Learn to Explore: on Bootstrapping Interactive Data Exploration with Meta-learning

Yukun Cao, Xike Xie, and Kexin Huang
University of Science and Technology of China
{ykcho, huang_1773}@mail.ustc.edu.cn, xkxie@ustc.edu.cn

***Abstract*—Interactive data exploration (IDE) is an effective way of comprehending big data, whose volume and complexity are beyond human abilities. The main goal of IDE is to discover user interest regions from a database through multi-rounds of user labelling. Existing IDEs adopt active-learning framework, where users iteratively discriminate or label the interestingness of selected tuples. The process of data exploration can be viewed as the process of training of a classifier, which determines whether a database tuple is interesting to a user. An efficient exploration thus takes very few iterations of user labelling to reach the data region of interest. In this work, we consider the data exploration as the process of few-shot learning, where the classifier is learned with only a few training examples, or exploration iterations. To this end, we propose a learning-to-explore framework, based on meta-learning, which learns how to learn a classifier with automatically generated meta-tasks, so that the exploration process can be much shortened. Extensive experiments on real datasets show that our proposal outperforms existing explore-by-example solutions in terms of accuracy and efficiency.**

***Index Terms*—Interactive data exploration, Few-shot learning, Meta learning**

I. Introduction

Interactive data exploration (IDE *in short*) [1] is at the frontline of big data management, which tackles data comprehensibility challenges caused by fast data accumulation and limited human ability. The problem of IDE is challenging, because: 1) user interest is intangible so that incremental refinement/exploration of user interests is required [2]; 2) user interest is indescribable in the sense that it is often too complex to be specified by a user through traditional query languages (e.g., SQL) [3].

For example, Alice and Bob explore sky objects in the Sloan Digital Sky Survey (SDSS) database[1]. Alice is an amateur astronomer, and her familiar attributes are relatively limited, $\{rowc, colc, ra, dec\}$. However, her data interest is so uncertain that it is hard for her to express accurately. Alternatively, she can browse and selectively label some database tuples so that the recommendation of tuples or queries reflecting her interests can be enabled by IDEs. Bob is an astronomical scientist whose data interest covers a wide range of attributes, $\{rowc, colc, ra, dec, sky\_u, sky\_g, ...\}$[2]. However, his requirements (e.g., mathematical expressions involving multiple attributes) are too complex to be expressed by conventional database queries, and even database experts take much time to write dedicated filters. But it is easy for Bob to label whether a specific data tuple meets his needs.

Following explore-by-example paradigm [2], [4], [5], the main goal of IDE is to discover user interest regions (UIR *in short*) from a to-be-explored database through multiple rounds of user labelling. The exploration process can be viewed as the training process of classifiers [3], deciding if a database tuple is "interesting" to a user. The output of IDE refers to arbitrarily attainable data query regions, covering user interested tuples in the explored database. Technically, the indescribability brings in the challenge of generality in UIR representation [3]. The intangibility brings in the challenge of "slow convergence". For example, hundreds of iterations of labelling is needed to converge to a UIR [2].

TABLE I
Evolution of IDE under "Explore-by-Example " Paradigm

	UIR in subspace	Classifier	Techniques
AIDE [2], [4]	Linear	Decision Tree	Active-Learning
DSM [5]	Convex	SVM	Active-Learning
LTE	Arbitrary	Neural Networks	Meta-Learning

In general, a high-performance IDE is expected to achieve both high efficiency and accuracy. The efficiency refers to human efforts expended on the "interestingness" labelling. The accuracy refers to the closeness between the inferred UIR and the real one. The pursuit of efficiency and accuracy can also be observed from IDE technology evolution, in Table I. A better classifier leads to a faster convergence; and a better UIR representation leads to a higher accuracy, which in turn prompts the classifier training, i.e., exploration efficiency.

It is thus a natural evolution of IDE classifiers, emanating from machine learning, e.g., decision trees [2], [4], support vector machines (SVM *in short*) [5], and taking shape in deep learning, e.g., neural networks (NN *in short*). The NN has good capabilities in capturing abstract feature representation in the manner of stacked layers, with a good match to user interest exploration which is intangible and indescribable. Despite the potential accuracy, there are several challenges: the NN classifier relies on a large number of user labels for training, which implies more exploration iterations and human efforts, and thus slower convergence, contradicting the efficiency target.

To this end, we propose to boost NN classifiers by meta-

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

[2]These attributes are the photometric attributes of sky objects. Details are in: https://skyserver.sdss.org/

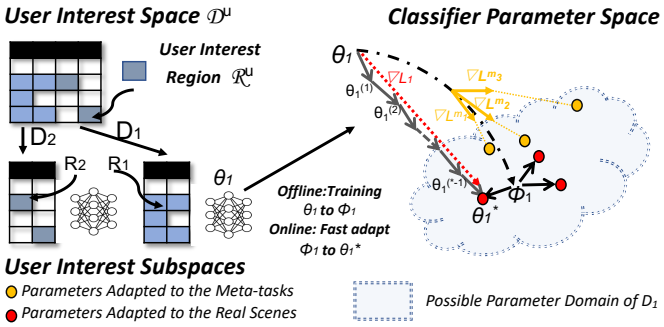


Fig. 1. An Example of LTE

learning[3] for IDE systems. The mechanism of meta-learning is also characterized as few-shot learning by literatures [7], which shows a good match to the cold start of data exploration, where labels are rare and precious. We call the meta-learning supported NN classifier the *meta-learner*, which is pre-trained with automatically generated synthetic *meta-tasks*. The pre-training equips meta-learners with good initialization parameters, so that they can be quickly adapted and generalized during the initial exploration phase. Unlike conventional training strategies (e.g., active learning [4], [5]), it takes merely a few gradient optimization steps for classifiers to converge, corresponding to fewer iterations of user labelling, and therefore higher efficiency during the online exploration. From this point, the meta-learning process can be viewed as "learning to explore" (LTE *in short*), which takes few-shot of labelling for a quality initialization of IDE. The LTE framework has two features, in addition to high accuracy and efficiency.

First, the pre-training processing of meta-learners in LTE is unsupervised [8], so that it does not incur the overhead of user labelling. Meanwhile, it extracts data distributions and insights from lightweight sampled tuples set, represented as meta-knowledge, for initializing meta-learners with good parameters. Figure 1 shows the idea of meta-training. Under regular training settings, a classifier's parameter θ_1 would be trained by backpropagation which computes the gradient decent of a loss function, e.g., ∇L_1, and goes through a long optimization path $\langle \theta_1^{(1)}, \theta_1^{(2)}, ..., \theta_1^{(*-1)} \rangle$ to converge to an optimal θ_1^*. It may not be cost efficient, since each gradient optimization step requires user labelling. With meta-learning, the classifier parameter θ_1 is pre-trained to ϕ_1, which can be quickly adapted to parameter θ_1^* with much less optimization steps so as to reduce user labels, during the online exploration.

More, it supports concave or even scattered user interest regions. The motivation is towards tackling the indescribability challenge, under which user interests are somehow hard to be explicitly expressed in SQL templates or filters. Figure 1 shows an example of UIR, which is shaded in a tabular dataset. Given a multi-dimensional dataset, an IDE system [5] usually decomposes it into multiple low-dimensional datasets. For example, data space $\mathcal{D}^u$ is decomposed into subspaces D_1 and D_2. The projection of UIR on each subspace can be of arbitrary shapes. For example, R_1 is concave on D_1, and R_2 is a scattered region on D_2. Therefore, a powerful classifier is needed to deal with the generality setting of UIR. Unlike existing works (in Table I), we do not make assumptions on UIR shapes.

Our main contributions are summarized as follows.

- We propose, to our best knowledge, the first "learn-to-explore" framework, that harnesses meta-learning based neural network classifiers for data exploration.
- The meta-learner of the LTE framework is pre-trained with automatically generated meta-tasks, so that only a few gradient optimization steps are needed during the online exploration, leading to less exploration iterations and human efforts.
- Experiments on real datasets demonstrate that our proposal outperforms existing explore-by-example solutions in terms of accuracy and efficiency.

The rest of the paper is organized as follows. Section II summarizes related works. Section III presents basic concepts and the LTE framework. Section IV investigates the meta-learning process. Section V studies the generation of meta-tasks. Section VI investigates the training process of meta-learners. Section VII introduces other critical techniques of the LTE framework, i.e., tabular data preprocessing and few-shot optimization. Section VIII reports experimental results and Section IX concludes the paper.

II. Related Work

Interactive Data Exploration. Interactive Data exploration is about how users can extract knowledge from data using system assistance and interactive guidance, when they do not have exact query requirements [1], [3], [9]. Early works focus on simple user interactions. For example, [10], [11] requires users to gradually provide interest attribute values to drill down and finally return interest tuple set. Some works support the exploration with error guarantees and response deadlines for specific data types and query templates [12]–[14]. Some works study preparatory data exploration with the support of online analytical processing [15]–[19]. In addition, there are works on the exploration result visualization [20], [21] and query formulation [22], [23].

Numerous recent researches aim to expand the diversity of exploration modes, especially by utilizing machine/deep learning to optimize/model various exploration modules. The "explore-by-example" systems [4], [5] are designed to discover user interest regions through tuple-level user labeling and employ active learning to improve interaction efficiency. "Insights" driven systems [24]–[26] formalize interesting patterns (including correlations, anomalies, trends, etc.) in multidimensional data as "insights" and propose some interactive exploration frameworks for "insights". Automated exploratory data analysis (EDA) systems recommend an exploration path for users, which generally requires predefined exploration modes

[3]A typical method such as MAML (Model-Agnostic Meta-Learning [6] has the objective of learning an appropriate model initialization parameter in a range of meta-tasks.

and various types of user interactions. Two representatives, ATENA [27] and Dora [28], utilize deep reinforcement learning to model the EDA process. ExplainED [29] automatically generates semantic explanations for each step of the EDA process to guide the user's exploration, by natural language processing (NLP) techniques. Our work is under the explore-by-example paradigm, which is considered as complementary systems to EDA systems [27].

In addition, some works are oriented toward specific exploration data types, such as graphs [30]–[32], spatio-temporal data [33]–[35], and time series [36], [37]. Some works [38], [39] focus on optimizing the visualization experience during interactive exploration.

