The VLDB Journal (2022) 31:753–777
https://doi.org/10.1007/s00778-021-00714-0

REGULAR PAPER

Interactively discovering and ranking desired tuples by data exploration

Xuedi Qin[1] · Chengliang Chai[1] · Yuyu Luo[1] · Tianyu Zhao[1] · Nan Tang[2] · Guoliang Li[1] · Jianhua Feng[1] · Xiang Yu[1] · Mourad Ouzzani[2]

Received: 26 February 2021 / Revised: 19 October 2021 / Accepted: 26 October 2021 / Published online: 18 January 2022

Abstract
Data exploration—the problem of extracting knowledge from database even if we do not know exactly what we are looking for —is important for data discovery and analysis. However, precisely specifying SQL queries is not always practical, such as "finding and ranking off-road cars based on a combination of Price, Make, Model, Age, Mileage, etc"—not only due to the query complexity (e.g.,the queries may have many *if-then-else*, *and*, *or* and *not* logic), but also because the user typically does not have the knowledge of all data instances (and their variants). We propose DExPlorer, a system for interactive data exploration. From the user perspective, we propose a simple and user-friendly interface, which allows to: (1) confirm whether a tuple is desired or not, and (2) decide whether a tuple is more preferred than another. Behind the scenes, we jointly use multiple ML models to learn from the above two types of user feedback. Moreover, in order to effectively involve *human-in-the-loop*, we need to select a set of tuples for each user interaction so as to solicit feedback. Therefore, we devise *question selection* algorithms, which consider not only the estimated benefit of each tuple, but also the possible partial orders between any two suggested tuples. Experiments on real-world datasets show that DExPlorer outperforms existing approaches in effectiveness.

Keywords Data exploration · SQL query · Ranking · Decision · Human-in-the-loop

✉ Chengliang Chai
chaicl15@mails.tsinghua.edu.cn

✉ Guoliang Li
liguoliang@tsinghua.edu.cn

Xuedi Qin
qxd17@mails.tsinghua.edu.cn

Yuyu Luo
luoyy18@mails.tsinghua.edu.cn

Tianyu Zhao
zhaoty17@mails.tsinghua.edu.cn

Nan Tang
ntang@hbku.edu.qa

Jianhua Feng
fengjh@tsinghua.edu.cn

Xiang Yu
x-yu17@mails.tsinghua.edu.cn

Mourad Ouzzani
mouzzani@hbku.edu.qa

[1] Department of Computer Science and Technology, Tsinghua University, Beijing, China

[2] Qatar Computing Research Institute, HBKU, Doha, Qatar

1 Introduction

We study the problem of *interactive data exploration*, for the scenarios that a user needs to find desired and ranked tuples, but the query intent is hard to precisely specify. This is common in practice, for both non-experts who cannot write SQL queries and scripts, and experts who are not familiar with the data. Adding to the complexity is that the query intent may be complicated, e.g.,finding tuples that satisfy a combination of many *if-then-else*, *and*, *or* and *not* conditions, and are ranked by a weighted function over multiple attributes.

Let *DE* denote the general problem that discovers and ranks tuples, *DE-Decision* a special case of DE where users only want desired tuples (without ranking), and *DE-Ranking* another special case where users want the ranking over all tuples. Existing works either only focus on finding desired tuples [14,17,28,53,68,70,75], i.e.,the *DE-Decision* problem, or only aim to rank targeted tuples [9,56,72], i.e.,the *DE-Ranking* problem. Therefore, we propose DExPlorer to interactively discover and rank tuples by data exploration. The main difference between this work and those existing works is how to *holistically* solve the general *DE* problem,

Table 1 Comparison with the state-of-the-art. User Input—(K): Keywords, (E_H): Examples Offered by the Human, (E_S): Examples Offered by the System, (P): Partial Orders, (P_R): Predicates

Project	Input	DE-Decision	DE-Ranking	DE	Supported special cases	QRE	ML
DISCOVER [27]	(K)	✓			Only find a small number of interesting tuples		
SQLSynthesizer [75]	(E_H)	✓	✓	✓	Only simple ranking function	✓	
SQuID [17]	(E_H)	✓			Does not support the "or" predicate	✓	
AIDE [14]	(E_S)	✓					✓
Huang et al. [28]	(E_S)	✓			Does not support the "or" predicate		✓
Chaudhuri et al. [9]	(P)		✓		Only rank categorical attributes		
Qian et al. [56]	(P)		✓				
Xie et al. [72]	(P)		✓		Only find top-1 result		
Mishra et al. [51]	(P_R)	✓			Refine query to satisfy cardinality constrain		
Y. Weiss et al. [70]	(E_H)	✓				✓	
TALOS [67,68]	(E_H)	✓			Multiple inferred queries are ranked	✓	
Li et al. [34]	(E_H, E_S)	✓				✓	
FastQRE [31]	(E_H)	✓			Multiple inferred queries are ranked	✓	
S4 [55]	(E_H)	✓			Multiple inferred queries are ranked	✓	
Zhang et al. [74]	(E_H)	✓				✓	
PALEO [53,54]	(E_H)	✓	✓	✓	Can only support the "and" predicate	✓	
DExPlorer	(K, E_S, P)	✓	✓	✓			✓

instead of treating *DE-Decision* and *DE-Ranking* separately, by minimizing the interactions with users.

Our methodology We propose DExPlorer with the following main features. [**Interactivity**.] DExPlorer is designed as an interactive system, because providing a set of representative, unbiased, and sufficient samples in one shot is hard. [**Usability**.] It offers an easy-to-use interface for any user, which gives a list of tuples and allows two simple operations "click" and "drag" for the user to provide *true/false* tuple labels and partial orders between tuples. [**Capability**.] In order to infer (possibly) complicated query intent, we propose to jointly train several ML models instead of guessing SQL queries, along the same line of [28], because tightly specifying conditions in SQL queries in data exploration might be hard. More specifically, we use a random forests [38] to infer desired tuples, and a hybrid ranking model combining LambdaMART [71] and ranking SVM to rank tuples. [**Effectiveness**.] We devise novel question selection algorithms that jointly estimate the benefit of soliciting feedback from both tuple labeling and partial-order labeling.

Contributions Our contributions are summarized below.

(1) We present DExPlorer that iteratively trains ML models and on-the-fly predicts result for discovering and ranking tuples with the users. (Section 4)
(2) We present a hybrid decision and ranking model for question selection to minimize human cost. (Section 5)
(3) We describe methods to handle two special cases: the user wants only desired tuples, or only ranked tuples. (Section 5.3)
(4) We conduct extensive experiments to show the effectiveness of DExPlorer. (Section 7)

2 Related work

Table 1 compares different methods for data exploration, and we mainly categorize them to the following three categories: *Keyword-based tuple search* allows users to get interesting tuples by issuing keywords on relational databases without knowing the schema of databases (i.e.,the input type is (K) in Table 1). The language reflective SQL [50] is an extension of SQL, which can be used to search on the relational databases as in a web search engine. DISCOVER [27] is a development of reflective SQL which can support keyword queries on multiple tuples. There have also been many works [19,26,37,39–48,58–60,66,69] on improving the efficiency and effectiveness of keyword search results.

DExPlorer differs from keyword search in the following aspects: (1) It can refine the query results by further interaction with the system, while keyword search provides one-shot answers. (2) It can capture complex combination of predicates, which are hard to express using keywords. (3) It ranks tuples by users' hidden ranking intent, while keyword search ranks tuples by the relevance/proximity of keywords.

Query-by-example QBE [49] either infers an SQL query Q [17,53,55,63,68,70,74], which is a query reverse engineering

(QRE) approach, or learns a machine learning (ML) model M [14,28], which is a ML-based approach, over a database D using input tuple examples. Based on input tuple examples, QRE approaches try to infer Q such that $Q(D)$ is equal (or similar) to the input positive examples; ML approaches try to train a model M such that M can correctly classify input positive and negative examples.

The input tuple examples can be classified to two types: (1) *examples offered by the human* (denoted by (E_H) in Table 1): the tuple examples are provided by users [17,31,53–55,67,68,70,74,75]; and (2) *examples offered by the system* (denoted by (E_S) in Table 1): the tuple examples are provided by the data exploration system and users only need to label them [14,28,34].

DExPlorer differs from existing QBE approaches in: almost all existing QBE approaches only focus on *DE-Decision*. Besides DExPlorer, only SQLSynthesizer [75] and PALEO [53,54] can discover an SQL query with ranking. But SQLSynthesizer [75] can only support simple (hierarchical) ranking functions such as ORDER BY attribute *year* first and then ORDER BY attribute *kilometer*; PALEO [53,54] requires the user to specify the column that the input tuples are sorted by. Besides, the input types of SQLSynthesizer and PALEO are (E_H). No existing approach can support a combination of selection conditions and a more natural (but maybe more complicated) ranking function such as $-0.018 \times price + 0.982 \times powerPS$ with input type (E_S), which is more user-friendly and suitable for the exploration scenario compared with (E_H), because users are not familiar with the database which is to be explored.

Ranking database tuples There have been works on tuple ranking: Chaudhuri et al. [9] rank categorical SQL query results based on intuitions inferred from past workloads, which cannot support ranking on both categorical and numeric attributes. Qian et al. [56] and Xie et al. [72] rank tuples by computing a weighted sum score for each tuple, and the weight vector is learned from users' labeled partial orders, and the input of them are partial orders for tuple pairs, i.e.,the input type is (P) in Table 1. DExPlorer supports more general cases than the above approaches, as shown in Table 1.

3 Problem statement

Informally speaking, given a relational table T, the user wants to find a subset $R \subseteq T$, and tuples in R are ranked.

In practice, a user's query intent might be formulated as either an SQL query, machine learning (ML) models, or some specific logic. Generally speaking, we want to infer/learn a method/model $f()$ as the proxy of a user's query intent, where $f(T)$ returns the ranked list R' that is close to R.

Example 1 Suppose a user wants a manual petrol car that is sold after year 2010, not provided by commercial sellers, and its brand can be either BMW with price $\leq$ 10000, or Volkswagen with price $\leq$ 8000. Moreover, assume that the user wants all cars to be ranked, e.g.,using a weighted sum function: $-0.018 \times price + 0.982 \times powerPS$. That is, the query intent could be expressed as Q_1 below:

```
SELECT * FROM Car
WHERE seller != "commercial" AND year ≥ 2010
AND gearbox="manually" AND fuelType="petrol"
AND ((brand = "bmw" AND price ≤ 10000) OR
(brand = "volkswagen" AND price ≤ 8000))
ORDER BY -0.018 * price + 0.982 * powerPS DESC;
```

Q_1

Example 1 shows a user's query intent may be: (1) **hard to specify:** there might have complicated predicates in the WHERE clause and it is almost impossible to manually specify the weighted sum function with correct parameters in the ORDER BY clause, and (2) **hard to infer:** no existing work (see Table 1) can effectively infer such a query, not to say more complicated cases.

Remark Note that we do not request a user to know exactly how the SQL (i.e.,Q_1) is written, because the user may not be an expert and even not know SQL at all. The user only needs to know whether a tuple is desired and which tuple is preferred. For ease of representation, we assume that the user's intent can be expressed through an SQL query Q_1, and we write Q_1 to help the reader to understand the user's intent.

User operations We allow two *user operations*:

(1) *true/false* labeling: the user may label a given tuple as *true*(desired) or *false*(not desired); and
(2) a partial order: given two tuples t_i and t_j, the user might tell which tuple is more preferable.

User questions A *question* $\mathbb{I}$ is a list of k tuples, which solicits two types of labels from a user:

(1) *decision:* the user will split $\mathbb{I}$ into three disjoint sets of tuples: $\mathbb{D}^+$ with *true* labels, $\mathbb{D}^-$ with *false* labels, and $\mathbb{D}^?$ meaning unknown. Let $\mathbb{D}$ be the set of all tuple labels, i.e.,$\mathbb{D} = \mathbb{D}^+ \cup \mathbb{D}^- \cup \mathbb{D}^?$.
(2) *ranking:* rank (partial) pairs of tuples in $\mathbb{D}^+$, which gets a set $\mathbb{R}$ of partial orders. Implicitly, any tuple $t_i \in \mathbb{D}^+$ is more preferred than any tuple $t_j \in \mathbb{D}^-$, and no need to rank two tuples in $\mathbb{D}^-$, denoted by $t_x \equiv t_y$ for any $t_x, t_y \in \mathbb{D}^-$.

Example 2 The table in Fig. 1 shows the labeling examples based on Q_1 in Example 1. (1) The user can *annotate* the

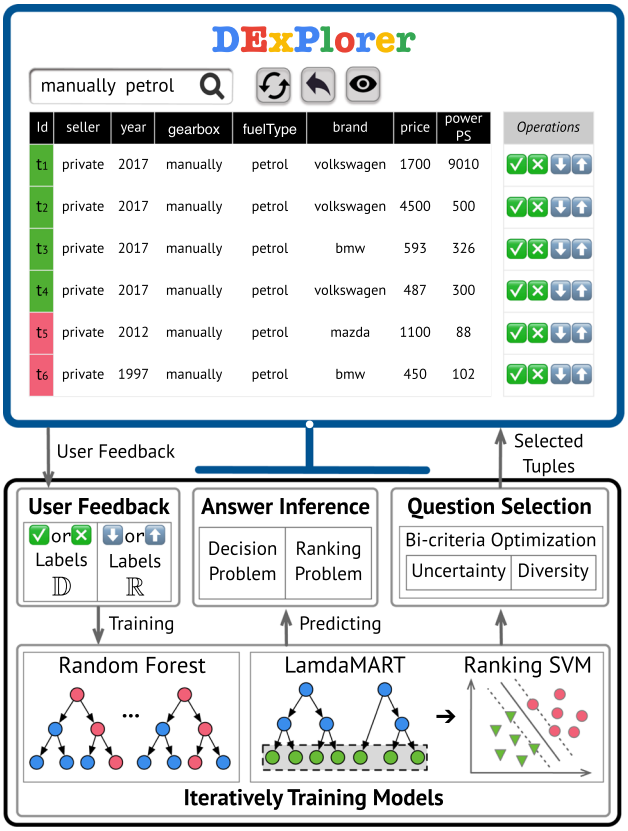


Fig. 1 An overview of DExPlorer

tuples as desired or undesired, e.g.,$\mathbb{D}^+ = \{t_1, t_2, t_3, t_4\}$ and $\mathbb{D}^- = \{t_5, t_6\}$. (2) The user can specify that which cars are more preferred for tuples in $\mathbb{D}^+$, either through a total order such as $(t_1 \succ t_2 \succ t_3 \succ t_4)$ or a set of partially ordered lists such as $(t_1 \succ t_2 \succ t_3)$ and $(t_1 \succ t_4)$.

