2
UserProfile	50,383	4	Users	212,062	3
KDD Cup 2014			Coupon Purchase		
Entity	Rows	Features	Entity	Rows	Features
Essays	664,098	2	UserCoupon	7,776,447	24
Projects	664,098	35	UserCouponPurchase	241,823	20
Resources	3,667,217	7	UserPurchase	168,996	13

5.1 The Performance of DAFEE and DFS

In this section, we compare the performance of DAFEE and DFS on four datasets in Table 1. Since IJCAI 2015 does not accept submissions any more, we use AUC score with 3-fold cross-validation as the performance metric. The results are illustrated in Table 2.

Table 2. The experiment performances. Online AUC for HI GUIDES and KDD Cup 2014, top 10 online MAP for Coupon Purchase and local AUC for IJCAI 2015.

Algorithm	HI GUIDES	KDD Cup 2014	Coupon Purchase	IJCAI 2015
DFS	0.84**0 ^a	0.61485	0.00434	0.647241
DAFEE	0.87150	0.6166	0.0053	0.66835

^aThe website shows the exact score as 0.84**0, which omits two decimal values.

It can be observed that DAFEE outperforms DFS on all of the four datasets. Specifically, the performance of DAFEE exceeds which of DFS **3.6%** on HI GUIDES, **0.3%** on KDD Cup 2014, **21.43%** on Coupon Purchase and **3.3%** on IJCAI 2015. The most significant improvement achieved on Coupon Purchase is over 20%. Obviously, DAFEE is able to generate more useful features than DFS, especially in the prediction of transaction types with time related variables.

5.2 The Scalability of DAFEE and DFS

In this experiment, the scalability of the DAFEE and DFS algorithm is measured by the speedup rate when the computation resource increases. Data preparation is applied beforehand to enable FeatureTools on Spark to work properly. Specifically, we divide each dataset into several partitions so that all worker nodes can

apply DFS in parallel. However, this method has two obvious shortcomings. First, if one non-target entity has a one-to-many relationship with the target entity, the samples in the non-target entity may be copied several times. Second, the data may be skewed if it is not uniformly distributed.

During the experiments, the number of driver cores and its memory is fixed to 1 and 16 GB. Furthermore, we set the total executor memory to 96 GB and the number of cores in each executor to 4 to ensure that the memory for feature expansion is sufficient. The number of executors are set to 2, 4, 6 and 8 to get the speedup curve. The number of data partitions is set to 96. Furthermore, two executors are treated as one node to make the graph more clear.

Since DFS segments the dataset before feature generation, the time of such segmentation should also be counted. Hence we provide both DFS Main time and DFS Total time, which stands for the time of main DFS process and the time including segmentation time. Because in the IJCAI 2015 experiment DFS failed to complete due to OOM exception, we only draw the speedup curve of DAFEE in Fig. 4.

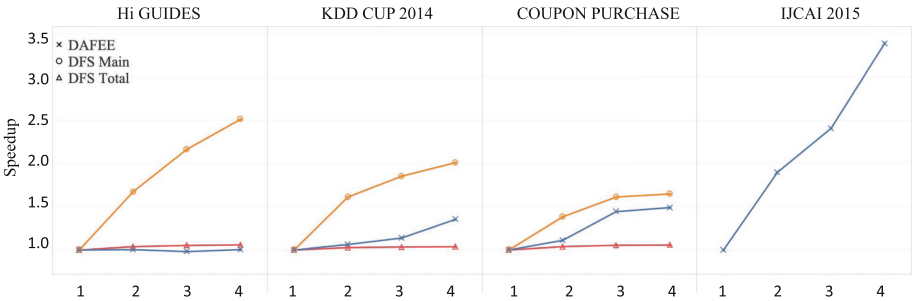


Fig. 4. The speedup performances (in times) of DAFEE and DFS main/total. 1, 2, 3, and 4 on the horizontal axis correspond to the cases where the number of executors is 2, 4, 6, and 8, respectively. The vertical axis is Speedup (run time/run time when executors equal to 2).

Since we manually assign the required data to each partition beforehand, we can find that in Fig. 4 the speedup of DFS Main is always the highest. However, after considering the overhead of data preparation, the speedup of DFS Total drops dramatically, which indicates that the data preparation takes the majority of execution time. On the DAFEE side, we find that it reaches its highest speedup, which is 3.388, on the large IJCAI 2015 dataset, while achieves a low speedup when the size of dataset is small. The reason may be that the communication overhead overwhelm that of computation when the dataset is small. Since the real data are usually huge, DAFEE may achieve a potential high speedup on them. To sum up, it can be inferred from Fig. 4 that DAFEE has a higher Speedup than DFS Total in most scenarios, especially when the amount of data is huge.

6 Discussions and Conclusions

In this paper we introduce DAFEE, a scalable distributed automatic feature engineering algorithm for multiple tables in real-world applications. By providing a novel search method with flexible time windows and new cross relationships, DAFEE can generate more useful features like proportional feature and high-order features than DFS to improve the prediction performance. Besides, DAFEE includes feature selection strategies to remove useless features and reduce the memory consumption. Additionally, experiment results demonstrate DAFEE is more scalable than DFS, which is valuable in the era of big data.

For the future work, we plan to dig deeper into the feature selection part. Since XGBoost used in DAFEE is a supervised learning algorithm, we can only apply feature selection on the target entity. To generalize the feature selection strategies, unsupervised feature selection methods other than XGBoost are needed. In addition, although the scalability of DAFEE is better than DFS, it is still not optimal. In the future, we intend to improve our algorithm and its Spark implementation to get better scalability. Finally, we plan to revise the framework to generate more useful high-order features.