Explore-by-Example. IDEs under this paradigm [2] originate from the research of "query by example" [40], which recommends selective tuples in the databases as proxies for exploration targets. The latest IDE frameworks [4], [5] regard the exploration process as an incremental classification problem, and employ active learning to select the tuples that are most difficult to "discriminate" for users to label. However, due to the limitation of classifiers and the bottleneck of active learning, these frameworks focus on specific exploration targets. For example, the state-of-the-art, DSM [5], assumes subspatial convexity and conjunctivity of UIRs. Our work bootstraps the explore-by-example IDE paradigm, aided by meta-learning, for better exploration efficiency.

Meta-Learning. It is often known as "learning to learn", which seeks to gain meta-knowledge from a set of machine learning tasks in order to improve the learning process [7]. It belongs to the scene of few-shot learning [41]. We focus on a type of meta-learning method, which learns good initial parameters for meta-learners with meta-tasks. A typical meta-task (e.g., MAML [6]) has both training and validation data, called *support set* and *query set*, respectively. During the meta-training, the meta-learner iterates over the meta-tasks. At each iteration, a local learner is trained on the support set and tested by the query set. The meta-learner's parameters are then globally updated according to aggregated backpropagated loss measured by local testing errors. Our work belongs to a challenging topic of unsupervised learning via meta-learning [8], since meta-tasks are generated without label sets.

III. Overview

We introduce basic concepts in Section III-A, show the framework overview in Section III-B.

A. Basic concepts

User Interest Space. Suppose a database consisting of a set of attributes $A = \{a_1, ..., a_{|A|}\}$ and a set of $|A|$-dimensional tuples. We define the domain space formed by attribute set A as $\mathcal{D} = \{domain(a_1) \times domain(a_2) \times ...domain(a_{|A|})\}$, which covers all the database tuples. A user u is interested in a subset of attributes $A^u \subseteq A$, of which the domain space can be represented by $\mathcal{D}^u = \{domain(a_1^u) \times ...domain(a_{|A^u|}^u)\}_{a_j^u \in A^u}$, called *user interested space*.

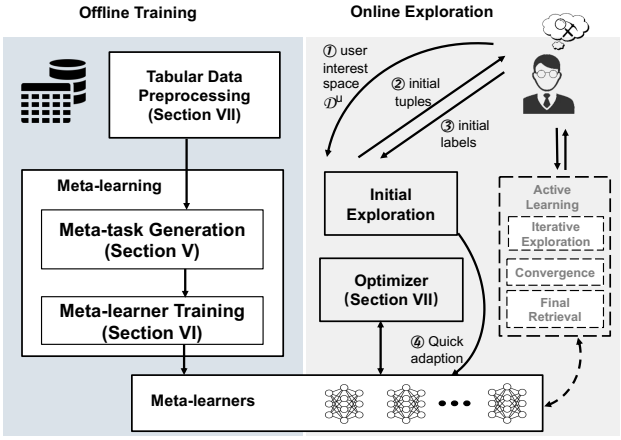


Fig. 2. Overview of Learn-to-explore Framework

User Interest Subspace. The exploration target is to browse $\mathcal{D}^u$ for retrieving tuples interesting to user u. Existing IDEs [5], [42] decompose $\mathcal{D}^u$ into a set of disjoint low-dimensional *subspaces* $\{D_i\}_{i \leq n}$, where $\mathcal{D}^u = D_1 \times ... \times D_n$.

User Interest Subregion (UIS). Given a user interest subspace D_i, the UIS $R_i \subseteq D_i$ can be defined as the tuples $\{\tau \in D_i\}$ satisfying a user's interest. If user u's exploration interest on D_i is represented by a binary classifier $\mathcal{I}_i : D_i \rightarrow \{0, 1\}$[4], UIS R_i can be represented by $R_i = \{\tau \in D_i | \mathcal{I}_i(\tau) = 1\}$.

User Interest Region (UIR). Essentially, a user interest region $\mathcal{R}^u$ for user u is the conjunctive combination of its subregions, $\mathcal{R}^u = \vee_{i \leq n} R_i$. The target of data exploration is to efficiently and accurately approximate UIR $\mathcal{R}^u$, determined by some prediction models, e.g., a classifier, for each of the n subspaces, acting as $\{\mathcal{I}_i\}_{i \leq n}$.

B. LTE Framework

A bird's-eye view of LTE framework is shown in Figure 2. It consists of four functional modules, *Meta-learning* , *Initial Exploration*, *Preprocessing*, and *Optimizer* modules, operating in and two phases, *offline training* and *online exploration* phases.

Meta-Learning is the core module of offline training phase. Its functionality is on training meta-learners by automatically generated meta-tasks. As aforementioned, the data space of a database is decomposed into a set of subspaces, which we term *meta-subspaces*, $\{D_i^M\}$. The components *meta-task generation* is in charge of generating meta-tasks, each of which contains a *support set* and a *query set* (Section V) of a meta-subspace. Then, the *meta-learner training* locally updates the meta-learner by support sets, and globally updates the meta-learner by query sets (Section VI).

Initial Exploration is the core module of online exploration phase. At the initial stage of data exploration, a user first selects his/her interesting attributes from the database schema to form a user interest space $\mathcal{D}^u$. The $\mathcal{D}^u$ is decomposed into a set of subspaces mapped to meta-subspaces. Then, he/she is presented with a selected set of initial tuples of UIS for

[4]Let 1 be "interesting", and 0 be "not interesting".

labelling, the number of which is constrained by a given budget. The selection of initial tuples is similar to the support set construction during meta-learning training (Section VI). Then, user labels are collected[5]. Finally, user labels are fed to pre-trained meta-learners on-the-fly to fast adapt to the real user interests. The adapted meta-learners can determine the result UIR.

Preprocessing is to convert an input tabular dataset into a series of composite vectors that can be fed to meta-learners, i.e., neural networks. The input of the module is a sampled database, achieving good feature representability and data scalability. The output of the module is feature-rich and high-dimensional vectors, conforming to the input of NN training. Details are reported in Section VII.

Optimizer is heuristically dedicated to adjusting UIS predicted by each meta-learner in few-shot exploration. For each subspace, the module takes user labeled tuples during the initial exploration as input. After that, it takes two optimization steps for reducing false positives and false negatives, in order to polish the prediction results. Details are covered in Section VII.

Other IDE Modules. Notice that our LTE framework can also be plugged to existing IDE systems [2], [4], [5] by connecting the trained meta-learners to active learning mechanisms. For being self-contained, we also briefly review existing IDE modules [2], [4], [5] that can be combined with our LTE framework to make a complete system: *1) Iterative exploration.* If a user wants to continue exploring after the initial exploration phase, active learning can be employed to feed more labelled tuples to the meta-learner for further training [43]. *2) Convergence.* The user can set budgets for labelling, or use data visualization methods [20], [21], [44], [45] to determine whether the exploration should be stopped. If such prerequisites are made, our framework can incorporate additional indicators (like *three-set metric* in [5]) for supporting the determination of exploration convergence. *3) Final retrieval.* An IDE system returns a sampled (or complete) set of user interest tuples, or infers corresponding query regions based on trained classifiers. The results can also be transformed to query filters (e.g., in SQL), if prerequisite assumptions about UIR and query templates are made [2], [22], [23], [46], [47].

IV. Meta-learning Process

A. Concepts

The core of the LTE framework is the meta-learning process, which is formalized as follows.

*Definition 1 (**Meta-learning Process**):* Given a pre-defined meta-subspace D^M, a meta-task set $\mathcal{T}^M$, and a meta-learner $\mathcal{C}_\theta^M$ with randomly initialized parameter θ, the meta-learning process can be modeled as a function F_M, as follows.

$$F_M(\mathcal{T}^M, \mathcal{C}_\theta^M) \rightarrow \mathcal{C}_\phi^M \tag{1}$$

[5]As pointed in [5], collecting user's labelling feedback belongs to the field of human-computer interaction and is beyond the scope of this paper.

Here, ϕ refers to an initialization parameter trained with meta-task set $\mathcal{T}^M$, and $\mathcal{C}_\phi^M$ refers to the learned meta-learner equipped with ϕ. A quality ϕ helps $\mathcal{C}_\phi^M$ in efficiently approaching towards the optimization target during the online exploration phase. Next, we formalize the concept of a meta-task.

*Definition 2 (**Meta-task**):* A meta-task t of a meta-task set $\mathcal{T}^M$ has three parts, a simulated UIS R_t^M, a support set $\mathcal{S}_t^{sp}$ and a query set $\mathcal{S}_t^{qs}$.

$$t : (R_t^M, \mathcal{S}_t^{sp}, \mathcal{S}_t^{qs}) \tag{2}$$

Following typical meta-learning settings [6], [7], a meta-task is expected to be associated with a support set and a query set, whereas the support set is used for updating the meta-learner with local updates and the query set is used for updating the meta-learner with global updates. Specifically, for data exploration, a meta-task t is associated with a simulated UIS R_t^M, which is automatically generated.

Then, of a meta-task t, both the corresponding the support set ($\mathcal{S}_t^{sp}$) and query set ($\mathcal{S}_t^{qs}$) consist of a certain number of labelled tuples, where a tuple is labelled by checking whether it is within the UIS. Differently, the support set is for simulating user actions of labeling during the exploration, and the query set is for simulating the evaluation of the trained meta-learner.

In Figure 3, t_1 is a meta-task on meta-subspace D_1^M, which consists of UIS (the shaded region), a support set (red points), and a query set (yellow points). Note that meta-tasks of the same meta-subspace can share the tuples of the support and query sets. For example, meta-tasks t_1 and t_2 offer different coverages to the same set of red and yellow points.

B. The Learning Process

The meta-learning process can be viewed as a search optimization problem on the parameter space of the meta-learner, i.e., the domain of the parameter matrix of a neural network. The parameter space is huge [7]. It is known that a quality initialization parameter enables a fast convergence to the optimization target [6] of the huge parameter space.

Intuitively, the initialization parameter is expected to have optimization distances uniformly close to the parameters corresponding to real tasks (during the online exploration). However, the challenges are two-fold: 1) the real tasks are of high diversity, which cannot be enumerated during the pre-training; 2) the parameter space is enormous, whereas conventional search optimization methods fall short.