Problem statement Given a table T, a parameter k (i.e.,the number of tuples to show in each question), and a budget n for the number of questions, the problem of *interactive data exploration* (**IDE**) studied in our paper is to iteratively ask n questions $\{\mathbb{I}_1, \ldots, \mathbb{I}_n\}$ ($|\mathbb{I}_i| = k$ for $i \in [1, n]$) to the user, and infer a ranked set R of tuples based on the user feedback.
Two research challenges In order to solve IDE, there are two main research challenges: (1) *Answer Inference*: *how to infer R based on the user feedback for* $\mathbb{I}_1, \ldots, \mathbb{I}_n$? (Sect. 4) (2) *Question Selection*: *how to select a question* $\mathbb{I}_i$ in the i-th iteration such that the human cost is minimized? (Sect. 5)
Two special cases (1) *IDE-Decision*: the user has no preference between desired tuples; and (2) *IDE-Ranking*: the user only wants to rank all tuples.
Data exploration on multiple tables If users want to conduct data exploration on multiple tables, they should first specify which tables are of interest to them (and how to join these tables if there are multiple joining paths), and then DExPlorer joins these related tables. Then users can perform data exploration on this joined table. Our solution mainly focuses on inferring the decision condition and ranking criteria of an SQL query, rather than inferring the join graph of an SQL query (although it is an important topic in many QRE approaches [31,55,68,74]). For simplicity, we discuss our solution for one table in the rest of the paper.

4 Overview of DExPlorer

4.1 System overview (Fig. 1)

Front-end It interacts with the user in multiple iterations until user budget is used up or the answer cannot be improved. At each iteration, it provides a question I with k tuples, on which two operations are permissible: (1) "*click*" to annotate a tuple to be either *true* or *false*; and (2) "*drag*" to annotate that one is ranked higher than another.

The answers annotated by the user are then transformed to a set of *true/false* labels of tuples in $\mathbb{I}$, and a set $\mathbb{R}$ of partial orders between pairs of tuples in $\mathbb{I}$.
Back-end In the i-th iteration, the user will provide a set $\mathbb{D}_i$ of *true/false* labels and a set $\mathbb{R}_i$ of partial orders, the back-end of DExPlorer needs to address two problems: answer inference and question selection.
Answer inference Given the user feedback from all i iterations, i.e.,$\{\mathbb{D}_1, \mathbb{D}_2, \ldots, \mathbb{D}_i\}$ and $\{\mathbb{R}_1, \mathbb{R}_2, \ldots, \mathbb{R}_i\}$, it is to infer the (ranked) result R.
Question selection It is to select a set $\mathbb{I}_i$ with k tuples for the user to annotate in the i-th iteration.

The major challenge is that question selection for IDEis a bi-criteria problem that needs to estimate the *true/false* labeling and the partial-order labeling between tuples.
Termination The process will terminate, either if the user budget is used up, or the back-end inference will converge.

In the following, we will focus on answer inference. The details for question selection will be discussed in Sect. 5.

4.2 Answer inference

Note that if the function $f()$ can be expressed by a simple SQL query, then using previous methods (e.g.,SQLSynthesizer [75]) is enough. In practice, nevertheless, it might be hard to capture a user's query intent by simple SQL queries. Consequently, ML-based methods (e.g.,using decision trees in [28]) have been used for the IDE-decision problem, and different mathematical models [56,72] have been studied to model the IDE-ranking problem. As discussed earlier in Table 1, existing methods fall short of modeling many real-world cases. Hence, we advocate to jointly use several more advanced ML models for interactive data exploration.
A Naïve solution A straightforward solution is to learn a function $f(T)$ (for example, a learning-to-rank [24] model

such as ranking SVM [30]) that can rank and score tuples in T (e.g.,$\langle(t_1, 0.99), (t_2, 0.98), \ldots, (t_n, 0.04)\rangle$) and select a threshold (e.g.,$\theta = 0.8$) to separate desired and undesired tuples—all the tuples above the threshold are returned as desired tuples and ranked based on their corresponding scores.

Example 3 Suppose a user wants to buy a BMW car, and ranks them by a weighted sum function: $-0.018 * \text{price} + 0.982 * \text{powerPS}$, the ground truth Q can be expressed as:

```
SELECT * FROM Car WHERE brand = "bmw"
ORDER BY -0.018 * price + 0.982 * powerPS DESC;
```
Q_2

Let's use ranking SVM to rank tuples, we may get weights w_1, w_2 for attribute price and powerPS, and w_3 for one-hot attribute of the value bmw, that is, the score of a tuple t is $w_1 \times t[\text{price}] + w_2 \times t[\text{powerPS}] + w_3 \times t[\text{brand=bmw}]$. However, if there is an Audi car with a high $w_1 \times t[\text{price}] + w_2 \times t[\text{powerPS}]$ value, it may be ranked high although it is not desired.

Example 3 shows that although ranking SVM (or other learning-to-rank models) can be used, it is not ideal to handle the combined cases of both decision and ranking. Similarly, a decision-only model (e.g.,a binary classifier) is also not ideal.

Next, we will discuss how DExPLORER does the decision answer inference, the ranking answer inference, and then the hybrid approach of combining them for the IDE problem.

Decision inference Essentially, it is a binary classification problem—deciding whether a tuple is desired or not. There are several choices: decision tree (DT), random forests (RF) [38], or support vector machines (SVM) [11]. [14] uses DT for IDE-Decision. However, as observed by [28], DT is not ideal to capture complex predicates.

DExPLORER employs RF for three reasons.

(1) RF, a collection of many DTs, is more robust to capture complicated cases (e.g.,with *if-then-else*, *and*, *or* and *not*) [65] and better prevents overfitting, compared with DTs.

(2) In some cases, RF, with shallow decision trees and a few branches, is still interpretable. That is, using RF does not sacrifice too much interpretability. By extracting true branches of the trees in the random forest, transforming the conditions in the branches to predicates, and merging predicates of these true branches by conjunction *or*, we can return an SQL corresponding for current inferred results.

Example 4 Figure 2 shows an example of RF. There are two trees in the forest. The two pink paths in the trees are true branches. The left child of a node does not satisfy the condition of the node, and the right child of a node satisfies the condition of the node. By merging predicates of the two true branches by conjunction *or*, we can return an SQL query Q_3.

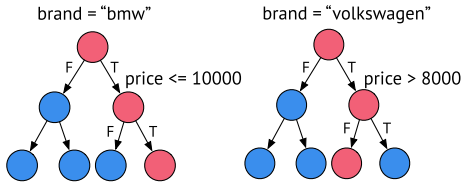


Fig. 2 Explainability of Random Forest

```
SELECT * FROM Car
WHERE (brand = "bmw" AND price ≤ 10000) OR
(brand = "volkswagen" AND price ≤ 8000);
```
Q_3

Remark The interpretability remains to be an issue when the inferred random forest is large and needs to be studied separately in the future work.

(3) As will be seen shortly in question selection (Sect. 5), RF, which outputs a vector with 0's and 1's (each DT in an RF will output either 0 or 1 for one input), is a better choice than SVM that gives a single value for computing diversity between tuples, an important feature for question selection.

Ranking inference We consider to support linear weighted ranking functions, because in many applications, especially in databases with numeric values, a weighted sum among multiple attributes is widely used to model user preferences, which is a common assumption in many data exploration systems [30,52,56,72,73]. In practice, the ranking function may not be linear. But it is common to build a linear function as an approximation of reality, and a linear function often returns good enough results [56], which can also be proved in the experiments.

The ranking problem in DExPLORER is to learn a function $g(t)$. We employ ranking SVM [30,56] as the basic model for two reasons. (1) The scoring function used to rank tuples can be roughly captured by a linear function, i.e., $g(t) = \mathbf{w} \cdot \mathbf{t}$, where $\mathbf{w}$ is a weight vector quantifying the preference and importance of attributes, and $\mathbf{t}$ is the feature vector of tuple t. Ranking SVM is a natural choice for this case. (2) Compared with complicated model like RankNet [1], ranking SVM can avoid overfitting by using a small number of training data.

However, one drawback of using a lightweight model, e.g.,Ranking SVM, is that it needs sophisticated feature engineering. One way to combat this is to use another model to capture more *distinguishing features*, which is inspired by a classical solution for Click-Through Rate (CTR) prediction.

In many commercial IR systems [23,25], the solution for CTR follows the GBDT + LR [25] framework, where GBDT is gradient boosting decision tree and LR is linear regression. More specifically, the CTR problem is a binary classification problem to predict whether an advertisement will be clicked by a user. In the model, the sub-model GBDT can capture

distinguishing features combination and LR is used to make prediction based on these features.

LambdaMART + Ranking SVM Inspired by the GBDT + LR model, we propose a hybrid ranking model: LambdaMART + Ranking SVM, where LambdaMART [71] is a tree structure model based on the MART (Multiple Additive Regression Tree) [18] that transforms the input features. The "Lambda" in LambdaMART denotes a special Lambda value which is the negative gradient in the MART algorithm. The output of LambdaMART is then fed to ranking SVM to infer tuple ranking.

4.3 Iteratively training and predicting

Given $\{\mathbb{D}_1, \mathbb{D}_2, \ldots, \mathbb{D}_i\}$ as decision labels and $\{\mathbb{R}_1, \mathbb{R}_2, \ldots, \mathbb{R}_i\}$ as partial-order labels, we describe how to use them to train and predict for decision and ranking problem. Before that, we first describe how to get the feature vector of a tuple t.

Getting feature vector For a tuple t, it always has columns of different data types. In this paper, we consider four commonly-used data types: numeric, categorical, textual, and date. We then describe how to encode columns of different data types of a tuple t.

- Numeric. We normalize the values in a numeric column to [0, 1].
- Categorical. We use one-hot encoding to encode categorical column values. And for a categorical column with many distinct values (e.g.,thousands of distinct values), we treat it as a textual column.
- Textual. We embed the strings in a textual column to fixed-length (e.g.,128) vector embeddings. The intuition is that: the more similar two strings are (i.e.,the distance, e.g.,Cosine distance, Edit distance, between the two strings is small), the closer the embeddings of them are (i.e.,the Euclidean distance between the two embeddings is small). Please refer to [12] for more details about string embedding.
- Date. We split each date to three columns: year, month and day. Then we treat the three columns as numeric columns.

After we encode each column value of t, we concatenate the encodings of all column values by the order of the columns in T, and then we can get the feature vector of t, denoted by $\mathbf{t}'$.

Remark The encoding method for categorical attributes with high cardinalities may not be optimal. And we would like to explore more encoding methods for this case in the future work, for example, using label encoding (i.e.,randomly assign an integer from 1 through N to a category, where N is the number of categories, and different categories correspond to different integers) and adding more nonlinear modules to alleviate the problem that label encoding would bring some orders between categorical values.

Training Algorithm 1 shows the training process of the i-th iteration, where the inputs are the *true/false* labels $\{\mathbb{D}_1, \mathbb{D}_2, \ldots, \mathbb{D}_i\}$ and partial-order labels $\{\mathbb{R}_1, \mathbb{R}_2, \ldots, \mathbb{R}_i\}$ from the 1st to i-th iteration, and the outputs are the trained models in the i-th iteration.

(1) *Decision training using RF:* it trains RF using all labeled positive tuple examples (i.e.,tuples in $\{\mathbb{D}_1^+, \mathbb{D}_2^+, \ldots, \mathbb{D}_i^+\}$), negative tuple examples (Here we include unknown tuples into negative examples, i.e., $\mathbb{D}^- = \{\mathbb{D}_1^-, \mathbb{D}_2^-, \ldots, \mathbb{D}_i^-\} \cup \{\mathbb{D}_1^?, \mathbb{D}_2^?, \ldots, \mathbb{D}_i^?\}$. Then we obtain the model RF_i (lines 1 - 3).