References

1. Armbrust, M., et al.: Spark SQL: relational data processing in spark. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 1383–1394 (2015)
2. Battiti, R.: Using mutual information for selecting features in supervised neural net learning. *IEEE Trans. Neural Netw.* **5**(4), 537–550 (1994)
3. Chen, T., 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)
4. Davis, J.C., Sampson, R.J.: *Statistics and Data Analysis in Geology*, vol. 646. Wiley, New York (1986)
5. Dor, O., Reich, Y.: Strengthening learning algorithms by feature discovery. *Inf. Sci.* **189**, 176–190 (2012)
6. Duda, R.O., Hart, P.E., Stork, D.G.: *Pattern Classification*. Wiley, Hoboken (2012)
7. Guo, H., Jack, L.B., Nandi, A.K.: Feature generation using genetic programming with application to fault classification. *IEEE Trans. Syst. Man Cybern. Part B (Cybern.)* **35**(1), 89–99 (2005)
8. Guyon, L., Weston, J., Barnhill, S., Vapnik, V.: Gene selection for cancer classification using support vector machines. *Mach. Learn.* **46**(1–3), 389–422 (2002)
9. He, X., Cai, D., Niyogi, P.: Laplacian score for feature selection. In: *Advances in Neural Information Processing Systems*, pp. 507–514 (2006)
10. He, X., et al.: Practical lessons from predicting clicks on ads at Facebook. In: *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, pp. 1–9 (2014)
11. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration (extended version). Technical report TR-2010-10. Computer Science, University of British Columbia (2010)

12. Kanter, J.M., 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)
13. Katz, G., Shin, E.C.R., Song, D.: ExploreKit: automatic feature generation and selection. In: 2016 IEEE 16th International Conference on Data Mining (ICDM), pp. 979–984. IEEE (2016)
14. Kaul, A., Maheshwary, S., Pudi, V.: AutoLearn—automated feature generation and selection. In: 2017 IEEE International Conference on Data Mining (ICDM), pp. 217–226. IEEE (2017)
15. Ke, G., et al.: LightGBM: a highly efficient gradient boosting decision tree. In: Advances in Neural Information Processing Systems, pp. 3146–3154 (2017)
16. Khurana, U., Samulowitz, H., Turaga, D.: Feature engineering for predictive modeling using reinforcement learning. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
17. Khurana, U., Turaga, D., Samulowitz, H., Parthasarathy, S.: Cognito: automated feature engineering for supervised learning. In: 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW), pp. 1304–1307. IEEE (2016)
18. Lam, H.T., Minh, T.N., Sinn, M., Buesser, B., Wistuba, M.: Neural feature learning from relational database. arXiv preprint [arXiv:1801.05372](https://arxiv.org/abs/1801.05372) (2018)
19. Lam, H.T., Thiebaut, J.M., Sinn, M., Chen, B., Mai, T., Alkan, O.: One button machine for automating feature engineering in relational databases. arXiv preprint [arXiv:1706.00327](https://arxiv.org/abs/1706.00327) (2017)
20. Leather, H., Bonilla, E., O’Boyle, M.: Automatic feature generation for machine learning based optimizing compilation. In: 2009 International Symposium on Code Generation and Optimization, pp. 81–91. IEEE (2009)
21. Lewis, D.D.: Feature selection and feature extraction for text categorization. In: Proceedings of the Workshop on Speech and Natural Language, pp. 212–217. Association for Computational Linguistics (1992)
22. Li, J., et al.: Feature selection: a data perspective. *ACM Comput. Surv. (CSUR)* **50**(6), 1–45 (2017)
23. Liu, H., Setiono, R.: Chi2: feature selection and discretization of numeric attributes. In: Proceedings of 7th IEEE International Conference on Tools with Artificial Intelligence, pp. 388–391. IEEE (1995)
24. Markovitch, S., Rosenstein, D.: Feature generation using general constructor functions. *Mach. Learn.* **49**(1), 59–98 (2002)
25. Mitra, P., Murthy, C., Pal, S.K.: Unsupervised feature selection using feature similarity. *IEEE Trans. Pattern Anal. Mach. Intell.* **24**(3), 301–312 (2002)
26. Nargesian, F., Samulowitz, H., Khurana, U., Khalil, E.B., Turaga, D.S.: Learning feature engineering for classification. In: IJCAI, pp. 2529–2535 (2017)
27. Sheikhpour, R., Sarram, M.A., Gharaghani, S., Chahooki, M.A.Z.: A survey on semi-supervised feature selection methods. *Pattern Recogn.* **64**, 141–158 (2017)
28. Tang, J., Alelyani, S., Liu, H.: Feature selection for classification: a review. In: Data Classification: Algorithms and Applications, p. 37 (2014)
29. Tran, B., Xue, B., Zhang, M.: Genetic programming for feature construction and selection in classification on high-dimensional data. *Memetic Comput.* **8**(1), 3–15 (2015). <https://doi.org/10.1007/s12293-015-0173-y>
30. Yuanfei, L., et al.: AutoCross: automatic feature crossing for tabular data in real-world applications. arXiv preprint [arXiv:1904.12857](https://arxiv.org/abs/1904.12857) (2019)
31. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I., et al.: Spark: cluster computing with working sets. *HotCloud* **10**(10–10), 95 (2010)