In our work, the former is addressed by automatically generated meta-tasks, consisting of a set of simulated UISs on a meta-subspace (*meta-task generation* in Section V). The latter is addressed by a meta-training algorithm under the "gradient by gradient" setting, taking quadratic derivation for the optimization of initialization parameters (*meta-training* in Section VI).

An example of the meta-learning process is shown in Figure 3. Suppose a meta-learning process on meta-subspace D_1^M, which is to find an initialization parameter (ϕ_1) from a

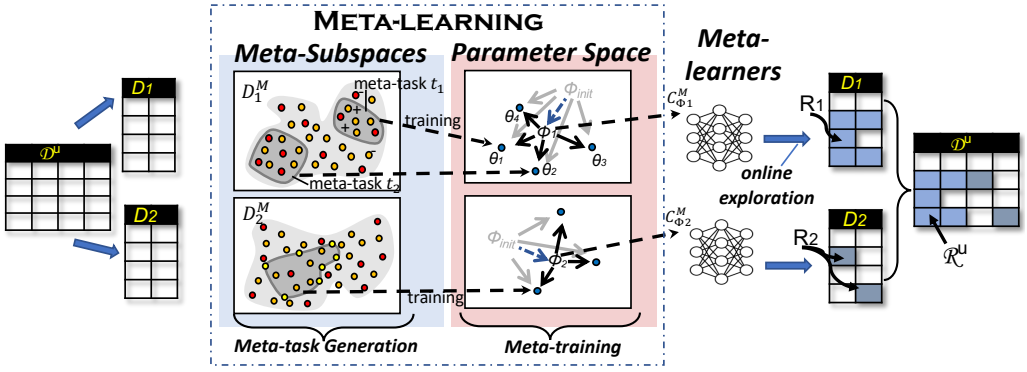


Fig. 3. The Meta-learning Process

random parameter (ϕ_{init}). It is expected that the optimization distances from ϕ_1 to "anchor parameters" (e.g., $\theta_1 \sim \theta_4$) are uniformly close. An anchor parameter corresponds to a meta-task (e.g., θ_1 corresponds to t_1), which can be obtained through the meta-training with its corresponding meta-task. The functionality of an anchor parameter is to guide the direction of the search optimization of the meta-learning. Thus, the quality of meta-learning process depends on the quality of meta-tasks. Next, we discuss how the meta-tasks are generated.

V. Meta-task Generation

A. Meta-task Generation Algorithm

In this section, we investigate how meta-tasks can be automatically generated for each meta-subspace, so that meta-learning can be enabled in an unsupervised way. In general, simulations require representing the key characteristics or behaviors of the data exploration process, so that two principles can be summarized for designing a meta-task.

- *Faithfulness:* The tuples of support/query set should conform to the data distributions of a given meta-subspace, making a basis for the quality inference of UISs. Otherwise, the "bias" can be prorogated to the anchor parameters, thus affecting the optimization of initial parameters.
- *Generality:* A meta-task should be flexible in covering different tendencies of UISs, i.e., being general in putting together different pieces of data interests of a meta-subspace.

For the first principle, a natural solution is to sample tuples conforming to the data distribution of a meta-subspace. In this paper, we opt to a clustering-based sampling method, (e.g., k-means [48]), which is proved to be primitive and effective for summarizing data insights [2].

For the second principle, we construct a general form of UIS, which can be represented as the composition of any set of convex parts on a meta-subspace, thus being general in supporting arbitrary shaped UISs, according to the convex decomposition theory [49], [50].

The overall process about meta-task generation is formalized in Algorithm 1. In general, the process consists of two steps, clustering step (Section V-B) and task generation step, where the latter contains UIS formulation (Section V-C) and support/query set formulation (Section V-D).

Algorithm 1 Meta-task generation

Input: Parameters k_u, k_s, k_q, α and ψ
Output: a meta-task set $\mathcal{T}^M$

1: Perform three rounds of k-means clustering (k= k_u, k_s, and k_q), get cluster center sets C^u, C^s, and C^q, and calculate P^u (Section V-B);
2: Generate UISs based on C^u, P^u, α and ψ (Section V-C);
3: Get Support/Query set on C^s, C^q and UISs (Section V-D);
4: Collect UISs and corresponding support/query sets to get $\mathcal{T}^M$.
5: **return** $\mathcal{T}^M$;

B. Clustering Step

Clusters (or cluster centers) can be viewed as a lightweight summary of a meta-subspace. During the clustering step, we perform k-means clustering independently for three rounds[6], because a meta-task t is a triple, i.e., $t : (R_t^M, \mathcal{S}_t^{sp}, \mathcal{S}_t^{qs})$. Each round is with a different parameter k (i.e., k_u for simulated UIS, k_s for support set, and k_q for query set). Accordingly, we get three sets of cluster centers $C^u = \{c_i^u\}_{i \le k_u}$, $C^s = \{c_i^s\}_{i \le k_s}$, and $C^q = \{c_i^q\}_{i \le k_q}$, based on which the simulated UIS, support set, and query set of a meta-task are generated.

During the clustering, we maintain two proximity matrices for the efficiency of subsequent steps, P^u and P^s, based on C^u and C^s. The first matrix P^u stores $k_u \times k_u$ elements, representing the distances between the k_u cluster centers of C^u, for constructing the simulated UIS (Section V-C). The second P^s stores $k_s \times k_u$ elements, representing the distances between the k_s cluster centers in C^s and the k_u cluster centers in C^u, used to expand the feature vectors of UIS (Section VI-A) and few-shot prediction optimization (Section VII). Without loosing generality, *Euclidean distance* is employed for measuring the proximity. The proximity matrices can be done in $O(k_u^2 + k_s \cdot k_u)$.

C. UIS Formulation

The generated meta-task set is expected to offer good coverage of UIS, which can be arbitrarily shaped in low-dimensional spaces. According to the convex decomposition theory, a region (or UIS) can be viewed as a combination of multiple intervals of different lengths in a 1D subspace or multiple 2D convex polygons in a 2D subspace, and similarly in higher dimensions. Therefore, the UIS of a meta-task can be formulated by randomly combining a set of convex shaped

[6]The clustering is run on a randomly sampled (1%) subset of the tuples of the meta-subspace for scalability.

parts on a meta-subspace. In this paper, we implement an efficient and straightforward generation method, which utilizes sampled tuples to construct multiple external convex regions to combine into the final simulated UIS. Moreover, we can control the size of a part and the number of parts, by changing the selection of the sampled tuples.

To generate a meta-task, we start by constructing a simulated UIS, which requires three steps. First, we randomly select a cluster center $c_j \in C^u$, and retrieve the set S_j of c_j's ψ nearest neighbors (i.e., cluster centers), where $S_j = \psi NN(c_j)$ and $S_j \subseteq C^u$. It can be done with the proximity matrix P^u in $O(k_u)$.

Second, we build the *convex hull* for S_j, represented by $Cvx(S_j)$, which is the largest circumscribed convex polygon for the cluster centers. Notability, there can be other options for the circumscribed region, such as minimum bounding rectangles or circles. It can even be concave, as long as the selected cluster centers are circumscribed. In our implementation, convex hulls are adopted for their simplicity. The convex hull serves as the basic building block of a UIS, which can be done in $O(\psi \cdot log(\psi))$. The first two steps are repeated until α convex hulls are collected.

Finally, the α convex hulls are combined to get a simulated UIS, $R_t^M = \bigcup_{j \leq \alpha} Cvx(S_j)$. Notice that the UISs in existing works can be viewed as special cases generated by the above method. For example, [5] assumes the UIS as a connected convex region ($\alpha = 1$). In our implementation, we do not explicitly maintain the exact shapes of R_i^M. All we need is to determine, during the offline training, if a point of the meta-subspace is within the given UIS, which can be transformed into determining if a point is located within any of the α convex hulls. It can be done in $O(\alpha \cdot log(\psi))$. In empirical studies (Section VIII-C), we also consider different combination of ψ and α as a UIS mode, and examine the performance on various modes.

D. Support/Query Set Formulation

We can use a generated UIS to formulate the corresponding support and query sets. we first take the k_s cluster centers from C^s as tuples of the support set. The label y of each tuple is determined by checking if it belongs to the corresponding UIS. To increase the generality of meta-training, we further sample a few tuples randomly from the meta-subspace. Therefore, the size of the final support set is $k_s + \Delta$[7]. The query set is built in a similar way on C^q. The size of the query set is $k_q + \Delta$.

Notably, since the role of the support set is to simulate the set of tuples labeled by users, the initial tuples for online exploration in the LTE framework are also generated by the clustering step, for a subspace. Then, for each tuple of a subspace, a user needs to label it for initial exploration[8].

[7]The default Δ is 5 in our implementation.

[8]If there exist assumptions on the UISs of subspaces [2], [4], [5], we can put the subspaces that have a conjunctive relationship into a group to reduce the number of subspaces labeled by the user.

E. Discussion

Dynamic Maintenance. The meta-learner is trained on the basis of meta-tasks, and meta-tasks are built on sampled tuples. So, one only needs to check if sampled tuples should be updated to decide if the meta-tasks and meta-learners should be updated, when the data distributions of the meta-subspaces change. Then, the problem is reduced to check for each subspace whether its corresponding clustering results violate, if the exploratory database is updated. The solution is to capture the locality of dynamic changes to data distributions of subspaces, corresponding sampled tuples set, and meta-tasks. To this point, existing works of dynamic clustering [51], [52] can be applied. Details about dynamic clustering are beyond the scope of the paper are omitted due to page limits.

Splitting Data Space to Meta-subspaces. The data space should be split into a set of mutually exclusive meta-subspaces in the offline phase. One may establish as many meta-subspaces as possible, for the matching with subspaces specified in the online phase. However, one may need to generate $\binom{|A|}{d}$ d-dimensonal meta subspaces for covering all possibilities of splitting a $|A|$-dimensional space, which can be costly. On one hand, we can determine some commonly used meta-subspaces based on the semantic/dependency relationship between attributes, or logs of user exploration. On the other hand, even if the meta-learner is not used for the subspace, the basic NN classifier combined with tabular data preprocessing still achieves better performance than existing methods (Section VIII-C). In our implementation, the domain space is randomly split into meta-subspaces, because we assume zero knowledge about data semantics and user priors.