(2) *Ranking training using LambdaMART + Ranking SVM:* Given a set of features of ordered tuple pairs $\mathbb{R}_i$ as training examples: (a) We first use the labeled pairwise tuples in $\mathbb{R} = \mathbb{R}_1 \cup \mathbb{R}_2 \cup \ldots \cup \mathbb{R}_i$ to train the LambdaMART model, obtaining LM_i (lines 4 - 5). The training inputs of LM_i are $\{(\mathbf{t}'_y, \mathbf{t}'_z, label(t_y, t_z)) \mid (t_y, t_z) \in \mathbb{R}\}$, where $label(t_y, t_z)$ is the partial order of t_y and t_z: if $t_y \succ t_z$, $label(t_y, t_z) = 1$; otherwise, $label(t_y, t_z) = 0$. (b) For each tuple t (whose feature vector is denoted as $\mathbf{t}'$) appearing in $\mathbb{R}$, $LM_i(t)$ will output a m-dimension transformed feature vector $\mathbf{x} = (x_1, \ldots, x_m)$, where m is the number of trees in LambdaMART (see Fig. 1), and x_j ($j \in [1, m]$) is the leaf node index of $\mathbf{t}'$ that ends up falling in the j-th tree (lines 8–12). Let $\mathbf{t}''$ be the one-hot encoding of the transformed feature vector $\mathbf{x}$ (line 13). Let $\mathbf{t}$ be the concatenation ($\oplus$) of $\mathbf{t}'$ and $\mathbf{t}''$ (line 14). (c) For each ordered tuple pair (t_y, t_z) in $\mathbb{R}_i$, we use the enriched features $(\mathbf{t}_y, \mathbf{t}_z)$ to incrementally train ranking SVM, obtaining RS_i (lines 15–18). The training inputs of RS_i are $\{(\mathbf{t}_y, \mathbf{t}_z, label(t_y, t_z)) \mid (t_y, t_z) \in \mathbb{R}\}$, where $label(t_y, t_z)$ is the partial order of t_y and t_z: if $t_y \succ t_z$, $label(t_y, t_z) = 1$; otherwise, $label(t_y, t_z) = 0$. (d) Finally, the trained random forest model RF_i, LambdaMart model LM_i, and ranking SVM model RS_i are returned in the i-th iteration (line 19).

Example 5 Consider the two trees in Fig. 1 generated by LambdaMART. The first tree has four leaves and the second tree has three leaves. Assume that $\mathbf{t}' = (0.8, 0.7)$ ends up falling in the second and third leaf node of tree 1 and tree 2, respectively. Thus, the transformed feature for $\mathbf{t}'$ is (1, 2), with the corresponding one-hot encoding for the first and second features to be (0, 1, 0, 0) and (0, 0, 1), respectively. Hence, $\mathbf{t}'' = (0, 1, 0, 0, 0, 0, 1)$; $\mathbf{t} = \mathbf{t}' \oplus \mathbf{t}'' = (0.8, 0.7, 0, 1, 0, 0, 0, 0, 1)$.

Prediction Algorithm 1 shows the prediction process of the i-th iteration, where the inputs are the table T and the trained models RF_i, LM_i, RS_i from the training process of the i-th

Input: $\{\mathbb{D}_1, \mathbb{D}_2, \ldots, \mathbb{D}_i\}$ and $\{\mathbb{R}_1, \mathbb{R}_2, \ldots, \mathbb{R}_i\}$
Output: trained models RF_i, LM_i, RS_i
1 $\mathbb{D}^+ \leftarrow \mathbb{D}_1^+ \cup \mathbb{D}_2^+ \cup \ldots \cup \mathbb{D}_i^+$;
2 $\mathbb{D}^- \leftarrow \mathbb{D}_1^- \cup \mathbb{D}_2^- \cup \ldots \cup \mathbb{D}_i^- \cup \mathbb{D}_1^? \cup \mathbb{D}_2^? \cup \ldots \cup \mathbb{D}_i^?$;
3 $RF_i \leftarrow TrainRandomForest(\mathbb{D}^+, \mathbb{D}^-)$;
4 $\mathbb{R} = \mathbb{R}_1 \cup \mathbb{R}_2 \cup \ldots \cup \mathbb{R}_i$;
5 $LM_i \leftarrow TrainLambdaMart(\mathbb{R})$;
6 $m \leftarrow$ the number of trees in LM_i;
7 **for** t **in** $\mathbb{R}$ **do**
8 $\quad \mathbf{t}' \leftarrow$ the feature vector of t;
9 $\quad \mathbf{x} \leftarrow ()$;
10 $\quad$ **for** $j \leftarrow 1$ **to** m **do**
11 $\quad\quad x_j \leftarrow$ the leaf node index of $\mathbf{t}'$ that ends up falling in the j-th tree of LM_i;
12 $\quad\quad \mathbf{x} \leftarrow \mathbf{x} \oplus x_j$;
13 $\quad \mathbf{t}'' \leftarrow$ the one-hot encoding of $\mathbf{x}$;
14 $\quad \mathbf{t} \leftarrow \mathbf{t}' \oplus \mathbf{t}''$;
15 $\mathbb{R}' \leftarrow \emptyset$;
16 **for** (t_y, t_z) **in** $\mathbb{R}$ **do**
17 $\quad \mathbb{R}' \leftarrow \mathbb{R}' \cup (\mathbf{t}_y, \mathbf{t}_z)$;
18 $RS_i \leftarrow TrainRankingSVM(\mathbb{R}')$;
19 **return** RF_i, LM_i, RS_i;

Algorithm 1: Training Process for the i-th Iteration

iteration, and the output is the ranked desired tuples R_i in the i-th iteration. The overall process is to first find the desired tuples using the decision model, and then rank them using the ranking model. To be specific, (1) *Decision prediction:* it uses RF_i to predict each unlabeled tuple, predicting R_i as a set of desired tuples (i.e.,these tuples that are predicted *true* by RF_i) (lines 1–5). For a tuple t, the input of RF_i is $\mathbf{t}'$. (2) *Ranking prediction:* For each $t \in R_i$: (a) It uses trained LM_i to get the enriched feature $\mathbf{t}$ of t (line 8 of Algorithm 2, and lines 8–14 of Algorithm 1 show how to get enriched feature $\mathbf{t}$ of t); (b) It computes a score for t using the learned weight function $\mathbf{w}$ in RS_i as $\mathbf{w} \cdot \mathbf{t}$ (line 9); and (c) It ranks tuples in R_i based on their scores, and returns R_i (lines 10–11).

Input: T, trained models RF_i, LM_i, RS_i
Output: inferred ranked desired tuples R_i
1 $R_i \leftarrow \emptyset$;
2 **for** t **in** T **do**
3 $\quad label(t) \leftarrow PredictByRandomForest(RF_i, t)$;
4 $\quad$ **if** $label(t)$ **is** true **then**
5 $\quad\quad R_i \leftarrow R_i \cup \{t\}$;
6 $\mathbf{w} \leftarrow$ the learned weight vector of RS_i;
7 **for** t **in** R_i **do**
8 $\quad \mathbf{t} \leftarrow GetEnrichedFeature(LM_i, t)$;
9 $\quad score(t) = \mathbf{w} \cdot \mathbf{t}$;
10 sort R_i by descending order of $score(t)$;
11 **return** R_i;

Algorithm 2: Prediction Process for the i-th Iteration

5 Question selection

In each user iteration, we need to select a list of k tuples from the table T as one question $\mathbb{I}$, and then the user should provide *true/false* labels and the partial order labels for the k tuples (i.e.,the input type of DExPLORER is (E_S)). An *interactive data exploration* system whose input type is (E_S) should develop an algorithm to sample tuples for users to label in each iteration such that the human cost is minimized. To minimize human cost, existing works either (1) select the tuples with high decision uncertainties only for IDE-Decision problem [17,28,63], i.e.,select the tuples in which the data exploration systems are not sure about whether they are *true/false*; or (2) select the tuple pairs with high ranking uncertainties only for IDE-Rankingproblem [29,56,72], i.e.,select the tuple pairs in which the inferred ranking functions are not sure about their rankings. Different from the above works which only focus on one of IDE-Decision and IDE-Ranking problem, DExPLORER holistically solve the general IDE problem and consider both decision and ranking uncertainties as well as diversity, so the question selection problem of DExPLORER becomes challenging. There are three challenges for the problem of IDE question selection.

Challenge 1 [*Reducing Uncertainty for Decision and Ranking*.] Note that we have different models for decision and ranking answer inference. Also, the training examples for decision model are tuples, while the training examples for the ranking model are tuple pairs. Therefore, the first challenge is how to select tuples such that the uncertainty of both the decision and ranking models is reduced.

Challenge 2 [*Increasing Diversity of Tuple List*.] Existing works either provide sample tuples for decision question [17,28,63] or partial orders [2–4,8,9,35,56,72] for ranking questions. Different from them, DExPLORER provides a list of tuples as one question for one user iteration. Hence, besides that the tuples in the list should reduce uncertainty of both models, the tuples should also be diverse and representative.

Challenge 3[*Interactive Question Selection*.] As will be shown shortly, considering both the decision and ranking uncertainty and diversity of the selected k tuples, the IDEquestion selection problem is NP-Hard, and existing algorithms cannot provide the question $\mathbb{I}$ in interactive time. Thus, how to design an efficient yet effective algorithm for IDEquestion selection is important.

5.1 Uncertainty and diversity

Uncertainty is the most commonly used criteria for tuple selection (i.e., **Challenge 1**), which measures the confidence of the current model on evaluating a question.

Uncertainty for decision questions We define the decision uncertainty of a tuple t as the entropy of the predicted results of all decision trees in the random forest. Suppose

we have n trees, and m is the number of trees which predicted the tuple t as positive, we use the *entropy* which is a general uncertainty sampling measure in active learning [61,62] as the decision uncertainty of tuple t: $u(t) = -(e \log e + (1-e)\log(1-e))$,[1] where $e = \frac{m}{n}$. By *entropy*, the tuples whose probability of being predicted positive are nearing 0.5 are more likely to be sampled, which is also a widely used uncertainty sampling strategy for binary classification [32,33]. We denote $u(t)$ as the decision uncertainty of tuple t: the larger $u(t)$ is, the larger the decision uncertainty of t is. And we denote $1 - u(t)$ as the decision certainty of tuple t.

Example 6 Suppose we have five trees in RF. For tuple t, assume that there are 3 trees predicting t as positive, then the decision uncertainty of t is $u(t) = -(\frac{3}{5}\log\frac{3}{5} + \frac{2}{5}\log\frac{2}{5}) = 0.97$.

Uncertainty for ranking questions Given a pair of tuples t_i and t_j, the ranking SVM model uses a hyperplane to make decision about whether $t_i \succ t_j$. It also learns a parameter vector $\mathbf{w}$, such that $\mathbf{w} \cdot \mathbf{t}_i$ denotes the ranking score of tuple t_i. The higher the score is, the higher the ranking t_i should be. If the pair is above the hyperplane, i.e.,if $\mathbf{w} \cdot \mathbf{t}_i - \mathbf{w} \cdot \mathbf{t}_j > 0$, then $t_i \succ t_j$, and vice versa. Thus, those pairs close to the hyperplane are the uncertain ones, i.e.,when $|\mathbf{w} \cdot \mathbf{t}_i - \mathbf{w} \cdot \mathbf{t}_j|$ is close to 0. We denote $|\mathbf{w} \cdot \mathbf{t}_i - \mathbf{w} \cdot \mathbf{t}_j|$ as the ranking certainty of tuple pair (t_i, t_j): the smaller $|\mathbf{w} \cdot \mathbf{t}_i - \mathbf{w} \cdot \mathbf{t}_j|$ is, the larger the ranking uncertainty of (t_i, t_j) is.

Example 7 Suppose we have three tuples t_1, t_2 and t_3, a parameter vector $\mathbf{w} = (0.5, 0.5)$, and $k = 2$. And we have $\mathbf{t}_1 = (0.8, 0.7)$, $\mathbf{t}_2 = (0.8, 0.6)$, $\mathbf{t}_3 = (0.6, 0.6)$. Then, we compute $|\mathbf{w} \cdot \mathbf{t}_i - \mathbf{w} \cdot \mathbf{t}_j|$ for each pair, i.e.,$|\mathbf{w} \cdot \mathbf{t}_1 - \mathbf{w} \cdot \mathbf{t}_2| = 0.05$, $|\mathbf{w} \cdot \mathbf{t}_2 - \mathbf{w} \cdot \mathbf{t}_3| = 0.1$, $|\mathbf{w} \cdot \mathbf{t}_1 - \mathbf{w} \cdot \mathbf{t}_3| = 0.15$. Since the score of $|\mathbf{w} \cdot \mathbf{t}_1 - \mathbf{w} \cdot \mathbf{t}_2| = 0.05$ is the closest to 0 (i.e.,the closest to the hyperplane), we choose tuples t_1 and t_2 for preference labeling.

Besides selecting tuples that the decision and ranking models are uncertain about, we should also care about the diversity of the selected tuples (i.e., **Challenge 2**). Consider two tuples with high uncertainty but with highly similar content, it is a waste of human efforts to label both of them because only labeling one of them can produce a model that is very likely to predict the other correctly. Naturally, we should also consider the diversity of selected tuples.

Diversity A simple way to measure the diversity is to compute the string similarity of each tuple pair using some predefined function. However, such method does not consider the semantic information (dynamically) incorporated by user feedback. Thus, we can use the feature vectors produced by RF and LambdaMART to measure the similarity of two tuples. Let $\mathbf{v}'(t)$ be the prediction vector of all trees in RF for tuple t, where the i-th element in $\mathbf{v}'(t)$ denotes the prediction label (1 or 0) of the i-th tree in the random forest. Let $\mathbf{v}''(t)$ be the one-hot encoding transformed feature vector of tuple t output by the LambdaMART sub-model (i.e., $\mathbf{v}''(t) = \mathbf{t}''$).

We concatenate $\mathbf{v}'(t)$ and $\mathbf{v}''(t)$ as $\mathbf{v}(t)$, which is used to compute the diversity. We denote $s(t_i, t_j)$ as the similarity of tuple pair (t_i, t_j): the smaller the similarity, the larger the diversity between two tuples is. More specifically, we define the similarity $s(t_i, t_j)$ of tuple t_i and t_j using the Cosine similarity: $s(t_i, t_j) = cos(\mathbf{v}(t_i), \mathbf{v}(t_j)) = \frac{\mathbf{v}(t_i)\cdot\mathbf{v}(t_j)}{||\mathbf{v}(t_i)||\times||\mathbf{v}(t_j)||}$.