VI. Meta-learning training

Given a meta-subspace D^M with the meta-task set $\mathcal{T}^M$. Each meta-task $t \in \mathcal{T}^M$ contains a simulated UIS R_t^M, a support set $\mathcal{S}_t^{sp}$ and a query set $\mathcal{S}_t^{qs}$. Both query and support sets are composed of a set of 2-tuples $\{\tau, y_{R,\tau}\}$, where $\tau \in D^M$ is a meta-subspace tuple, and $y_{R,\tau}$ is the label indicating whether τ belongs to UIS R_t^M of t. The meta-training goal is to find suitable initialization parameters ϕ, so the neural network classifier $\mathcal{C}_\theta^M$ can fast adapt to $\mathcal{C}_\phi^M$.

We first introduce a UIS classifier based on neural networks in Section VI-A, and memory-augmented optimization for meta-learning in Section VI-B. After that, we propose the meta-learning algorithm in Section VI-C.

A. Basic UIS Classifier

We introduce a NN classifier, which contains three building blocks: *UIS feature embedding block*, *Data tuple feature embedding block*, and *Classification block*.

UIS Feature Embedding Block (f_{θ_R}). To enrich the input features of the classifier in the few-shot exploration. We construct a $0/1$ vector of length k_s from the set C^s, where each vector bit corresponds to a cluster center in C^s. The bit is assigned to 1, if the user is interested in the corresponding cluster center, and 0 otherwise. Notice that the bit position of the vector representing a cluster center is fixed and is therefore

consistent all through the training phase. To a certain extent, the vector reflects structural features of UIS for a given task. Meanwhile, C^s as a predetermined unified set can ensure the comparability of different UISs' features, so that the UIS-Feature Embedding Block in Section VI-B can extract higher-level mode information from a large number of UISs' features.

Since k_s reflects the number of tuples to be labelled initially with a limited budget, it corresponds to a small value. As a result, the feature vectors in some fine-grained UIS may be highly sparse. So, we enlarge the vector from set C^s to set C^u, with a heuristic expansion technique.

Specifically, for any bits of the original k_s-bit feature vector are 1, we first retrieve the cluster centers represented by such bits in C^s, then get their l-nearest neighbors from C^u, by using the precomputed $k_s \times k_u$ proximity matrix P^s. l represents the degree of heuristic expansion, which is set to a constant value (e.g., $l = 0.1 \times k_u$ by default). Finally, we redefine a $0/1$ vector of length k_u and set all the bits corresponding to the cluster centers located in C^u to 1. For a meta-task t's UIS R_t^M, this vector is known as the UIS feature vector $v_R \in \mathbb{R}^{k_u}$. Thus, our embedding block f_{θ_R} can be expressed as:

$$emb_R = f_{\theta_R}(v_R), \tag{3}$$

where θ_R represents the parameters of fully connected layers, and emb_R represents the output of the embedding layer.

Data Tuple Embedding Block (f_{θ_τ}). Assuming that a data tuple τ 's representation vector is of size N_r, denoted by $v_\tau \in \mathbb{R}^{N_r}$, this block can be written as:

$$emb_\tau = f_{\theta_\tau}(v_\tau), \tag{4}$$

where θ_τ represents the fully connected layer parameters, and emb_τ represents the output of the embedding layer. For f_{θ_R} and f_{θ_τ}, we set the embedding size to N_e. Thus, emb_R and emb_τ are equally sized, emb_R, $emb_\tau \in \mathbb{R}^{N_e}$.

Classification Block ($f_{\theta_{clf}}$). Given a meta-task t 's UIS feature embedding emb_R, and a list of the corresponding data tuple embeddings emb_τ for $\tau \in \mathcal{S}_t^{sp}$ or $\mathcal{S}_t^{qs}$, we can get the predicted label $\hat{y}_{R,\tau}$ by classification block $f_{\theta_{clf}}$:

$$\hat{y}_{R,\tau} = f_{\theta_{clf}}([emb_R, emb_\tau]), \tag{5}$$

where $[emb_R, emb_\tau]$ is the concatenation of the UIS embedding and the data tuple embedding, and θ_{clf} denotes the parameters of fully connected layers for classification block.

Thus, the parameters θ for $\mathcal{C}_\theta^M$ is $\{\theta_R, \theta_\tau, \theta_{clf}\}$, and the goal of meta-learning is to get the learned initialization parameters $\phi = \{\phi_R, \phi_\tau\ , \phi_{clf}\}$.

B. Memory-Augmented Optimization.

Inspired by [53]–[55], we utilize extra memories (parameters matrices) to store and update some model parameters to overcome the problem that the conventional meta-learning method is easy to slip into the local optimum. Since the basic meta-learning method assigns the same learned initialization parameters (e.g., $\{\phi_R, \phi_\tau, \phi_{clf}\}$) to model parameters (e.g. $\{\theta_R, \theta_\tau, \theta_{clf}\}$) for all tasks during meta-training and the actual use, we hope these learned parameters can be fine-tuned appropriately on different tasks to obtain task-wise parameters. Based on these initialization parameters, we can use labeled tuples to train the classifier more efficiently in the optimization direction of the current task. Therefore, we introduce two types of memories similar to [54], *UIS-feature memory* and *embedding-conversion memory*. The former memory focuses on adjusting the learned initialization parameters of the UIS feature embedding block. The latter memory focuses on the conversion of parameters before inputting $[emb_R, emb_\tau]$ into $f_{\theta_{clf}}$. Noted that the two memories will be updated simultaneously with the meta-learning process.

UIS-Feature Memory. The UIS-feature memory includes the UIS embedding parameters matrix M_R, and the UIS feature vector matrix M_{v_R}. Given a certain task t, the meta-learned initialization parameters of the UIS feature embedding block is ϕ_R. Our goal is to fine-tune ϕ_R to obtain the task-wise initialization parameter θ_R:

$$\theta_R \Leftarrow \phi_R - \sigma\omega_R, \tag{6}$$

where ω_R represents the parameters that need to be adjusted on ϕ_R (i.e., ω_R is a bias term [53], [54]), and $\sigma \in [0, 1]$ is a hyper-parameter that indicates how much ϕ_R needs to be updated. Since we expect ω_R to be associated with a specific task, we adopt the following method to obtain it:

First, we calculate an attention values a_R form M_{v_R} that stores information relevant to a UIS feature vector v_R:

$$a_R = Sim(v_R, M_{v_R}), \tag{7}$$

where $M_{v_R} \in \mathbb{R}^{m \times k_u}$ is a $m \times k_u$ matrix storeing the mode information extracted from the UIS feature vectors of all meta-tasks during the meta-learning training. Here, m is a hyper-parameter, representing the number of implicit modes/patterns [53], [54] we want to extract from the UIS feature vectors of the meta-task set. Sim function calculates the *Cosine similarity* between a UIS feature vector v_R and M_{v_R}, which is normalized by $SoftMax$ function. Thus, we can get $a_R \in \mathbb{R}^m$. Then, the retrieval attention value a_R is applied for extracting parameters ω_R from the memory M_R:

$$\omega_R = a_R^{\mathrm{T}} M_R, \tag{8}$$

where each row of M_R keeps the parameters (fast gradients/bias terms of fully connected layers) of the UIS feature embedding block. Since the UIS embedding block may be comprised of more than one fully connected layer and more than one parameter, $M_R \in \mathbb{R}^{m \times |\theta_R|}$ is not a numerical matrix but stores all the parameters in the same form as the parameters in the UIS embedding block (parameters size is $|\theta_R|$ for both).

The two memory matrices are randomly initialized at the training beginning and will be updated during global update phase of meta-learning.

Embedding-Conversion Memory. The memory aims to obtain task-wise embedding conversion parameters for $[emb_R, emb_\tau] \in \mathbb{R}^{2N_e}$, corresponding to a specific task. We employ an extra parameters matrix $M_{cp} \in \mathbb{R}^{N_e \times 2N_e}$ to store

Algorithm 2 Training process of meta-learning

Input: Meta-task set $\mathcal{T}^M$; UIS feature vector v_R for $t \in \mathcal{T}^M$, Representation vector v_τ and true label $y_{R,\tau}$ for tuple $\tau \in \mathcal{S}_t^{Sp}$, $\mathcal{S}_t^{Qs}$; Hyper-parameters $\eta, \beta, \gamma, \sigma, \rho, \lambda$;
Output: Learned parameters: $\phi_R, \phi_\tau, \phi_{clf}, M_R, M_{v_R}, M_{CP}$;

Random initialize $\phi_R, \phi_\tau, \phi_{clf}, M_R, M_{v_R}, M_{CP}$;
while not reach training epochs **do**
  for $t \in \mathcal{T}^M$ **do**
    Get a_R (Equation 7), Initialize θ_R (Equation 6);
    Initialize $\theta_\tau, \theta_{clf}$ (Equation 11), M_{cp} (Equation 12);
    for $\{\tau, y_{R,\tau}\} \in \mathcal{S}_t^{sp}$ **do**
      Get emb_R, emb_τ (Equation 3,4);
      Get prediction label $\hat{y}_{R,\tau}$ (Equation 9);
      Locally update $\theta_R, \theta_\tau, \theta_{clf}$ (Equation 12);
      Locally update M_{cp} by back-propagation;
  Globally update M_{vR}, M_R, M_{CP} (Equation 14,15,16);
  for $\{\tau, y_{R,\tau}\} \in \mathcal{S}_t^{qs}$ **do**
    Globally update $\phi_R, \phi_\tau, \phi_{clf}$ (Equation 13);
return $\phi_R, \phi_\tau, \phi_{clf}, M_R, M_{v_R}, M_{CP}$;

the conversion parameters. Thus, Equation 5 can be rewritten as :

$$\hat{y}_{R,\tau} = f_{\theta_{clf}}([emb_R, emb_\tau]) = f_{\theta_{clf}}(M_{cp} \cdot [emb_R, emb_\tau]) \tag{9}$$

Similar to the UIS-feature memory, we employ the attention value a_R to retrieve the parameters M_{cp} from the global conversion parameters matrix M_{CP}:

$$M_{cp} = a_R^{\mathrm{T}} \cdot M_{CP}, \tag{10}$$

where $M_{CP} \in \mathbb{R}^{m \times N_e \times 2N_e}$ stores the "equalization" conversion parameters with m implicit modes/patterns, which are extracted from the conversion parameters obtained on all meta-tasks.

During meta-learning training, M_{cp} will be updated in the local update phase together with the updates of parameters of the classifier, and M_{CP} will be updated with M_{cp} in the global update phase.

C. Training strategy

Algorithm 2 depicts the entire meta-learning process. At the beginning of the training, we randomly initialize all global parameters (including the parameters of the classifier and extra memories): $\phi_R, \phi_\tau, \phi_{clf}, M_R, M_{v_R}$, and M_{CP}. After that, according to the sequence of parameter updates, we divide the training process into *local* and *global* phases.

Local Update on the Support Sets. The phase refers to the updating of local parameters: $\theta_R, \theta_\tau, \theta_{clf}, M_{cp}$ on the support set. For each task $t \in \mathcal{T}^M$, we have support set $\mathcal{S}_t^{sp}$. During the local update phase, we first initialize the local classifier parameters : $\{\theta_t, \theta_d, \theta_{clf}, M_{cp}\}$. We use Equation 6 for initialization of θ_R, and Equation 12 for M_{cp}. Since θ_τ and θ_{clf} do not involve memory-augmented optimization, we use the conventional meta-learning initialization method [6]:

$$\theta_\tau; \Leftarrow \phi_\tau; \theta_{clf} \Leftarrow \phi_{clf} \tag{11}$$

The optimization goal for a single task in local training is to minimize the loss of the classification. Thus, the local parameters will be updated as:

$$\theta_* \Leftarrow \theta_* - \rho \cdot \nabla_{\theta_*} LossFunc(y_{R,\tau}, \hat{y}_{R,\tau}) \text{ or } \nabla M_{cp}, \tag{12}$$

where $*$ could be any element in $\{R, \tau, clf\}$; ρ is the learning rate for updating local parameters. It is worth noting that the parameters in M_{cp} are also updated through back-propagation [55].

Global Update on the Query Sets. The phase aims to update global parameters: $\phi_R, \phi_\tau, \phi_{clf}, M_R, M_{v_R}$, and M_{CP}.

According to the "gradient by gradient" setting of meta-learning training [6], we need to perform gradient descent on the locally updated gradient on the support set by minimizing the loss on the query set $\mathcal{S}_t^{qs}$ to update the global parameters ϕ_R, ϕ_τ, and ϕ_{clf}. In order to save the cost of training, after the local update on support sets of all meta-tasks, we update the global parameters by taking one-step gradient descent like [54]. Thus, the global parameters are updated by

$$\phi_* \Leftarrow \phi_* - \lambda \sum_{\mathcal{T}^M} \sum_{\mathcal{S}_t^{qs}} \nabla LossFunc(\hat{\theta_*}), \tag{13}$$

where $*$ could be any element in $\{R, \tau, clf\}$. $\hat{\theta}_*$ is the parameters of $\mathcal{C}_\theta^M$ after training on all support sets, and λ is learning rate.

Meanwhile, M_R, M_{v_R}, and M_{CP} will also be updated as follows. M_{v_R} and M_R will be updated as [54]:

$$M_{v_R} = \eta \cdot (a_R \times v_R^{\mathrm{T}}) + (1 - \eta) M_{v_R}, \tag{14}$$

where $\times$ denotes the cross-product, and η is a hyper-parameter to control how much new UIS feature information is added. We add the attention mask a_R, when adding the new feature information so that the new information will be attentively added to the memory. Similarly, the M_R will be updated by

$$M_R = \beta \cdot (a_R \nabla_{\theta_R} (LossFunc(y_{R,\tau}, \hat{y}_{R,\tau})) + (1 - \beta) M_R), \tag{15}$$

where β is the hyper-parameter to control how much new information is kept. M_{CP} will be updated in the following [55]:

$$M_{CP} = \gamma \cdot (a_R \otimes M_{cp}) + (1 - \gamma) M_{CP}, \tag{16}$$

where $\otimes$ denotes the tensor product, and γ is a hyper-parameter to control how much new information from conversion parameters should be added.

In the online exploration phase, the steps to train the meta-learners by user-labeled tuples are similar to the local update of meta-learning (see the underlined steps in Algorithm 2), except that we directly use the learned global parameters and extra memories.

Discussion. The overhead of the meta-learning is mostly dependent on the size of the meta-task set $|\mathcal{T}^M|$ (see Section VIII-D). Since a well established $\mathcal{T}^M$ needs to traverse as many instances as possible in the meta-subspace, $|\mathcal{T}^M| \propto dim(\mathcal{D}^M)$. In summary, we apply meta-learning to low-dimensional subspaces, which, 1) conforms to the strategy of high-dimensional space decomposition of existing IDEs, and 2) significantly reduces the training overhead.

VII. Preprocessing and Optimization

A. Tabular Data Preprocessing

It refers to the preprocessing for tabular before the meta-learner training, during the offline phase. A straightforward

way is to use the maximum and minimum normalization to process the database (numerical attribute) tuples. However, the method is far from providing feature representations that guarantee the essential performance of NN classifiers. Moreover, training NN classifiers with simple normalization in a low-dimensional data space may cause gradient saturation [56], when training with few-labeled tuples.

In our implementation, the tabular representation is based on multi-modal attribute features, extracted by Gaussian mixture model (GMM) [57] and Jenks natural breaks classification (JKC) [58], [59]. The representation vector of an attribute value is the concatenation of two parts. The first part is a one-hot vector, indicating which GMM component or JKC interval the value belongs to. The second part is a value of range $[0, 1]$, which is the normalized value on its corresponding GMM component or JKC interval.

Thus, given a database attribute a_j, and a set of tuples of a_j sampled from the database T_j^s, the tabular representation transforms τ_j to v_{τ_j} (Algorithm 3). A tuple contains multiple attributes, and therefore the vector representation of the tuple is obtained by concatenating the vector representations on all attributes. Next, we discuss GMM and JKC models.

Gaussian Mixture Model. We can capture the feature of the attribute values by training the GMM composed of a series of Gaussian distribution components. Suppose a batch of sample data for a certain attribute, we may get the set of Gaussian distribution components $\{g_i\}_{i\leq|g|}$, in accordance to the specified number of components, $|g|$, for the GMM training. The mean and variance of each component g_i are represented by μ_i and θ_i, respectively. Then, given an attribute value τ_j, we can compute the probability distribution $\{p_1^j, .., p_{|g|}^j\}$ that the value belongs to the components, and choose the one maximizing the likelihood, as the GMM component corresponding to τ_j.

Jenks Natural Breaks Classification. JKC is a data clustering method designed to determine values' best arrangement to different classes, called Jenks Natural Breaks intervals, or JKC intervals. JKC divides the distribution of a numerical attribute into approximately smooth JKC intervals, $\{b_i\}_{i\leq|b|}$, by minimizing the variance within an interval, and maximizing the variance between different intervals. We can also specify the number of JKC intervals, $|b|$. Then, given an attribute value τ_j, we can quickly determine which JKC interval it belongs to, by comparing with boundary values of different JKC intervals.

Notice that GMM is suitable for processing numerical attributes with distribution composed of one or more peaks, i.e., unimodal and multimodal distributions, according to [60]. Also, we find that there are a large number of numerical attributes with distributions composed of smooth intervals, like trends or time series, which are more suitable for being processed by JKC or other interval scanning techniques [61]. In addition, Using GMM or JKC for the entire exploratory database can be costly, though it is effective. A practical solution is to make tabular representation on sampled data, so that the scalability can be ensured. In our work, we use random sampling and limit the sampling ratio under 1%.

Algorithm 3 Tabular data preprocessing

Input: T_j^s, a sampled tuple set on attribute a_j; τ_j, a tuple to be represented on a_j; $|g|$, # of GMM components; $|b|$, # of JKC intervals;
Output: v_{τ_j}, a represented vector for τ_j

1: $\{g_i\}_{i\leq|g|} \leftarrow$ ***GMM***$(T_j^s, |g|)$ or $\{b_i\}_{i\leq|b|} \leftarrow$ ***JKC***$(T_j^s, |b|)$;
2: **if** using GMM **then**
3: Compute $\{p_1^j, .., p_{|g|}^j\}$ for τ_j and $k = argmax_\kappa\ p_\kappa^j$;
4: $label_j \leftarrow$***ont-hot***$(g_k, \{g_1, .., g_{|g|}\})$; $Norm_j \leftarrow \frac{\tau_j - \mu_k}{2\theta_k}$;
5: **if** using JKC **then**
6: Find the JKC interval b_k corresponding to τ_j;
7: $label_j \leftarrow$***one-hot***$(b_k, \{b_1, .., b_{|b|}\})$; $Norm_j \leftarrow \frac{\tau_j - b_k.\mathbf{min}}{b_k.\mathbf{max} - b_k.\mathbf{min}}$;
8: **return** $v_{\tau_j} \leftarrow label_j \oplus [Norm_j]$;

B. Optimization for Few-shot Prediction.

The purpose of a classifier is to predict a UIS during online exploration. The quality of prediction can be measured by "false positive" (FP) and "false negative" (FN). FPs refer to errors of falsely predicting tuples that are "not interest" as "interest" to a user. FNs refer to errors of falsely predicting tuples that are "interest" as "not interest" to a user. Given a well-trained classifier, the quality is much dependent on the "few shots" during the online training. we study optimizations for quality refinement. Note that since we study UIS of arbitrary shapes, the optimization method is heuristic and preliminary. It thus creates a vacuum for further optimization over specified UISs based on our approach.

For FP Errors. We study an inevitable source of FP errors which are common for few-shot learning. An uninteresting tuples far away from user labeled tuples could be randomly predicted by a classifier as "interesting", because of lacking of sufficient information. Thus, we hope to get a superset of UIS to fix such errors. First, we use positively labelled tuples by users as "anchor points". Then, we retrieve the proximate tuples of anchor points to build a set of large-scale circumscribed regions. The combination of large-scale regions (called "outer-subregion") is conceived to cover the real UIS. For the tuples located within the outer-subregion, we follow the prediction of classifiers, so that there is no chance of a negative tuple being recognized as positive, and vice versa. Also, classifiers revise a tuple from positive to negative, if the tuple is not covered by the outer-subregion, i.e., beyond UIS.