Example 8 Consider two tuples t_1 and t_2. The prediction vectors of RF for t_1 and t_2 are $\mathbf{v}'(t_1) = (1, 0, 1)$, $\mathbf{v}'(t_2) = (1, 1, 1)$. The one-hot encoding transformed features for t_1 and t_2 are $\mathbf{v}''(t_1) = (0, 1, 0, 1)$, $\mathbf{v}''(t_2) = (1, 0, 0, 1)$. Then $\mathbf{v}(t_1) = (1, 0, 1, 0, 1, 0, 1)$, $\mathbf{v}(t_2) = (1, 1, 1, 1, 0, 0, 1)$, and the similarity of t_1 and t_2 is $s(t_1, t_2) = cos(\mathbf{v}(t_1), \mathbf{v}(t_2)) = 0.67$.

We want to select k tuples with maximum uncertainties (including decision and ranking uncertainties) and maximum diversities. In other words, we want to select k tuples with minimum certainties (including decision and ranking certainties) and minimum similarities. So we define the IDEquestion selection problem as follows.

Definition 1 (IDE*Question selection*) Given a partially trained RF model and hybrid ranking model, a table T, and a number k, the problem is to select a set of k tuples S^* from T such that the following equation is minimized:

$$S^* = \underset{S\subseteq T, |S|=k}{\arg\min}\ \gamma \sum_{t\in S}(1-u(t)) + \frac{2}{(\max(\mathbf{w}\cdot\mathbf{t}) - \min(\mathbf{w}\cdot\mathbf{t}))(k-1)}\alpha' \sum_{t_i,t_j\in S}|\mathbf{w}\cdot\mathbf{t}_i - \mathbf{w}\cdot\mathbf{t}_j| + \frac{2}{k-1}\beta' \sum_{t_i,t_j\in S} s(t_i,t_j) \quad (1)$$

there are three summations of Eq. 1. The first summation is the sum of decision certainties, where $1 - u(t)$ is the decision certainty of tuple t, and $1 - u(t) \in [0, 1]$; The second summation is the sum of ranking certainties, where $|\mathbf{w} \cdot \mathbf{t}_i - \mathbf{w} \cdot \mathbf{t}_j|$ is the ranking certainty of tuple pair (t_i, t_j), $\mathbf{w}$ is the weight vector output by the hybrid ranking model and $|\mathbf{w} \cdot \mathbf{t}_i - \mathbf{w} \cdot \mathbf{t}_j| \in [0, \max(\mathbf{w} \cdot \mathbf{t}) - \min(\mathbf{w} \cdot \mathbf{t})]$; the third summation is the sum of similarities, where $s(t_i, t_j)$ is the Cosine similarity of tuple pair (t_i, t_j), and $s(t_i, t_j) \in [0, 1]$, because there is no negative elements in $\mathbf{v}(t)$. There are k elements of the first summation, and $\frac{k(k-1)}{2}$ elements of the second and third summations. To scale the three summations to the same order of magnitude (i.e.,same range), there is a coefficient $\frac{2}{(\max(\mathbf{w}\cdot\mathbf{t})-\min(\mathbf{w}\cdot\mathbf{t}))(k-1)}$ of the second summation, and there is a coefficient $\frac{2}{k-1}$ of the third summation.

[1] The logarithmic function takes 2 as the base, and we can know that $u(t) \in [0, 1]$ for $e \in [0, 1]$, and $u(t) = 1$ when $e = 0.5$.

γ, α', β' are parameters to control the proportions of decision uncertainty, ranking uncertainty, and diversity. For example, to make decision uncertainty and ranking uncertainty equivalently important, we can set $\gamma = 1$ and $\alpha' = 1$; to make uncertainty and diversity equivalently important, we can set $\beta' = 2$.

Equation 1 is long and complex. By dividing Eq. 1 by γ (which does not change the optimal solution of Eq. 1), and setting $\alpha = \frac{2\alpha'}{\gamma(\max(\mathbf{w}\cdot\mathbf{t})-\min(\mathbf{w}\cdot\mathbf{t}))(k-1)}$ and $\beta = \frac{2\beta'}{\gamma(k-1)}$, we can get:

$$S^* = \arg\min_{S\subseteq T, |S|=k} \sum_{t\in S}(1-u(t)) + \alpha \sum_{t_i,t_j\in S} |\mathbf{w}\cdot\mathbf{t}_i - \mathbf{w}\cdot\mathbf{t}_j| + \beta \sum_{t_i,t_j\in S} s(t_i,t_j) \quad (2)$$

We use Eq. 2 for ease of description in the following.

Theorem 1 *The IDEquestion selection is NP-hard.*

Proof Sketch We can prove that the IDEquestion selection problem is NP-hard by a reduction from the MaximumDispersion problem that is known to be NP-hard [21].

The MaximumDispersion problem is: given an undirected complete graph $G = (V, E)$, where $V = (v_1, v_2, \ldots, v_n)$ contains all vertexes and each edge (v_i, v_j) has an associated nonnegative weight $w(v_i, v_j)$; given $k \in \{2, \ldots, n\}$, and $p \in 1, 2, \ldots, \lfloor \frac{n}{k} \rfloor$, the MaximumDispersion problem is to find p disjoint subsets $V_1, \ldots, V_p$ of V, with $|V_q| = k, q = 1, \ldots, p$, such that $\sum_{q=1}^{p} \sum_{(v_i,v_j)\in V_q\times V_q} w(v_i, v_j)$ is maximized. The MaximumDispersion problem is NP-hard [21]. By setting each tuple t in T as a node, the edge weight between t_i and t_j as $c-\left(\frac{(1-u(t_i))}{k-1} + \frac{(1-u(t_j))}{k-1} + \alpha|\mathbf{w}\cdot\mathbf{t}_i - \mathbf{w}\cdot\mathbf{t}_j| + \beta s(t_i,t_j)\right)$, where c is a large enough number such that all edge weights are positive, k as the number of tuples that are selected to be labeled by users, and p as 1, we can reduce the known NP-hard MaximumDispersion problem to the IDEquestion selection problem. So the IDEquestion selection problem is NP-hard.

5.2 Question selection algorithms

5.2.1 AQS: an approximate algorithm

Inspired by [20], we present a 2-approximation algorithm, denoted as AQS, for IDEquestion selection problem. The algorithm first initializes an empty result set and calculates a new similarity $s'(t_i, t_j) = \frac{1-u(t_i)}{k-1} + \frac{1-u(t_j)}{k-1} + \alpha|\mathbf{w}\cdot\mathbf{t}_i - \mathbf{w}\cdot\mathbf{t}_j| + \beta s(t_i, t_j)$ for each tuple pair (t_i, t_j). It then selects the tuple pair that has not been added to the result set and with the least s' scores, and adds the pair to the result set. It iteratively runs $\lfloor \frac{k}{2} \rfloor$ times. It adds another tuple to the result set if k is odd and current result has $k-1$ tuples. The selected k tuples are then returned.

Time complexity The time complexity of AQS is $O(\max(|T|^2 L, |T|^2 k))$, where L is the length of tuples. However, the time to compute a question can still be long when $|T|$ is large, thus the above algorithm is not efficient enough to satisfy the online question selection requirement.

5.2.2 IQS: an efficient algorithm

In the previous version [57] of DExPlorer, we propose an efficient approximate algorithm IQS (Algorithm 3) to solve the IDEquestion, which can return approximate result of S^* in interactive time. The main idea of this algorithm is that it first selects the tuple with the highest decision uncertainty score, and adds it to the result set if this tuple has low similarity and high ranking uncertainty with the current selected tuples.

We first sort T by descending order of $u(t)$ (line 1) and initialize the result set S as empty (line 2). Next, we iterate each tuple t_i in T, and check whether t_i has high similarities and low ranking uncertainties with current tuples in S (line 4–8). If yes, we just drop it; otherwise, we add t_i into S. Finally, S is returned as the selected question with k tuples (line 11).

Input: T, k, a weight vector $\mathbf{w}$, a threshold δ
Output: an approximate optimal set S
1 sort T by descending order of $u(t)$, $T \leftarrow [t_1, t_2, ..., t_{|T|}]$;
2 initialize $S \leftarrow \emptyset$;
3 **for** t_i **in** T **do**
4 $flag \leftarrow True$;
5 **for** t_j **in** $|S|$ **do**
6 **if** $\alpha \cdot |\mathbf{w}\cdot\mathbf{t}_i - \mathbf{w}\cdot\mathbf{t}_j| + \beta \cdot s(t_i,t_j) > \delta$ **then**
7 $flag \leftarrow False$;
8 **break**;
9 **if** $flag$ **then** $S \leftarrow S \cup \{t_i\}$;
10 **if** $|S| = k$ **then break;**
11 **return** S;

Algorithm 3: IDE Question Selection (IQS)

Time complexity We first sort T (line 1), and the time complexity for sorting is $O(|T|\log|T|)$. The time complexity to check whether t_i can be added to the result set is $O(kL)$, where L is the length of tuples. The outer loop iterates up to $|T|$ times. Therefore, the time complexity for the lines 3–8 is $O(kL|T|)$. So the time complexity of Algorithm 4 is $O(\max(|T|k\log|T|, kL|T|))$, which is much more efficient than AQS.

5.2.3 IQS+: an efficient and effective algorithm

In this paper, we further propose an efficient yet effective algorithm IQS+ (Algorithm 4). IQS+ is more effective than IQS, but has similar efficiency performance with IQS.

Considering both uncertainty and diversity is hard due to its high computational complexity (NP-hard). We thus

propose to first solve the IDEquestion selection by only considering decision and ranking uncertainty (i.e., $\beta = 0$), and then incorporate the diversity into the solution.

Question selection without diversity We set $\beta = 0$ in Eq. 2 to ignore diversity:

$$S^* = \underset{S \subseteq T, |S|=k}{\arg\min} \sum_{t \in S} (1 - u(t)) + \alpha \sum_{t_i, t_j \in S} |\mathbf{w} \cdot \mathbf{t}_i - \mathbf{w} \cdot \mathbf{t}_j| \tag{3}$$

To better illustrate the optimization problem, we first denote $S = [t_1, t_2, \ldots, t_k]$, where $\mathbf{w} \cdot \mathbf{t}_i \leq \mathbf{w} \cdot \mathbf{t}_j$ iff $i \leq j$, then we expand the second term of Eqs. 3 to 4. Thus, we have:

$$\begin{aligned} S^* &= \underset{S \subseteq T, |S|=k}{\arg\min} \sum_{i=1}^{k} (1 - u(t_i)) + \alpha \sum_{i=1}^{k} ((i-1)\mathbf{w} \cdot \mathbf{t}_i - (k-i)\mathbf{w} \cdot \mathbf{t}_i) \\ &= \underset{S \subseteq T, |S|=k}{\arg\min} \sum_{i=1}^{k} (1 - u(t_i)) + \alpha \sum_{i=1}^{k} (2i - k - 1)\mathbf{w} \cdot \mathbf{t}_i \end{aligned} \tag{4}$$

We first introduce some notations. We sort T by $\mathbf{w} \cdot \mathbf{t}$ in ascending order and obtain a sorted list $T = [t_1, t_2, \ldots, t_{|T|}]$. We use T_m to denote the prefix of list T with length m, i.e., $T_m = [t_1, t_2, \ldots, t_m]$. We define $S(m, n) = \underset{S \subseteq T_m, |S|=n}{\arg\min} \sum_{i=1}^{|S|} (1 - u(t_i)) + \alpha \sum_{i=1}^{|S|} (2i - k - 1)\mathbf{w} \cdot \mathbf{t}_i$ and $F(m, n)$ denotes the corresponding optimal value. We can see that $S(|T|, k)$ is the optimal solution of Eq. 3 and $F(|T|, k)$ is the corresponding optimal value. Then we have:

$$F(m, n) = \min(F(m-1, n), F(m-1, n-1) + \phi(m, n)) \tag{5}$$

where $\phi(m, n) = (1 - u(t_m)) + \alpha(2n - k - 1)\mathbf{w}\mathbf{t}_m$. We devise a dynamic programming algorithm to find the optimal solution.

Dynamic-programming for uncertainty only. It first initializes $F[0...|T|][0...k]$ to all zero, where $F[m][n] = F(m, n)$ and initializes $S[0...|T|][0...k]$ to all $\emptyset$, where $S[m][n] = S(m, n)$. Then we can calculate F by Eq. 5 and update S correspondingly. Finally, we can derive $F[|T|][k]$ and $S[|T|][k]$.

Question selection considering both uncertainty and diversity. When considering both uncertainty and diversity, we optimally choose tuples with high uncertainty by the above dynamic programming algorithm, but when adding a tuple t to the result set S, we check whether t has a high similarity with existing selected tuples. If yes, we just drop it; else, we include it to the result.

Algorithm 4 (IQS+) shows how to do question selection with both uncertainty and diversity. Based on the dynamic programming algorithm, we add a constraint to maintain the selected tuples diversified (i.e.,the similarity between these selected n tuples is less than a threshold δ) as far as possible.

The algorithm first sorts all tuples in T by $\mathbf{w} \cdot \mathbf{t}$ (line 1), and gets the sorted list $T = [t_1, t_2, \ldots, t_{|T|}]$. Then it initializes all $F[0...|T|][0...k]$ to zeros (line 2), and $S[0...|T|][0...k]$ to $\emptyset$ (line 3). First, the same as the above uncertainty-only algorithm, if $F[m-1][n] \leq F[m-1][n-1] + \phi(m, n)$, we will drop t_m (lines 7–8) because discarding t_m will get a better solution. Otherwise, the algorithm checks whether the tuple t_m has a high similarity with existing tuples which have been added into the current result (lines 10) using the function *hasHighSimilarity*(t, S, δ). If so, t_m is also dropped to keep high diversity (lines 11–12). Otherwise, t_m is added to the diversified solution of $S[m][n]$ (lines 14–15). More specifically, *hasHighSimilarity*(t, S, δ) checks whether there exists a tuple $t' \in S$ such that $s(t, t') > \delta$. If so, it returns *true*, and *false* otherwise. Finally, $S[|T|][k]$ is returned as the selected question with k tuples (line 16).

Input: T, k, a weight vector $\mathbf{w}$, a diversity threshold δ
Output: an approximate optimal set S

```
1  sort T by ascending order of w · t, T ← [t_1, t_2, ..., t_|T|];
2  initialize F[0...|T|][0...k] ← 0;
3  initialize S[0...|T|][0...k] ← ∅;
4  for m ← 1 to |T| do
5      for n ← 1 to min(m − 1, k) do
6          if F[m − 1][n] ≤ F[m − 1][n − 1] + φ(m, n) then
7              F[m][n] ← F[m − 1][n];
8              S[m][n] ← S[m − 1][n];
9          else
10             if hasHighSimilarity(t_m, S[m − 1][n − 1], δ) then
11                 F[m][n] ← F[m − 1][n];
12                 S[m][n] ← S[m − 1][n];
13             else
14                 F[m][n] ← F[m − 1][n − 1] + φ(m, n);
15                 S[m][n] ← S[m − 1][n − 1] ∪ {t_m};
16 return S[|T|][k];
```

Algorithm 4: IDE QUESTION SELECTION (IQS+)

Time complexity The time complexity of the *hasHighSimilarity* function is $O(kL)$, where L is the length of tuples. Note that the attributes number in T is small, thus the function is effective. We first sort T (line 1), and the time complexity for sorting is $O(|T| \log |T|)$. The time complexity for the lines 4–15 is $O(k^2 L|T|)$. Thus, the time complexity for Algorithm 4 is $O(\max(|T| \log |T|, k^2 L|T|))$.

Memory complexity When we calculate $F[m][n]$ and $S[m][n]$ (lines 6 - 15), only $F[m-1][n]$, $F[m-1][n-1]$, $S[m-1][n]$, $S[m-1][n-1]$ are used. So when we calculate a new row of F and S (i.e.,$F[m][1...k]$ and $S[m][1...k]$), we only need to store the last row of F and S (i.e.,$F[m-1][0...k]$ and $S[m-1][0...k]$), and the new row of F and S (i.e.,$F[m][0...k]$ and $S[m][0...k]$). Besides, we should first initialize $F[m][0]$ to 0 and $S[m][0]$ to $\emptyset$ before calculating $F[m][1...k]$ and $S[m][1...k]$. When we initialize F and S (lines 2 - 3), we only need to store and initialize the first row of F and S (i.e.,$F[0][0...k]$ and $S[0][0...k]$). In summary, we only need to store at most two rows of F and S at any time, but we use two arrays F and S which both have $(|T|+1)*(k+1)$ elements (i.e.,$|T|+1$ rows) in Algorithm 4

for ease of explanation. To store two rows of F, the memory complexity is $O(2(k+1))$, and to store two rows of S, the memory complexity is $2 \times O(0+1+2+\cdots+k) = O(k(k+1))$, so the memory complexity of Algorithm 4 is $O(2(k+1)+k(k+1)) = O(k^2)$, where k is a small number, for example, k is 10 in our experiments.

Remark As heuristic algorithms, both IQS and IQS+ aim to find the optimal solution of Eq. 2, which consists of three parts (terms), measuring the decision uncertainty, ranking uncertainty and diversity of tuples, respectively. The difference of IQS and IQS+ lies in that IQS (resp., IQS+) mainly optimizes the first term (resp., the first two terms), and designs the solution to address the following two terms (resp., the last term), so as to further optimize the entire equation.

To be specific, IQS can first find the optimal solution of the first term (i.e., the optimal solution of $\arg\min_{S\subseteq T,|S|=k} \sum_{t\in S}(1-u(t))$), and by checking whether a new tuple has high similarities and low ranking uncertainties with existing tuples (*i.e.,* by checking whether $\alpha\cdot|\mathbf{w}\cdot\mathbf{t}_i-\mathbf{w}\cdot\mathbf{t}_j|+\beta\cdot s(t_i,t_j)>\delta$), IQS decides whether to add the new tuple into the current solution; IQS+ first finds the optimal solution of the first two terms of Eq. 2 (i.e., the optimal solution of $\arg\min_{S\subseteq T,|S|=k} \sum_{t\in S}(1-u(t))+\alpha\sum_{t_i,t_j\in S}|\mathbf{w}\cdot\mathbf{t}_i-\mathbf{w}\cdot\mathbf{t}_j|$), and by checking whether a new tuple has high similarities with existing tuples (*i.e.,* by checking whether $s(t,t')>\delta$), IQS+ decides whether to add the new tuple. Hence, we can observe that IQS+ addresses this problem mainly based on the optimal results on the first two terms of Eq. 2 (i.e., the last term is approximated), while IQS is only based on the first term (*i.e.,* the last two terms are all approximated). Therefore, IQS+ is more effective than IQS, which is also empirically verified in experiments (Section 7.1.1).

Example 9 Figure 3 shows an example of computing S when considering only uncertainty (Fig. 3(c)) and both uncertainty and diversity (Fig. 3(d)). Suppose we have 10 tuples, and want to select 5 tuples for labeling. Figure 3(a) shows the variables for calculating decision and ranking uncertainty of $t_1,\ldots,t_{10}$, and Fig. 3(b) shows the similarity of any tuple pairs. For ease of description, let $\alpha=\beta=1$ and $\delta=0.9$. Figure 3(a) shows the matrix S where each cell is a solution without considering diversity during the dynamic programming process and Fig. 3(b) shows the matrix S' when diversity is considered. Note that we only need to store the last and current rows of S and S', but we show the complete matrices for ease of explanation in the example.

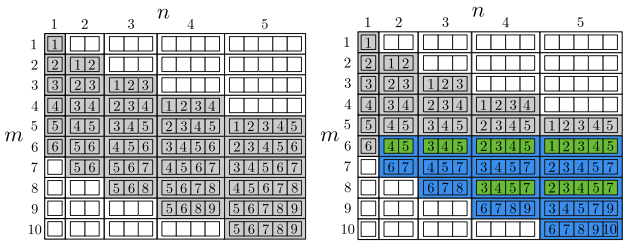
The cells colored in blue in the matrix S' (Fig. 3(d)) are different from those cells in S (Fig. 3(c)) because we consider diversity. The cells $S'[m][n]$ marked by green in Fig. 3(d) are copied from $S'[m-1][n]$ by dropping t_m because t_m has high similarities with some tuples in $S'[m-1,n-1]$ even when $F[m-1][n-1]+\phi(m,n)\le F[m-1][n]$. Take the cell $S[6][2]$ and the green cell $S'[6][2]$ as examples. First, when

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
$1-u(t_i)$	1.0	0.12	0.28	0.53	0.12	0.12	0.53	0	0.12	0.53
$\vec{\mathbf{wt}}_i$	0.002	0.16	0.26	0.49	0.68	0.68	0.70	0.79	0.80	0.94

(a) Variables for Computing Decision and Ranking Uncertainty

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
t_2	0.28								
t_3	0.09	0.45							
t_4	0.02	0.77	0.04						
t_5	0.63	0.03	0.20	0.68					
t_6	0.84	0.78	0.50	0.81	0.97				
t_7	0.13	0.18	0.68	0.79	0.29	0.61			
t_8	0.42	0.21	0.81	0.92	0.01	0.68	0.63		
t_9	0.07	0.27	0.80	0.65	0.13	0.42	0.07	0.87	
t_{10}	0.26	0.52	0.95	0.25	0.79	0.77	0.77	0.01	0.79

(b) Similarity Scores of Tuple Pairs



(c) Matrix S (no diversity) **(d)** Matrix S' (with diversity)

Fig. 3 Example of computing S

we only consider the uncertainty to compute the $S[6][2]$, we have $F[6][2]=\min(F[5][2], F[5][1]+\phi(6,2))$, $F[5][2]=-2.67$, and $F[5][1]+\phi(6,2)=-3.84$. We then observe that $F[5][1]+\phi(6,2)<F(5,2)$, thus adding t_6 to $S[5][1]$ is the optimal solution for selecting 2 tuples in $\{t_1,t_2,\ldots,t_6\}$ ($S[6][2]=\{t_5,t_6\}$). However, now we consider the diversity to compute $S'[6][2]$. Before adding t_6 to $S'[5][1]$, i.e., $\{t_5\}$, we first check whether t_6 has high similarity with $\{t_5\}$. Because $s(t_5,t_6)=0.97>\delta=0.9$ in Fig. 3(d), we cannot add t_6 to $S'[5][1]$, and thus $S'[6][2]=S'[5][2]=\{t_4,t_5\}$.

5.3 Special cases

In this section, we discuss two special cases: IDE-Decision (only *true/false* labels) and IDE-Ranking(only partial orders).

Definition 2 (IDE-Decision *Question selection*) Given a partially trained RF model, a table T, and a number k, the problem is to select a set of k tuples S^* from T so that the following equation is minimized:

$$S^* = \arg\min_{S\subseteq T,|S|=k} \sum_{t\in S}(1-u(t)) + \beta \sum_{t_i,t_j\in S} s'(t_i,t_j) \tag{6}$$

where the notations in Eq. 6 have the same meaning with Eq. 2 except $s'(t_i,t_j)=cos(\mathbf{v}'(t_i),\mathbf{v}'(t_j))$. $\mathbf{v}'(t)$ is the prediction vector of all trees in RF for tuple t, where the i-th element in

$\mathbf{v}'(t)$ denotes the prediction label (1 or 0) of the i-th tree in the random forest.

Solution for IDE-Decision: We still solve the IDE-Decision problem using algorithm 4 but transform Eq. 2 to Eq. 6 by setting $\mathbf{w} \cdot \mathbf{t} = 0$ and replacing $s(t_i, t_j)$ with $s'(t_i, t_j)$.

Definition 3 (IDE-Ranking*Question selection*) Given a partially trained hybrid ranking model, a table T, and a number k, the problem is to select a set of k tuples S^* from T so that the following equation is minimized:

$$S^* = \arg\min_{S \subseteq T, |S|=k} \alpha \sum_{t_i, t_j \in S} |\mathbf{w} \cdot \mathbf{t}_i - \mathbf{w} \cdot \mathbf{t}_j| + \beta \sum_{t_i, t_j \in S} s''(t_i, t_j) \tag{7}$$

where the notations in Eq. 7 have the same meaning with Eq. 2 except $s''(t_i, t_j) = cos(\mathbf{v}''(t_i), \mathbf{v}''(t_j))$. $\mathbf{v}''(t)$ is the one-hot encoding transformed feature vector of tuple t output by the LambdaMART sub-model (i.e., $\mathbf{v}''(t) = \mathbf{t}''$, where $\mathbf{t}''$ is defined in the training paragraph of Sect. 4.3).

Solution for IDE-Ranking: We also solve the IDE-Ranking problem using Algorithm 4 but transform Eq. 2 to Eq. 7 by setting $u(t) = 1$ and replacing $s(t_i, t_j)$ with $s''(t_i, t_j)$.

Remark The similarity function in Eq. 2 is $s(t_i, t_j) = \cos(\mathbf{v}(t_i), \mathbf{v}(t_j))$, where $\mathbf{v}(t)$ is the concatenation of $\mathbf{v}'(t)$ and $\mathbf{v}''(t)$, $\mathbf{v}'(t)$ is output by the RF model of decision problem, and $\mathbf{v}''(t)$ is output by the LambdaMART sub-model of ranking problem. There is no $\mathbf{v}''(t)$ in the IDE-Decision Question Selection problem, so $s'(t_i, t_j) = cos(\mathbf{v}'(t_i), \mathbf{v}'(t_j))$; and there is no $\mathbf{v}'(t)$ in the IDE-RankingQuestion Selection problem, so $s''(t_i, t_j) = cos(\mathbf{v}''(t_i), \mathbf{v}''(t_j))$.

6 Implementation details of DExPlorer

In this section, we discuss the implementation details of our system DExPlorer. We provide DExPlorer as a web service, where different users can visit the website, interact with the front-end, and DExPlorer will return the ranked inferred results to users. We use a Python web framework Django[2] to build our website, and the server is built on a host of our laboratory. More details are discussed in the following.

Data Structures In the front-end, we need to show the k tuples for users to label. The k tuples are passed from the back-end in the form of a JSON string, and shown to users in an interactive table Tabulator[3] in the front-end. In the back-end, there are mainly two things to do: answer inference and question selection. For answer inference, we need to store the feature vectors of the labeled tuples (pairs), and these feature vectors are stored in the list of Python. For question selection, we need to calculate the decision uncertainty and ranking score of each tuple, where each feature vector of a tuple is stored in a list, the decision uncertainties of all tuples are stored in a list, and the ranking scores of all tuples are also stored in a list. Besides, when running Algorithm 4, we need to store the last and current rows of matrices F and S, where the row of F is a list whose element is float, and the row of S is a list whose element is set of integer (an integer denotes the id of a tuple).

Read/Write Operations Read operations mainly occur when loading data from the database to memory, and write operations mainly occur when users upload new datasets to the server, and DExPlorer will store the datasets to the database.

Multi-user interaction There may be multiple users making requests to the server at the same time. We use the Django framework to create a session to identify the user and a thread to run the query for the user in the back-end.