In our implementation, the approach of searching neighboring tuples is similar to expanding the UIS feature vector in Section VI-A. After tuples are labelled in the initial exploration phase, for each cluster center that is identified as "interesting" in C^s, we search for N_{sup} more proximate cluster centers from other cluster center set (e.g., C^u) by proximity matrix P^s(see Section V-B). Parameter N_{sup} reflects the extent of expansion, in our implementation, we set it to k_u or k_q multiplied by a certain scale factor. Then, we construct a circumscribed region, e.g., convex hull, on these cluster centers. Finally, all convex hulls are combined as the outer-subregion.

For FN Errors. Similarly, an inevitable source of FN errors originates in the randomness of classifier prediction under few-labeled tuples. A type of FN errors appear as some small "false" regions within the real UIS, which are predicted as "not interesting" by classifiers. As a result, we build a small-scale

circumscribed region set (called *inner-subregion*) from the positive tuples in the initial tuple set, which can be considered a subset of the UIS. We can infer that the tuples in the inner-subregion should be positive, since they are located within the UIS. Also, classifiers revise a tuple to positive, if it is falsely recognized as negative, i.e., located within an inner region. The approach of constructing the inner-subregion is the same as that of the outer-subregion, except that the expansion step size N_{sub} is significantly less than N_{sup}, which we term as conservative expansion.

VIII. Results

A. Setup

Datasets. We use 2 public datasets, SDSS [9] and CAR [10], which are commonly used in previous works. SDSS is a scientific dataset of sky objects [62]. We use 100K tuples of 8 attributes follow the settings of [5]. CAR has 50K tuples for second-hand car information in eBay. We select 5 commonly used attributes out of 19 attributes based on the guidance in [62]. For each dataset, data space is randomly split into a set of 2D subspaces, in consistency with settings of baselines for fair comparison.

Baselines. There are two state-of-the-arts for UIR/UIS classification, DSM [5] and AL-SVM [4]. The settings of DSM and AL-SVM follows the settings in [5] and [4], respectively. AL-SVM uses active learning to select training tuples for SVM. DSM improves AL-SVM by incorporating the polytope-based optimization. To fully examine the performance of LTE, we list several variants, *Basic*, *Meta*, and *Meta**. Basic uses basic UIS Classifiers without any optimization. Meta improves Basic with meta-learning. Meta* adopts all optimizations proposed (i.e., using the optimizer module based on Meta), Note that the effect of the optimizer is completely dependent on the underlying results of the meta-learner, and the optimizer cannot be used alone.

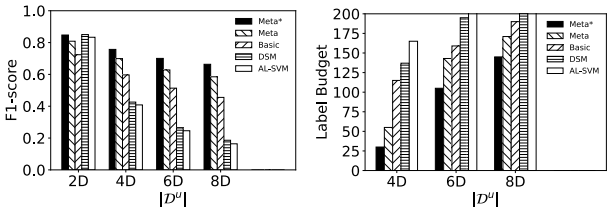
Metrics. Accuracy and efficiency are two common metrics for IDE evaluation. Accuracy is measured by F1-score = $\frac{2\cdot\text{precision}\cdot\text{recall}}{\text{precision}+\text{recall}}$. Efficiency refers to cost efficiency, which is constrained by budget B,i.e., the number of labelled tuples needed.

***Parameters*[11].** *Meta-task generation*: we set the $k_u = 100$ and $k_q = 200$ in each meta-subspace. Since the size of support sets ($k_s + \Delta$) reflects the exploration budget B for labelling, we trained the corresponding meta-learners under $B = \{\mathbf{30}, 40, 50, 100\}$. We set $\alpha = 1$ and $\psi = 50$ for generating meta-tasks following the setting of baselines in Section VIII-B. We set $\alpha = 4$ and $\psi = 20$ for generating meta-tasks to study the adaptability of our method to various types of complex UISs in Sections VIII-C. We also test the performance by varying the size of the meta-task set $|\mathcal{T}^M|$ as $\{1000, 5000, 10000, \mathbf{15000}\}$. *Meta-learning training*: for the meta-learner, we set the embedding size $N_e = 100$ and use the $Relu$ activation function between all layers. By searching for meta-learning training hyper-parameters [54], we search $\eta, \beta, \gamma, \sigma, \rho, \lambda$ in $\{0.01,\ 0.001,\ 0.0001,\ \mathbf{0.00005}\}$ and m in $\{2, 4, \mathbf{6}\}$. The number of training epochs is in $\{1, 2, 3, \mathbf{4}\}$. The training batch size is in $\{5, 10, \mathbf{15}\}$ and the training step size is in $\{5, 10, 20, \mathbf{30}\}$ for the local update phase. *Optimizer*: we set N_{sup} as $\{20\%, \mathbf{30}\%, 40\%\}$ of k_u and N_{sub} as $\{\mathbf{5}\%, 10\%, 15\%\}$ of k_u.

[9] https://www.sdss.org/dr17/

[10] https://data.world/data-society/used-CARs-data

[11] The default parameters are bolded.



(a) Accuracy w.r.t Dimension (b) Efficiency w.r.t Dimension

Fig. 4. Learn-to-explore vs. Baselines (SDSS)

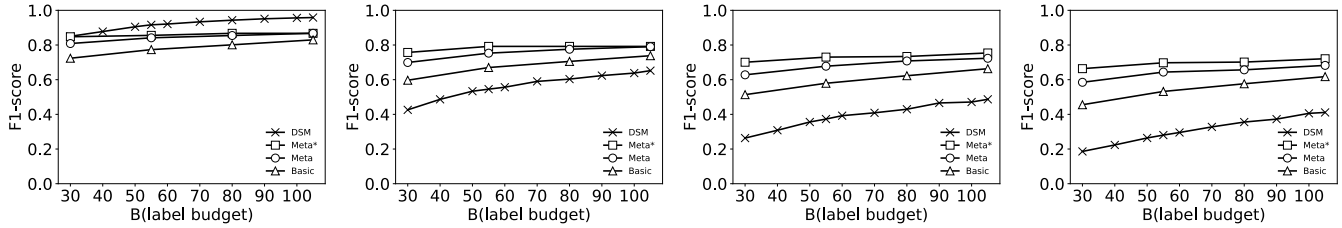
B. Comparison with Baselines

For comparison with baselines, we follow DSM and AL-SVM's assumption on subspatial convexity and conjunctivity. Therefore, for a testing UIR in the high dimensional space, we construct convex UIS in low-dimensional subspaces by *convex hull* model (i.e., fixing $\alpha = 1$ and vary $\phi \in \{20, 15, 10, 5\}$) and use the conjunctive property to unite these UISs to get final UIR. We totally generate 2,500 UIRs from 2D to 8D for testing, and the result is in Figure 4. In addition, the result of our proposal without UIR assumptions is to be shown in Section VIII-C. Notice that the cost of initial sampling [63] for baselines is not counted in their statistics, in all testings.

Figure 4(a) examines the effect of dimensionality over the accuracy, by fixing B to 30. It shows that the accuracy decreases w.r.t. the increase of dimensionality, for all competitors. It is because that the selection of representative tuples becomes more difficult, due to the sparse data distribution of a higher dimensional space. Compared to the sharp drop of SVM-based methods, DSM and AL-SVM, the performance of NN-based methods are much more stable. In particular, the F1-score of DSM decreases about 75% when the number of dimensions changes from 2 to 8. In comparison, the drop rate of all NN-based classifiers are steadily below 40%. Among them, the number of Meta* only drops about 18%, showing good scalability with dimensions.

Figure 4(b) examines the effect of dimensionality over efficiency, by fixing F1-score to 0.75. It shows that Meta* can achieve a given F1-score with a budget of less than 150 labeled tuples on 4-8D. However, DSM and AL-SVM require more than 150 labeled tuples in 6-8D (especially in 8D, far exceeding 150), which can be tedious for users. Since DSM outperforms AL-SVM in all testings, we only show the result of DSM in following experiments.

We test the effect of exploration budgets B over the accuracy in Figure 5 (a-d). It shows that all methods' accuracy increases, if the given budget increases. DSM performs better for 2D space, because experiments are done following the



(a) Results on 2D (b) Results on 4D (c) Results on 6D (d) Results on 8D

Fig. 5. Accuracy w.r.t. B (SDSS, 4-8D)

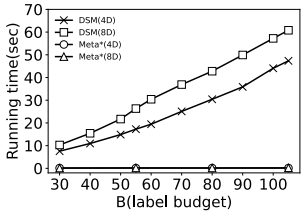
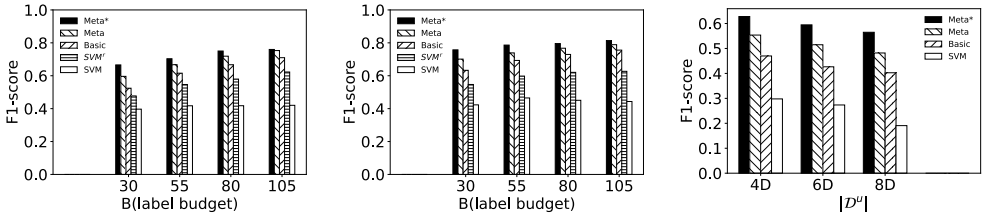
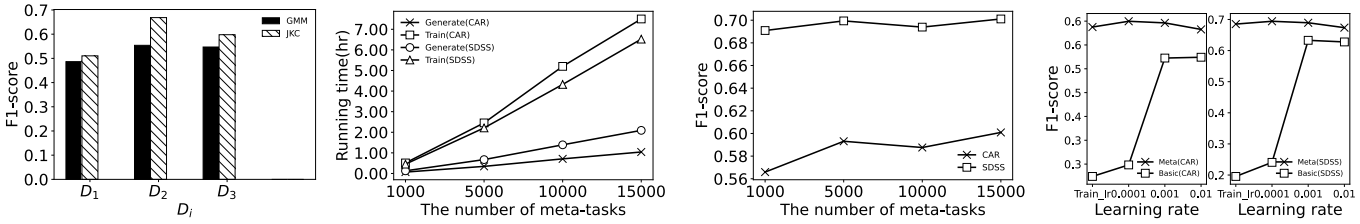


Fig. 6. Efficiency in Online Exploration



(a) Accuracy w.r.t. B (CAR) (b) Accuracy w.r.t. B (SDSS) (c) Accuracy w.r.t. UIR Dim (SDSS)