Memory optimizations When the dataset is small, we can load all tuples of the table to the memory (the feature vectors of tuples are stored in lists), and then calculate their decision uncertainties and ranking scores. But when the datasets are big and there is not enough memory, we need to do some memory optimization. We allocate a fixed size of memory for each user. When we do question selection, we load tuples until the allocated memory is used up, calculate decision uncertainties and ranking scores of these tuples, and reload new tuples to the allocated memory. We repeat the above process until all tuples are calculated.

7 Experiments

Datasets We use three datasets: Car, Publication and TPCH. The Car dataset is from the Kaggle[4] website with one relational table; the Publication dataset is crawled from the ACM Digital Library[5] that contains six relational tables; and the TPCH dataset[6] contains eight relational tables. Table 2 gives some statistics of these two datasets, where **#-Tuples**, **#-Col**, **#-Num**, **#-Cat**, **#-Text**, and **#-Date** denote the number of tuples, columns, numeric columns, categorical columns, textual columns and date columns of each table, respectively; and Table 3 gives more statistics of columns of different datasets, where **Column** denotes the name of a column of a dataset, **Type** denotes the data type of a column: i.e.,numeric, categorical, textual, or date, and **Range**

[2] https://www.djangoproject.com/.

[3] http://tabulator.info/.

[4] https://www.kaggle.com/orgesleka/used-cars-database.

[5] https://www.acm.org/publications/digital-library.

[6] https://relational.fit.cvut.cz/dataset/TPCH.

Table 2 Statistics of datasets (#-Col: #-Column; #-Num: #-Numeric; #-Cat: #-Categorical; #-Text: #-Textual; #-Date: #-Date)

Dataset	Table	#-Tuples	#-Col	#-Num	#-Cat	#-Text	#-Date
Car	**Car**	248419	14	4	10	0	0
Publication	**Author**	6881	3	0	3	0	0
	Institution	1453	4	1	3	0	0
	Paper	1947	3	0	2	1	0
	Conference	16	4	1	3	0	0
	Paper_Author	8615	2	0	2	0	0
	Paper_Keyword	2322	2	0	2	0	0
TPCH	**lineitem**	4423659	16	4	8	1	3
	orders	1500000	9	1	6	1	1
	partsupp	800000	5	2	2	1	0
	customer	150000	8	1	4	3	0
	part	200000	9	2	6	1	0
	supplier	10000	7	1	3	3	0
	nation	25	4	0	3	1	0
	region	5	3	0	2	1	0

denotes how many distinct values of a categorical or textual column has, or the range of a numeric or date column. All columns in Car and Publication are listed in Table 2, and for TPCH, only the columns involved in the tested SQL queries are reported in Table 2 due to space limitation (please refer to http://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp for more details).

The ground Truth SQL queries We use SQL queries as the proxy for user's query intent, because providing other types of ground truth is subjective and thus not feasible. Generally speaking, an SQL query for an IDE problem can be considered as a combination of an IDE-Decision and an IDE-Ranking problem, which is specified by the **SELECT**, **WHERE**, followed by the **ORDER BY** clause.

The **decision** cases of the tested SQL queries are shown in Fig. 4(a): D_1–D_8 are for dataset Car, D_9–D_{11} are for dataset Publication, and D_{12}–D_{15} are for dataset TPCH. These cases are designed to be representative and cover many different cases. More specifically, we designed 4 types of queries: AND (only *and* conjunction is included in the selective condition), OR (only *or* disjunction), NO (only *not* exclusion) and MIXED (a mix of *and, or, not*). In addition, Fig. 4(a) also shows the size of the result of the query D_i in the column "$|D|$", and its selectivity in the column "%" (e.g., $|D_1|/|\texttt{Car}| = 5.77\%$). The queries which contain square brackets are query templates, and by varying the values in the brackets, we can get more queries with different parameters. In experiments, we synthesize 10 queries with different parameters for each query template, and report the average experiment results of the 10 queries for each query template.

The **ranking** cases of the tested SQL queries are shown in Fig. 4(b): R_1–R_4 are for dataset Car, R_5, R_6 are for dataset Publication, and R_7–R_{10} are for dataset TPCH. They are categorized into 4 types: Hierarchical Sorting (sorted by multiple attributes hierarchically), Weighted Sum Sorting (sorted by a weighted sum score), Hierarchical Weighted Sorting (sorted by combining the above two types), and Nonlinear Sorting (sorted by a nonlinear combination of multiple attributes).

The 15 tested SQL queries, Q_1–Q_{15}, for the **IDE** problem are shown in Fig. 4(c), which combine the cases in Figs. 4(a) and 4(b). Here, the D and R denote the cases in Figs. 4(a) and 4(b), respectively. For example, Q_1 is obtained by combining D_1 for decision and R_1 for ranking. More natural language descriptions about the SQL queries can be found in https://github.com/Qinxuedi/dexplorer/blob/master/SQL_desc.pdf.

Note that, in each iteration, the selected tuples are labeled by a simulated user using the ground truth.

Environment All experiments are conducted on a MacBook Pro with 16 GB 1600 MHz RAM and 2.5 GHz Intel Core i7 CPU, running OS X Version 10.14.5.

Parameters We include tuples ($k = 10$) in one question for each interaction with the user. We will ask 20 iterations for all datasets and show the performance during this period. To bootstrap, we allow the user to input one keyword, e.g.,the user can input "bmw" for Q_1. Moreover, we set $\gamma = 1, \alpha' = 1$ and $\beta' = 2$ in Eq. 1 to make decision and ranking, and uncertainty and diversity equivalently important. And for the function $hasHighSimilarity(t, S, \delta)$, we set $\delta = 0.6$. For random forest model, we use the RandomForestClassifier of sklearn,[7] where we set the $n\_estimators$ (i.e.,the number of trees) as 10, $bootstrap = False$ (i.e.,all

[7] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

Table 3 Statistics of columns of different datasets

Dataset	Table	Column	Type	Range
Car	**Car**	seller	cat	2
		offerType	cat	2
		price	num	[1, 745000]
		abtest	cat	2
		vehicleType	cat	8
		year	num	[1910, 2018]
		gearbox	cat	2
		powerPS	num	[1, 20000]
		model	cat	250
		kilometer	num	[5000, 150000]
		fuelType	cat	7
		brand	cat	39
		notRepairedDamage	cat	2
		postalCode	cat	8072
Publication	**Author**	id	cat	6881
		name	cat	5623
		institution_id	cat	1451
	Institution	id	cat	1453
		name	cat	1453
		country	cat	34
		rank	num	[1, 2139]
	Paper	id	cat	1947
		conference_id	cat	12
		title	text	1947
	Conference	id	cat	16
		name	cat	4
		year	num	[2013, 2018]
		country	cat	12
	Paper_Author	paper_id	cat	1929
		author_id	cat	6871
	Paper_Keyword	paper_id	cat	1775
		keyword	cat	1657
TPCH	linelitem	l_orderkey	cat	6000000
		l_dicsount	num	[0, 0.1]
		l_extendedprice	num	[902, 104849.5]
		l_suppkey	cat	10000
		l_quantity	num	[1, 50]
		l_partkey	cat	200000
		l_commitdate	date	[1993-08-26, 1998-10-30]
		l_shipmode	cat	7
		l_shipinstruct	cat	4
	orders	o_orderdate	date	[1992-01-01, 1998-08-01]
		o_orderkey	cat	1500000
		o_custkey	cat	100000
		o_orderpriority	cat	5
		o_totalprice	num	[910.17, 528557.31]

Table 3 continued

Dataset	Table	Column	Type	Range
	partsupp	ps_partkey	cat	200000
		ps_suppkey	cat	10000
		ps_supplycost	num	[1, 1000]
	customer	c_custkey	cat	150000
		c_nationkey	cat	25
	part	p_partkey	cat	200000
		p_type	cat	150
		p_size	num	[1, 50]
		p_brand	cat	25
		p_name	cat	199998
		p_container	cat	40
	supplier	s_suppkey	cat	10000
		s_nationkey	cat	25
		s_name	cat	10000
		s_acctbal	num	[-999.61, 9999.08]
	nation	n_nationkey	cat	25
		n_regionkey	cat	5
	region	r_regionkey	cat	5
		r_name	cat	5

labeled data is used to build each tree), and the other parameters are default settings; For LambdaMART model, we use the RankLib library,[8] where we set the number of trees as 10, max depth of tree as 6, and learning rate as 0.1; For Ranking SVM model, we use the SVM Rank library,[9] where we set the trade-off between training error and margin as 0.01, the slack variable as L1-norm, and the kernel function as linear.

7.1 IDE Problem

7.1.1 Effectiveness

Metrics The Kendall tau distance [13] is a widely used metric for quantifying two ranking lists [15,16,22], with the basic idea of counting the number of pairwise disagreements between two ranking lists. However, in our case, we do not treat the result as a single ranked list. Instead, we consider two parts: $Q(T)$ as the ground truth desired and ranked tuples, and $Q'(T) = T \backslash Q(T)$ as the undesired tuples. Given any two tuples $t_i, t_j \in T$, the ground truth partial order between them is either $t_i \succ t_j$ if (1) both t_i and t_j are in $Q(T)$ but t_i is ranked higher than t_j, or (2) $t_i \in Q(T)$ but $t_j \in Q'(T)$; or $t_i \equiv t_j$ when $t_i, t_j \in Q'(T)$.

Let R be the desired and ranked tuples predicted by an algorithm, and $R' = T \backslash R$. Given any tuple pair (t_i, t_j), based on R and R', it will predict a label, which is either $t_i \succ t_j$ or $t_i \equiv t_j$, similar to the above process.

Let $N = \binom{|T|}{2}$ be the total number of tuple pairs. Let KD be the "disagreement" of all tuple pairs, i.e.,the total number of tuple pairs whose predicted labels are different from ground truth labels. We use the normalized Kendall tau distance as the evaluation metric, denoted by $accuracy = \frac{N-KD}{N}$.

Example 10 Suppose $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$. The ground truth is $Q(T) = \{t_1, t_2, t_3\}$ that is ranked as $t_1 \succ t_2 \succ t_3$, and $Q'(T) = \{t_4, t_5, t_6\}$ (i.e.,$t_4 \equiv t_5 \equiv t_6$ for their ranking). The predicted result is $R = \{t_1, t_2, t_4\}$ that is ranked as $t_1 \succ t_4 \succ t_2$, and $R' = \{t_3, t_5, t_6\}$. The total number of tuple pairs is $N = \binom{6}{2} = 15$. There are $KD = 6$ mistakenly predicted tuple pairs: $\{t_4 \succ t_2, t_4 \succ t_6, t_3 \equiv t_5, t_3 \equiv t_6, t_4 \succ t_5, t_4 \succ t_6\}$. Therefore, $accuracy = \frac{15-6}{15} = 0.6$.

Besides using *accuracy* to capture the global ranking quality, the users might also be interested in the top-l results. Thus, we also used $precision@l = \frac{|R[1:l] \cap Q(T)[1:l]|}{l}$, where $R[1:l]$ and $Q(T)[1:l]$ denote the set of top-l tuples in R and $Q(T)$, respectively.

Comparisons We tested (1) DExPlorer-IQS+ using the model in Section 4.2 for answer inference and IQS+ for question selection; (2) DExPlorer-IQS and (3) DExPlorer-

[8] https://sourceforge.net/p/lemur/wiki/RankLib/.

[9] http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html.

D	Dataset	Type	SQL	$\|D\|$	%
D_1	Car	AND	**SELECT** * **FROM** Car **WHERE** brand = "bmw" **AND** price ≤ 10000 **AND** gearbox = "manually"	14325	5.77
D_2	Car	AND	**SELECT** * **FROM** Car **WHERE** seller = "private" **AND** offerType = "offer" **AND** price ≤ 100000 **AND** abtest = "control" **AND** vehicleType = "limousine" **AND** year ≥ 2000 **AND** gearbox = "manually" **AND** powerPS ≥ 150 **AND** kilometer ≤ 150000 **AND** fuelType = "petrol" **AND** brand = "volkswagen" **AND** notRepairedDamage = "no"	545	0.22
D_3	Car	OR	**SELECT** * **FROM** Car **WHERE** brand = "bmw" **OR** brand = "audi"	53288	21.45
D_4	Car	OR	**SELECT** * **FROM** Car **WHERE** price ≤ 100 **OR** abtest = "test" **OR** vehicleType = "suv" **OR** year = 2018 **OR** powerPS ≥ 1500 **OR** model = "verso" **OR** kilometer ≤ 5000 **OR** brand = "volvo"	136984	55.14
D_5	Car	NO	**SELECT** * **FROM** Car **WHERE** brand ≠ "bmw"	219101	88.2
D_6	Car	MIXED	**SELECT** * **FROM** Car **WHERE** (brand = "bmw" **AND** price ≤ 10000) **OR** brand = "volkswagen"	71315	28.71
D_7	Car	MIXED	**SELECT** * **FROM** Car **WHERE** fuelType = "petrol" **AND** year ≥ 2010 **AND** gearbox = "manually" **AND** ((brand = "bmw" **AND** price ≤ 10000) **OR** (brand = "volkswagen" **AND** price ≤ 8000))	596	0.24
D_8	Car	MIXED	**SELECT** * **FROM** Car **WHERE** fuelType != "petrol" **AND** year ≥ 2010 **AND** gearbox = "manually" **AND** ((brand = "bmw" **AND** price ≤ 10000) **OR** (brand = "volkswagen" **AND** price ≤ 8000))	3541	1.43
D_9	Publication	AND	**SELECT** Paper.title, Conference.name, Conference.year, Institution.country, Institution.rank **FROM** Paper, Paper_Author, Author, Institution, Conference **WHERE** Paper.id = Paper_Author.paper_id **AND** Paper_Author.author_id = Author.id **AND** Author.institution_id = Institution.id **AND** Paper.conference_id = Conference.id **AND** Conference.name = "SIGMOD" **AND** Conference.year = 2018 **AND** Institution.country = "USA" **AND** Institution.rank ≤ 10	157	1.82
D_{10}	Publication	MIXED	**SELECT** Paper.title, Conference.name, Conference.year, Paper_Keyword.keyword, Institution.rank **FROM** Paper, Paper_Author, Author, Institution, Conference, Paper_Keyword **WHERE** Paper.id = Paper_Author.paper_id **AND** Paper_Author.author_id = Author.id **AND** Author.institution_id = Institution.id **AND** Paper.conference_id = Conference.id **AND** Paper.id = Paper_Keyword.paper_id **AND** (Paper_Keyword.keyword = "data visualization" **OR** (Conference.name = "SIGMOD" **AND** Conference.year = 2018))	3978	25.28
D_{11}	Publication	OR	**SELECT** * **FROM** Author, Institution, Paper, Paper_Author, Conference **WHERE** Paper.id = Paper_Author.paper_id **AND** Paper_Author.author_id = Author.id **AND** Author.institution_id = Institution.id **AND** Paper.conference_id = Conference.id **AND** (Paper.title **LIKE** "%machine learning%" **OR** Institution.country in ("[COUNTRY1]","[COUNTRY2]") **OR** Institution.name in ("[INSTITUTION1]", "[INSTITUTION2]", "[INSTITUTION3]"))	-	-
D_{12}	TPCH	AND	**SELECT** * **FROM** customer, orders, lineitem, supplier, nation, region **WHERE** c_custkey = o_custkey **AND** l_orderkey = o_orderkey **AND** l_suppkey = s_suppkey **AND** c_nationkey = s_nationkey **AND** s_nationkey = n_nationkey **AND** n_regionkey = r_regionkey **AND** r_name = "[REGION1]" **AND** o_orderdate ≥ "[DATE1]" **AND** o_orderdate ≤ "[DATE2]" **AND** l_quantitiy ≥ [QUANTITY1] **AND** l_quantity ≤ [QUANTITY2] **AND** o_orderpriority = [ORDERPRIORITY1] **AND** l_shipinstruct = "[SHIPINSTRUCT1]" **AND** o_totalprice ≥ [TOTALPRICE1] **AND** o_totalprice ≤ [TOTALPRICE2] **AND** s_acctbal ≥ [ACCTBAL1]	-	-
D_{13}	TPCH	OR	**SELECT** * **FROM** part, supplier, partsupp, nation, region **WHERE** p_partkey = ps_partkey **AND** s_suppkey = ps_suppkey **AND** s_nationkey = n_nationkey **AND** n_regionkey = r_regionkey **AND** (p_size ≥ [SIZE1] **OR** p_size ≤ [SIZE2] **OR** p_type = "[TYPE1]" **OR** p_name = "[PART1]" **OR** r_name = "[REGION1]" **OR** s_name = "[SUPPLIER1]" **OR** ps_supplycost ≥ [SUPPLYCOST1] **OR** ps_supplycost ≤ [SUPPLYCOST2] **OR** p_container = "[CONTAINER1]" **OR** p_brand = "[BRAND1]")	-	-
D_{14}	TPCH	MIXED	**SELECT** * **FROM** lineitem, part **WHERE** p_partkey = l_partkey **AND** p_brand = "[BRAND1]" **AND** p_container in ("[CONTAINER1]", "[CONTAINER2]", "[CONTAINER3]", "[CONTAINER4]") **AND** l_quantity ≥ [QUANTITY1] **AND** l_quantity ≤ [QUANTITY1] + 10 **AND** l_shipinstruct = "[SHIPINSTRUCT1]" **AND** p_size ≥ [SIZE1] **AND** p_size ≤ [SIZE2] **AND** l_shipmode in ("[SHIPMODE1]", "[SHIPMODE2]")	-	-
D_{15}	TPCH	MIXED	**SELECT** * **FROM** lineitem, part **WHERE** (p_partkey = l_partkey **AND** p_brand = "[BRAND1]" **AND** p_container in ("[CONTAINER1]", "[CONTAINER2]", "[CONTAINER3]", "[CONTAINER4]") **AND** l_quantity ≥ [QUANTITY1] **AND** l_quantity ≤ [QUANTITY1] + 10 **AND** l_shipinstruct = "[SHIPINSTRUCT1]" **AND** p_size ≥ [SIZE1] **AND** p_size ≤ [SIZE2] **AND** l_shipmode in ("[SHIPMODE1]", "[SHIPMODE2]")) **OR** (p_partkey = l_partkey **AND** p_brand = "[BRAND2]" **AND** p_container in ("[CONTAINER5]", "[CONTAINER6]", "[CONTAINER7]", "[CONTAINER8]") **AND** l_quantity ≥ [QUANTITY2] **AND** l_quantity ≤ [QUANTITY2] + 10 **AND** l_shipinstruct = "[SHIPINSTRUCT2]" **AND** p_size ≥ [SIZE3] **AND** p_size ≤ [SIZE4] **AND** l_shipmode in ("[SHIPMODE3]", "[SHIPMODE4]")) **OR** (p_partkey = l_partkey **AND** p_brand = "[BRAND3]" **AND** p_container in ("[CONTAINER9]", "[CONTAINER10]", "[CONTAINER11]", "[CONTAINER12]") **AND** l_quantity ≥ [QUANTITY3] **AND** l_quantity ≤ [QUANTITY3] + 10 **AND** l_shipinstruct = "[SHIPINSTRUCT3]" **AND** p_size ≥ [SIZE5] **AND** p_size ≤ [SIZE6] **AND** l_shipmode in ("[SHIPMODE5]", "[SHIPMODE6]"))	-	-

(a) Tested IDE-Decision Problems

R	Dataset	Type	SQL
R_1	Car	Hierarchical Sorting	**ORDER BY** year **DESC**, -kilometer **DESC**, -price **DESC**, powerPS **DESC**
R_2	Car	Weighted Sum Sorting	**ORDER BY** -0.018 ∗ price + 0.06 ∗ year + 0.982 ∗ powerPS - 0.001 ∗ kilometer **DESC**
R_3	Car	Hierarchical Weighted Sorting	**ORDER BY** notRepairedDamage **DESC**, vehicleType **DESC**, fuelType **DESC**, brand **DESC**, -0.018 ∗ price + 0.982 ∗ powerPS **DESC**
R_4	Car	Hierarchical Weighted Sorting	**ORDER BY** year **DESC**, -kilometer **DESC**, -0.018 ∗ price + 0.982 ∗ powerPS **DESC**
R_5	Publication	Weighted Sum Sorting	**ORDER BY** -0.8 ∗ rank + 0.2 ∗ year **DESC**
R_6	Publication	Hierarchical Sorting	**ORDER BY** -rank **DESC**, year **DESC**
R_7	TPCH	Hierarchical Sorting	**ORDER BY** l_extendedprice **DESC**, l_quantity **DESC**, l_commitdate **DESC**
R_8	TPCH	Hierarchical Sorting	**ORDER BY** s_acctbal **DESC**, -ps_supplycost **DESC**
R_9	TPCH	Non-linear Sorting	**ORDER BY** l_extendedprice ∗ l_discount **DESC**
R_{10}	TPCH	Non-linear Sorting	**ORDER BY** l_extendedprice ∗ (1 - l_discount) **DESC**

(b) Tested IDE-Ranking Problems

Q	Dataset	D	R	Q	Dataset	D	R	Q	Dataset	D	R	Q	Dataset	D	R	Q	Dataset	D	R
Q_1	Car	D_1	R_1	Q_2	Car	D_2	R_2	Q_3	Car	D_3	R_3	Q_4	Car	D_4	R_4	Q_5	Car	D_5	R_2
Q_6	Car	D_6	R_1	Q_7	Car	D_7	R_3	Q_8	Publication	D_8	R_4	Q_9	Car	D_9	R_5	Q_{10}	Publication	D_{10}	R_5
Q_{11}	Publication	D_{11}	R_6	Q_{12}	TPCH	D_{12}	R_7	Q_{13}	TPCH	D_{13}	R_8	Q_{14}	TPCH	D_{14}	R_9	Q_{15}	TPCH	D_{15}	R_{10}

(c) Tested IDE Problems

Fig. 4 Tested IDE-Decision, IDE-Ranking, IDE Problems

AQS use the same answer inference model with DExPLORER-IQS+ but different question selection algorithms: IQS and AQS, respectively. We also compared with (4) SQLSynthesizer [75] (denoted by SSY), which infers SQL by accepting ranked positive tuples, and (5) learning-to-rank (*i.e.,* LTR)-based solutions as described in Section 4.2. We use 3 LTR methods: Ranking SVM [56] (denoted by LTR1), LambdaMART [71] (denoted by LTR2) and Gaussian model [10] (denoted by LTR3). For answer inference, LTR1, LTR2, and LTR3 use the ranking labels to train the Ranking SVM model, LambdaMART model, Gaussian model, respectively, obtain a score for each tuple t and rank based on the scores. Then we learn a threshold θ to separate the desired and undesired tuples, which results in the maximum information gain; for question selection, we select tuples by the binary search strategy in [56] for LTR1; we randomly select k tuples for LTR2; and we select tuples with the highest entropy gains for LTR3 as described in [10].

Exp-1: Accuracy of IDE: Figure 5 shows the *accuracy* of the IDE problems $Q_1 - Q_{15}$ for DExPLORER-IQS+, DExPLORER-IQS, DExPLORER-AQS, SSY, LTR1, LTR2 and LTR3. Figure 5 tells us that:

(1) With the increasing of #-Questions, the accuracy of DExPLORER-IQS+, DExPLORER-IQS, DExPLORER-AQS and SSY all increase, as expected. They finally achieve *accuracy* of 0.978, 0.901, 0.965, 0.603, respectively, on average of Q_1–Q_{15} after asking 20 questions. DExPLORER-IQS+ behaves similarly to DExPLORER-AQS, and even better in some cases (i.e.,Q_7, Q_8, Q_{11}). However, as shown in Section 7.1.2, DExPLORER-AQS takes longer time (i.e.,more than 10 minutes for $Q_1 - Q_8$, $Q_{12} - Q_{15}$), while DExPLORER-IQS+ can return results in interactive time because our question selection algorithm, IQS+, is efficient. DExPLORER-IQS+ improves the final accuracy by 8.5% compared with DExPLORER-IQS, and converges faster, which proves that IQS+ is a more effective question selection algorithm compared with IQS, and both IQS+ and IQS are efficient (Section 7.1.2).

(2) SSY supports decision using a decision tree and supports ranking by one or more attributes hierarchically. More specifically, it only supports the ranking type like R_1, R_6, R_7, R_8 in Fig. 4(b), so SSY can only capture the **ORDER BY** clause of Q_1, Q_6, Q_{11}, Q_{12}, Q_{13}. Therefore, for Q_1 and Q_6, SSY learns the target SQL query, ranks the desired tuples correctly, and achieves an *accuracy* of 1.0. For Q_{11}, Q_{12}, Q_{13}, SSY can easily learns the ranking clauses, but SSY can not learn the decision clauses quickly, resulting in the low *accuracy*. However, for Q_3, Q_5, Q_9, Q_{10}, their corresponding IDE-Decision problems are inferred by SSY correctly, but the IDE-Rankingproblems are not learned. Thus, the *accuracy* of SSY is mainly decided by the size of the query results. The number of query result of Q_3, Q_9, Q_{10} are small, thus many tuple pairs which are not in the query result were correctly predicted, and the *accuracy* is high for Q_3, Q_9, Q_{10}. For Q_5, there are many tuples in the query result, but SSY does not capture the ranking information of tuples in the query result, thus the tuple pairs in the query result are all wrongly predicted, and thus the *accuracy* is low. For Q_2, Q_4, Q_7, Q_8, their corresponding IDE-Decision problems are so complex that the decision tree fails to capture the SQL. Thus, their accuracy is not good. Based on the above analysis, we can conclude that SSY can only capture simple SQL queries for both decision and ranking, and the accuracy highly depends on the size of the query results, while DExPLORER has a stable performance for all cases.

(3) LTR-based methods (*i.e.,* LTR1, LTR2, LTR3) have low accuracies even with the number of questions increasing: the final average accuracies achieved by LTR1, LTR2, LTR3 are 0.147, 0.184, 0.099 respectively, which are much less than that of DExPLORER (0.978). Even if they both build models to learn the ranking, DExPLORER performs much better than LTR because our system builds customized IDE-Decision model to discover desired tuples while LTR simply learns a threshold to do that. Therefore, we conclude that the IDE problem cannot be solved perfectly by a single model, i.e., the decision and ranking based models should work together to achieve a good performance. Besides, LTR-based methods do not have carefully designed question selection algorithm for our IDE problem, which is of great importance to our IDE problem. So LTR-based methods are not suitable for our problem.

(4) The more complex the query is, the more questions we need to answer to reach a certain *accuracy*. For example, $Q_6 - Q_8$ and $Q_{12} - Q_{15}$ are more complex than $Q_1 - Q_5$, so the *accuracy* of $Q_6 - Q_8$, $Q_{12} - Q_{15}$ increases slower than $Q_1 - Q_5$ as shown in Fig. 5. More specifically, for Q_3 (Fig. 5(c)) and Q_8 (Fig. 5(h)), when we ask five questions using DExPLORER, the *accuracy* of Q_3 is 0.88 while the *accuracy* of Q_8 is only 0.61.