Fig. 7. Performance on Generalized UIRs



(a) GMM vs. JKC (b) Pretraining Cost w.r.t. $|\mathcal{T}^M|$ (c) Accuracy w.r.t. $|\mathcal{T}^M|$ (d) Accuracy w.r.t Learning Rate (Online exploration)

Fig. 8. Analysis

TABLE II
ACCURACY W.R.T. UIS MODES (B=30)

		M1	M2	M3	M4	M5	M6	M7
CAR	**Meta***	0.839	0.723	0.544	0.307	0.71	0.749	0.786
	Meta	0.795	0.667	0.486	0.266	0.606	0.673	0.731
	Basic	0.737	0.612	0.421	0.231	0.462	0.568	0.652
	SVM^r	0.712	0.562	0.331	0.127	0.450	0.531	0.624
	SVM	0.683	0.487	0.206	0.017	0.316	0.468	0.598
SDSS	**Meta***	0.866	0.813	0.704	0.459	0.804	0.813	0.838
	Meta	0.812	0.744	0.605	0.380	0.758	0.771	0.782
	Basic	0.761	0.680	0.547	0.339	0.696	0.697	0.717
	SVM^r	0.758	0.645	0.373	0.156	0.573	0.628	0.692
	SVM	0.747	0.556	0.164	0.023	0.319	0.502	0.647

TABLE III
MODES OF TEST BENCHMARKS

Mode	M1	M2	M3	M4	M5	M6	M7
α	4	4	4	4	1	2	3
ψ	20	15	10	5	20	20	20

convexity and conjunctivity assumptions, which best fit its polytope-based optimization [5]. However, the performance of DSM degrades fast as the increase of dimensionality, which is consistent with the observation in Figure 4. For example, in 8-dimensional space, the F1-score of DSM is below 20%, when $B = 30$, and is below 41% when $B = 105$. Compared to that, our methods, Meta and Meta*, scale well w.r.t. the number of dimensions. The accuracy of Meta and Meta* dominates that of DSM. For example, when $|\mathcal{D}^u| = 8$ and $B = 30$, the F1-score of Meta* is 267% of that of DSM.

We test the efficiency by collecting the runtime cost during the online exploration phase, for all competitors, in Figure 6. It shows that the online training time of DSM increases almost linearly w.r.t. the given budget. Also, if the number of dimensions increases, the training time takes longer. For example, DSM takes about 50 and 60 seconds when B equals 105, on 4- and 8-dimensional spaces, respectively. Compared to that, Meta*'s online exploration cost is two orders of magnitude lower because we save much cost by avoiding the online active learning process. When the number of dimensions increases from 4 to 8, the online exploration cost only increases from 0.127s to 0.130s. It implies that our method has more potentials to provide the data exploration as a service, for a large number of users to access simultaneously.

C. Performance on Generalized UIRs

Our proposal supports UIRs, generalized from convex UIS to concave or even disconnected UIS in subspaces. We compare our proposals with SVM classifiers, since DSM degenerates into SVM classifiers, if UIS is not convex [5]. We consider a variant *SVM^r* referring to using tabular data preprocessing in addition to SVM. All competitors are fed with the same set of initial training tuples for fair comparison.

The performance is tested on different UIS modes, which are randomly generated following the way of meta-task generation, as specified by two hyper-parameters, α and ψ. To

test the results on the combination of the two parameters, we first fix α to 4 and vary $\psi \in \{20, 15, 10, 5\}$, then fix ψ to 20 and vary $\alpha \in \{1, 2, 3\}$, so that we get 7 UIS modes (M1-M7), as shown in Table III. For each mode, we generate 100 UISs for each subspace. According to the statistics of generated UISs, 92% of UISs are concave, 55% of them consist of separated regions. Note that the meta-tasks used to train the meta-learners is only generated under $\alpha = 4$, and $\psi = 20$, and all subsequent experiments follow this setting.

Table II shows the performance of each mode under labelling budget $B = 30$. It shows that NN-based methods outperform SVM-based variants. Meta-learning based methods, Meta and Meta*, further improve basic in all testing modes. For example, for M4, the F1-score of Meta* is 164% of that of SVM on CAR, showing the superiority of our method. Also, SVM^r is better than SVM, due to the effectiveness of tabular data preprocessing. We then test the effect of meta-learning by comparing Meta and Basic. It shows that, from M5-M7 (CAR), the improvement of Meta over Basic is about 31%, 18%, and 12%, when α is set to 1, 2, and 3. We also find that the improvement of Meta and Meta* over Basic is more significant, when α is small. Intuitively, a smaller α corresponds to a simpler task, thus the predicability can be higher. Compared to that, the trend over ψ is relatively stable. Similar results are observed on SDSS. The above results show that the meta-learners trained under larger α and ψ also perform well on UIS configured by small α and ψ, so that we recommend larger valued α and ψ for meta-task generation.

We also test the performance by varying budget B from 30 to 100 in Figures 7(a) and 7(b), for CAR and SDSS, respectively. It shows that if the given budget increases, the accuracies of all methods except SVM increase. The reason is that when SVM handles a complex UIS, it is difficult to determine the appropriate hyper-parameters and kernel functions. In addition, our methods, Meta* and Meta, better predict complex UIS under a small B with meta-knowledge. For example, on CAR, Meta with $B = 55$ achieves the same performance as Basic with $B = 80$.

Then, we show the performance w.r.t. dimensions of UIR, being generated by combining UISs from low-dimensional subspaces. UISs are generated according to Table III. Figure 7(c) examines the effect of accuracy over dimensions, with $B = 30$. It shows that our method achieves relatively stable performance in different dimensions when UIR is complex.

D. Analysis

GMM vs. JKC. We study the effectiveness of tabular representations, JKC and GMM, as multi-mode feature models, in Figure 8(a). If with GMM, the F1-score can be as high as 0.55 for 2D case. Basic integrates JKC and GMM representations, whose performance can be further improved (e.g, F1-score= 0.67 for 2D case). Without JKC and GMM, the model can hardly be trained and used, with a F1-score even much lower than baselines.

Pre-training Cost. We investigate the performance of runtime efficiency and accuracy w.r.t. the number of meta-tasks in Figures 8(b) and 8(c). The runtime cost refers to two parts, the generation time for meta-tasks, and training time. Both of the two parts are linearly proportional to the number of meta-tasks, as shown in Figure 8(b). Meanwhile, we find that the runtime cost does not depend on the dataset size. For example, CAR takes only half of the data size of SDSS, but the training time is only 12% less. We also test the accuracy w.r.t. the number of meta-tasks in Figure 8(c). It shows that for both datasets, the accuracy is not sensitive to the number of meta-tasks, except the number of tasks is low, e.g., $|\mathcal{T}^M| = 1,000$. There exist some fluctuations in Figure 8(c), which are consistent with the consensus [7] that the meta-learning performance w.r.t. the number of meta-tasks follows a gradual transition from positive correlation to fluctuation with stationarity. So, we can do an early stop for meta-training by finding a "sweet point" of accuracy and efficiency. According to Figures 8(b) and 8(c), when $|\mathcal{T}^M| = 5,000$, the accuracy is almost at the peak, while the training runtime cost is low.

The Effect of Meta-Learning. We study the effect of meta-learning by comparing Basic and Meta, under different values of learning rates. We only compare the two to show the effectiveness of meta-learning, by eliminating the influence of other factors. The learning rate refers to the step size at each iteration, while moving towards the minimum of a loss function during optimization. For offline training, a small learning rate (0.00005) is chosen to conservatively and deliberately capture the meta-knowledge. For online exploration, a large learning rate is preferred for fast converging to UIR. The result is shown in Figure 8(d), where Meta steadily outperforms Basic. The improvement is achieved because Meta is equipped with meta-knowledge, in form of good initial parameters, so the sensitiveness to learning rates is low and the performance is stable. So, Meta can achieve good results at a small learning rate. For example, when learning rate is 0.0001, the F1-score of Meta is 0.7, but the F1-score of Basic is only 0.25, which is 64% lower, on SDSS. Similar results are observed on CAR.

IX. Conclusion

In this paper, we study the problem of interactive data exploration by proposing a "learn-to-explore" framework. The framework leverages meta-learning based neural network classifiers, which are pre-trained by automatically generated meta-tasks in an unsupervised manner, and are fast adapted to optimal parameters during the online exploration. The framework can be plugged to existing IDE systems by providing good initial parameters for classifiers, thus yielding good accuracy and efficiency. To implement such a framework, we study a set of techniques, including meta-task generation, meta-training, etc. Experiments on real datasets show that our proposal outperforms existing solutions in terms of accuracy and efficiency.

X. Acknowledgements

This work is supported by NSFC (No.61772492, 62072428), the CAS Pioneer Hundred Talents Program. Xike Xie is the corresponding author.

REFERENCES

[1] S. Idreos, O. Papaemmanouil, and S. Chaudhuri, "Overview of data exploration techniques," in *SIGMOD*, 2015, pp. 277–281.

[2] K. Dimitriadou, O. Papaemmanouil, and Y. Diao, "Explore-by-example: an automatic query steering framework for interactive data exploration," in *SIGMOD*, 2014, pp. 517–528.

[3] T. Milo and A. Somech, "Automating exploratory data analysis via machine learning: An overview," in *SIGMOD*, 2020, pp. 2617–2622.

[4] K. Dimitriadou, O. Papaemmanouil, and Y. Diao, "AIDE: an active learning-based approach for interactive data exploration," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 11, pp. 2842–2856, 2016.

[5] E. Huang, L. Peng, L. D. Palma, A. Abdelkafi, A. Liu, and Y. Diao, "Optimization for active learning-based interactive database exploration," *Proc. VLDB Endow.*, vol. 12, no. 1, pp. 71–84, 2018.

[6] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *ICML*, 2017, pp. 1126–1135.

[7] T. M. Hospedales, A. Antoniou, P. Micaelli, and A. J. Storkey, "Meta-learning in neural networks: A survey," *CoRR*, vol. abs/2004.05439, 2020.

[8] K. Hsu, S. Levine, and C. Finn, "Unsupervised learning via meta-learning," *CoRR*, vol. abs/1810.02334, 2018.

[9] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas, "New trends on exploratory methods for data analytics," *Proc. VLDB Endow.*, vol. 10, no. 12, pp. 1977–1980, 2017.

[10] S. B. Roy, H. Wang, U. Nambiar, G. Das, and M. K. Mohania, "Dynacet: Building dynamic faceted search systems over databases," in *ICDE*, 2009, pp. 1463–1466.

[11] N. Kamat, P. Jayachandran, K. Tunga, and A. Nandi, "Distributed and interactive cube exploration," in *ICDE*, 2014, pp. 472–483.

[12] S. Agarwal, A. Panda, B. Mozafari, A. P. Iyer, S. Madden, and I. Stoica, "Blink and it's done: Interactive queries on very large data," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1902–1905, 2012.

[13] M. L. Kersten, S. Idreos, S. Manegold, and E. Liarou, "The researcher's guide to the data deluge: Querying a scientific database in just a few seconds," *Proc. VLDB Endow.*, vol. 4, no. 12, pp. 1474–1477, 2011.

[14] A. Kalinin, U. Çetintemel, and S. B. Zdonik, "Interactive data exploration using semantic windows," in *SIGMOD*, 2014, pp. 505–516.

[15] A. Wasay, X. Wei, N. Dayan, and S. Idreos, "Data canopy: Accelerating exploratory statistical analysis," in *SIGMOD*, 2017, pp. 557–572.

[16] X. Xie, K. Zou, X. Hao, T. B. Pedersen, P. Jin, and W. Yang, "OLAP over probabilistic data cubes II: parallel materialization and extended aggregates," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 10, pp. 1966–1981, 2020.

[17] R. Gao, X. Xie, K. Zou, and T. B. Pedersen, "Multi-dimensional probabilistic regression over imprecise data streams," in *WWW*, 2022, pp. 3317–3326.

[18] X. Xie, X. Hao, T. B. Pedersen, P. Jin, and J. Chen, "OLAP over probabilistic data cubes I: aggregating, materializing, and querying," in *ICDE*, 2016, pp. 799–810.

[19] S. Sarawagi, R. Agrawal, and N. Megiddo, "Discovery-driven exploration of olap data cubes," in *EDBT*, 1998, pp. 168–182.

[20] A. Key, B. Howe, D. Perry, and C. R. Aragon, "Vizdeck: self-organizing dashboards for visual analytics," in *SIGMOD*, 2012, pp. 681–684.

[21] A. Parameswaran, N. Polyzotis, and H. Garcia-Molina, "Seedb: Visualizing database queries efficiently," *Proc. VLDB Endow.*, 2013.

[22] A. Cheung, A. Solar-Lezama, and S. Madden, "Using program synthesis for social recommendations," in *CIKM*, 2012, pp. 1732–1736.

[23] A. Abouzied, J. M. Hellerstein, and A. Silberschatz, "Playful query specification with dataplay," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1938–1941, 2012.

[24] B. Tang, S. Han, M. L. Yiu, R. Ding, and D. Zhang, "Extracting top-k insights from multi-dimensional data," in *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*. ACM, 2017, pp. 1509–1524.

[25] R. Ding, S. Han, Y. Xu, H. Zhang, and D. Zhang, "Quickinsights: Quick and automatic discovery of insights from multi-dimensional data," in *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. ACM, 2019, pp. 317–332.

[26] P. Ma, R. Ding, S. Han, and D. Zhang, "Metainsight: Automatic discovery of structured knowledge for exploratory data analysis," in *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. ACM, 2021, pp. 1262–1274.

[27] O. B. El, T. Milo, and A. Somech, "Automatically generating data exploration sessions using deep reinforcement learning," in *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 2020, pp. 1527–1537.

[28] A. Personnaz, S. Amer-Yahia, L. Berti-Équille, M. Fabricius, and S. Subramanian, "DORA THE EXPLORER: exploring very large data with interactive deep reinforcement learning," in *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*. ACM, 2021, pp. 4769–4773.

[29] D. Deutch, A. Gilad, T. Milo, and A. Somech, "Explained: Explanations for EDA notebooks," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 2917–2920, 2020.

[30] Y. Diao, P. Guzewicz, I. Manolescu, and M. Mazuran, "Efficient exploration of interesting aggregates in RDF graphs," in *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. ACM, 2021, pp. 392–404.

[31] X. Zhu, X. Huang, J. Huang, B. Choi, and J. Xu, "Hdag-explorer: A system for hierarchical DAG summarization and exploration," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 2973–2976, 2020.

[32] B. Li, R. Cheng, J. Hu, Y. Fang, M. Ou, R. Luo, K. C. Chang, and X. Lin, "Mc-explorer: Analyzing and visualizing motif-cliques on large networks," in *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*. IEEE, 2020, pp. 1722–1725.

[33] Y. Luo, W. Li, T. Zhao, X. Yu, L. Zhang, G. Li, and N. Tang, "Deeptrack: Monitoring and exploring spatio-temporal data - A case of tracking COVID-19 -," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 2841–2844, 2020.

[34] J. Yu, K. Chowdhury, and M. Sarwat, "Tabula in action: A sampling middleware for interactive geospatial visualization dashboards," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 2925–2928, 2020.

[35] T. Guo, K. Feng, G. Cong, and Z. Bao, "Efficient selection of geospatial data on maps for interactive and visualized exploration," in *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*. ACM, 2018, pp. 567–582.

[36] R. Neamtu, R. Ahsan, E. A. Rundensteiner, and G. N. Sárközy, "Interactive time series exploration powered by the marriage of similarity distances," *Proc. VLDB Endow.*, vol. 10, no. 3, pp. 169–180, 2016.

[37] P. Eichmann, F. Solleza, N. Tatbul, and S. Zdonik, "Visual exploration of time series anomalies with metro-viz," in *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. ACM, 2019, pp. 1901–1904.

[38] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. G. Parameswaran, "Effortless data exploration with zenvisage: An expressive and interactive visual analytics system," *Proc. VLDB Endow.*, vol. 10, no. 4, pp. 457–468, 2016.

[39] Y. Luo, X. Qin, C. Chai, N. Tang, G. Li, and W. Li, "Steerable self-driving data visualization," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 1, pp. 475–490, 2022.

[40] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas, "Exemplar queries: Give me an example of what you need," *Proc. VLDB Endow.*, vol. 7, no. 5, pp. 365–376, 2014.

[41] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," *ACM Comput. Surv.*, vol. 53, no. 3, pp. 63:1–63:34, 2020.

[42] L. Palma, Y. Diao, and A. Liu, "Efficient version space algorithms for" human-in-the-loop" model development," Ecole Polytechnique, Tech. Rep., 2020.

[43] P. Ren, Y. Xiao, X. Chang, P. Huang, Z. Li, B. B. Gupta, X. Chen, and X. Wang, "A survey of deep active learning," *ACM Comput. Surv.*, vol. 54, no. 9, pp. 180:1–180:40, 2022.

[44] E. Wu, L. Battle, and S. R. Madden, "The case for data visualization management systems: Vision paper," *Proc. VLDB Endow.*, 2014.

[45] A. Kim, E. Blais, A. G. Parameswaran, P. Indyk, S. Madden, and R. Rubinfeld, "Rapid sampling for visualizations with ordering guarantees," *Proc. VLDB Endow.*, vol. 8, no. 5, pp. 521–532, 2015.

[46] A. Abouzied, D. Angluin, C. H. Papadimitriou, J. M. Hellerstein, and A. Silberschatz, "Learning and verifying quantified boolean queries by example," in *PODS*, 2013, pp. 49–60.

[47] Q. T. Tran, C. Chan, and S. Parthasarathy, "Query by output," in *SIGMOD*, 2009, pp. 535–548.

[48] W. Loh, "Classification and regression trees," *WIREs Data Mining Knowl. Discov.*, vol. 1, no. 1, pp. 14–23, 2011.

[49] H. Liu, W. Liu, and L. J. Latecki, "Convex shape decomposition," in *CVPR*, 2010, pp. 97–104.

[50] J. Lien and N. M. Amato, "Approximate convex decomposition of polygons," in *SCG*, 2004, pp. 17–26.

[51] A. Bouchachia, "Dynamic clustering," *Evolving Systems*, vol. 3, no. 3, pp. 133–134, 2012.

[52] A. L. Mary and K. S. Kumar, "A density based dynamic data clustering algorithm based on incremental dataset," *Citeseer*, 2012.

[53] H. Lee, J. Im, S. Jang, H. Cho, and S. Chung, "Melu: Meta-learned user preference estimator for cold-start recommendation," in *KDD*, 2019, pp. 1073–1082.

[54] M. Dong, F. Yuan, L. Yao, X. Xu, and L. Zhu, "MAMO: memory-augmented meta-optimization for cold-start recommendation," in *KDD*, 2020, pp. 688–697.

[55] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *CoRR*, vol. abs/1410.5401, 2014.

[56] J. Fan, T. Liu, G. Li, J. Chen, Y. Shen, and X. Du, "Relational data synthesis using generative adversarial networks: A design space exploration," *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 1962–1975, 2020.

[57] D. A. Reynolds, "Gaussian mixture models," *Encyclopedia of biometrics*, vol. 741, pp. 659–663, 2009.

[58] G. F. Jenks and F. C. Caspall, "Error on choroplethic maps: definition, measurement, reduction," *Annals of the Association of American Geographers*, vol. 61, no. 2, pp. 217–244, 1971.

[59] G. F. Jenks, "Optimal data classification for choropleth maps," *Department of Geographiy, University of Kansas Occasional Paper*, 1977.

[60] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional GAN," in *NeurIPS*, 2019, pp. 7333–7343.

[61] Q. Xie, C. Pang, X. Zhou, X. Zhang, and K. Deng, "Maximum error-bounded piecewise linear representation for online stream approximation," *VLDB J.*, vol. 23, no. 6, pp. 915–937, 2014.

[62] X. Qin, C. Chai, Y. Luo, N. Tang, and G. Li, "Interactively discovering and ranking desired tuples without writing SQL queries," in *SIGMOD*, 2020, pp. 2745–2748.

[63] W. Liu, Y. Diao, and A. Liu, "An analysis of query-agnostic sampling for interactive data exploration," *Communications in Statistics-Theory and Methods*, vol. 47, no. 16, pp. 3820–3837, 2018.