(5) The three versions of DExPLORER (IQS+, AQS, IQS) all aim to optimize Eq. 2, and thus they have the similar trend, i.e.,with the number of labeled tuples increasing, more examples will be used to train the model, so the effectiveness of IQS+, AQS, IQS improves. If the labeled data are sufficient,

Table 4 The reason of contributing to the final accuracy

Real	Predict	Reason
$t_i \succ t_j$	$t_i \in Q(T), t_j \in Q(T), t_i \succ t_j$	Ranking
	$t_i \in Q(T), t_j \in Q'(T)$	Decision
$t_j \succ t_i$	$t_i \in Q(T), t_j \in Q(T), t_j \succ t_i$	Ranking
	$t_i \in Q'(T), t_j \in Q(T)$	Decision
$t_i \equiv t_j$	$t_i \in Q'(T), t_j \in Q'(T)$	Decision

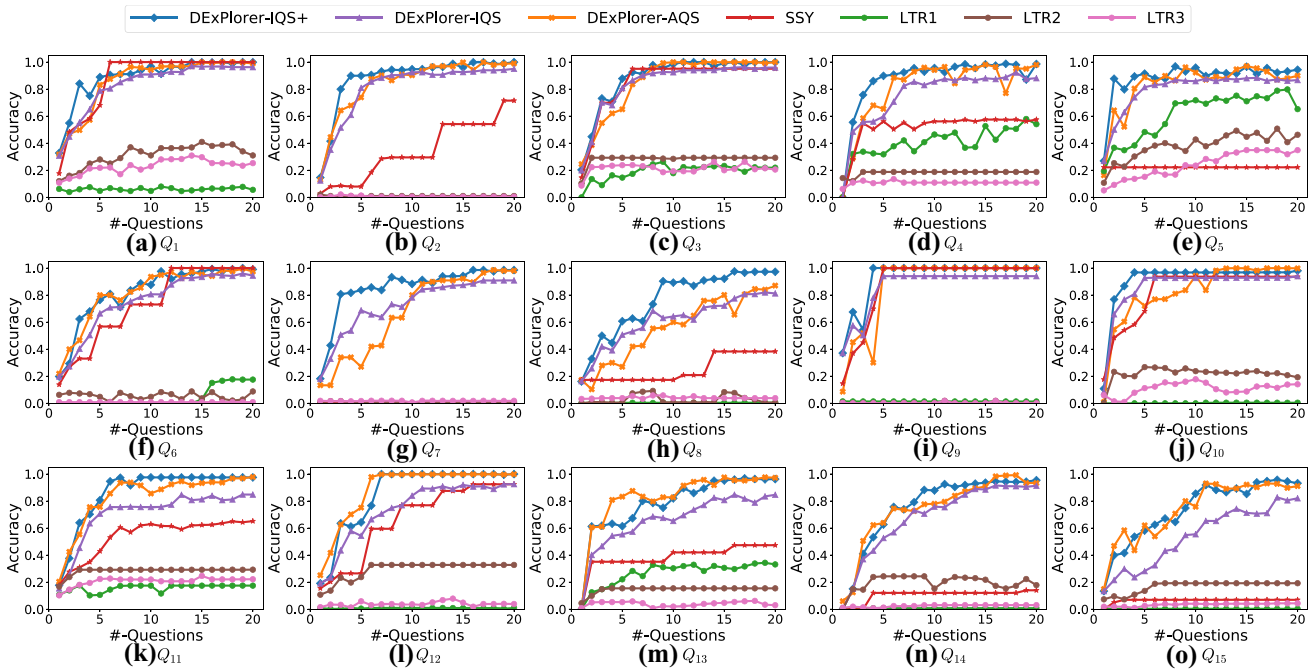


Fig. 5 Effectiveness study for **IDE** problem (x-axis: #-Questions; y-axis: Accuracy)

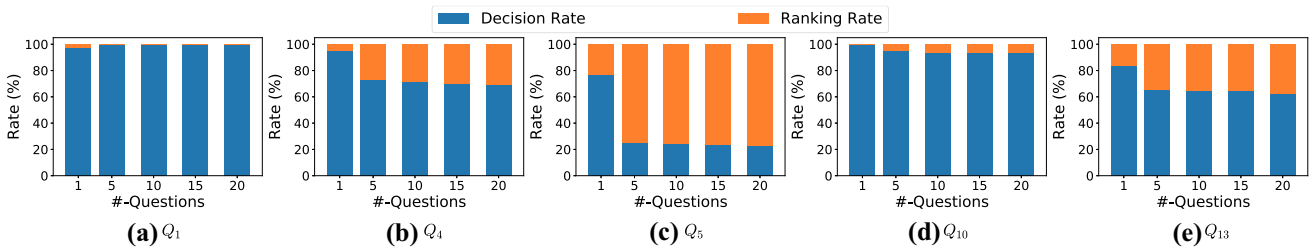


Fig. 6 Decision rate and ranking rate for **IDE** Problem

Table 5 Treating IDE-Decision and IDE-Rankingholistically versus separately using our proposed models

Dataset		Car								Publication			TPCH			
Query		Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_{10}	Q_{11}	Q_{12}	Q_{13}	Q_{14}	Q_{15}
DExPlorer	#-Click	120	80	120	120	90	140	140	120	40	40	90	70	140	130	160
	#-Drag	32	6	43	56	63	49	17	24	7	9	12	16	15	34	38
	Accuracy	0.97	0.94	0.99	0.96	0.94	0.96	0.94	0.93	0.99	0.97	0.98	0.99	0.95	0.92	0.94
DExPlorer-SEP	#-Click	120	80	120	120	90	140	140	120	40	40	90	70	140	130	160
	#-Drag	35	14	45	62	70	56	23	28	12	12	12	18	17	35	40
	Accuracy	0.88	0.84	0.80	0.74	0.71	0.79	0.69	0.71	0.88	0.83	0.92	0.87	0.73	0.75	0.67

the final effectiveness of IQS+, AQS, IQS should converge to an almost same *accuracy*, such as Q_1, Q_2, Q_3. However, as discussed above, IQS+ can optimize more components in Eq. 2 than IQS, so it will converge faster. For example, for Q_4, IQS+ achieves 0.96 *accuracy* when 10 question are answered, and IQS achieves 0.85 *accuracy* when 10 question are answered. In addition, for AQS and IQS+, we mainly focus on that IQS+ can conduct question selection in interactive time (∼1 second) while AQS cannot (Section 7.1.2).

(6) To better compare the three versions of DExPlorer from a big picture, we report the average number of questions answered for convergence of Q_1–Q_{15} and the average *accuracy* at convergence of Q_1–Q_{15}. The average numbers of questions answered for convergence achieved by

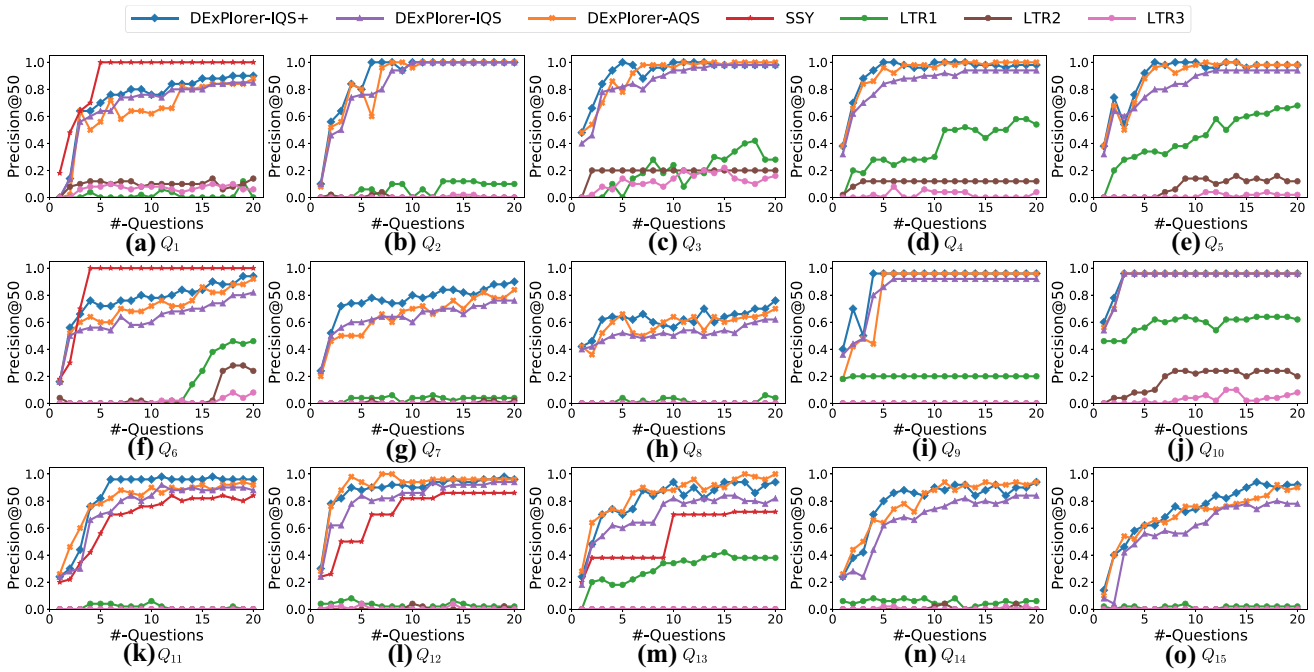


Fig. 7 Effectiveness study for **IDE** problem (x-axis: #-Questions; y-axis: Precision@50)

IQS+, AQS, IQS are 11, 11, 12, respectively. The average accuracies at convergence of IQS+, AQS, IQS are 0.954, 0.949, 0.882, respectively. As discussed in (5), IQS+ optimizes more components in Eq. 2 than IQS, so IQS+ converges faster than IQS (11 questions vs. 12 questions) and has higher *accuracy* at convergence than IQS (0.954 vs. 0.882). AQS also converges fast and has high *accuracy* at convergence, but AQS has a long question selection time. Each question selection of AQS takes more than 10 minutes for dataset Car and TPCH (Section 7.1.2).

Exp-2: Decision Rate and Ranking Rate Figure 5 shows the overall *accuracy* which combines the score of IDE-Decision and IDE-Rankingall together. But when checking the *accuracy* results, we may not understand which part of the *accuracy* score is due to the tuple selection performance of the method, and which part is due to the tuple ranking performance. So we report the decision rate (i.e., tuple selection rate) and ranking rate (i.e., tuple ranking rate) of current accuracy results in this experiment.

To calculate the *accuracy*, we are actually counting the number of correctly predicted tuple pairs. For a tuple pair (t_i, t_j), there are three cases of their real relations: $t_i \succ t_j, t_j \succ t_i$ and $t_i \equiv t_j$. If the predicted relation of (t_i, t_j) is the same as their real relation, this pair will contribute to the final *accuracy*, i.e.,it will count 1 among all correctly predicted pairs. Table 4 shows the reason that a tuple pair will contribute to the final *accuracy* (i.e.,count 1 among all correctly predicted pairs). For example, consider the first row in Table 4, for a tuple pair (t_i, t_j), their real relation (i.e.,ground truth) is $t_i \succ t_j$, and the prediction result is $t_i \in Q(T), t_j \in Q(T), t_i \succ t_j$. This pair is predicted correctly because of the ranking model (i.e.,the tuple ranking performance). And the pair of the second row is predicted correctly because of the decision model (i.e.,the tuple selection performance).

We report the decision rate (i.e.,the rate of pairs which are correctly predicted due to the decision model among all correctly predicted pairs) and ranking rate (i.e.,the rate of pairs which are correctly predicted due to the ranking model among all correctly predicted pairs) in Fig. 6. We report the evaluation results of five representative queries (Q_1, Q_4, Q_5, Q_{10}, Q_{13}). From Fig. 6, we can know that:

(1) With the increase of the number of questions answered, the decision rate gradually decreases (the ranking rate gradually increases correspondingly), and finally becomes stable. The decision performance is inaccurate at first, and the ranking function is applied to the tuples that are predicted *true*, so the decision rate is high at first. But with the increase of #-Questions, the decision model is getting accurate, so the ranking model can play a role.

(2) The final decision rate and ranking rate are highly correlated with the selectivity of the IDE-Decision problem (i.e.,the column "%" in Fig. 4(a)). Basically speaking, the higher the selectivity is, the higher the final ranking rate is. For example, the selectivities of Q_1, Q_4, Q_5, Q_{10} are 5.77%, 55.14%, 88.2%, 25.28%, respectively, and the final ranking rates of Q_1, Q_4, Q_5, Q_{10} are 0.37%, 30.92%, 77.21%, 6.20%, respectively. When the selectivity is high, there are

Table 6 Efficiency study for **IDE** Problem (AI: Answer Inference; QS: Question Selection; sec: second; 10 m: 10 minutes)

Dataset		Car								Publication		TPCH				
Query		Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_{10}	Q_{11}	Q_{12}	Q_{13}	Q_{14}	Q_{15}
IQS+	AI (sec)	0.98	1.01	0.88	1.14	1.13	1.00	1.03	1.31	0.03	0.08	0.17	0.83	0.95	0.87	1.04
	QS (sec)	0.61	0.73	0.58	0.62	0.60	0.60	0.87	0.93	0.02	0.05	0.04	0.52	0.72	0.83	0.91
IQS	AI (sec)	1.02	0.97	0.98	1.02	1.12	1.14	1.19	1.26	0.04	0.08	0.19	0.80	0.88	0.90	0.97
	QS (sec)	0.59	0.63	0.60	0.65	0.55	0.60	0.86	0.98	0.02	0.04	0.04	0.48	0.67	0.88	0.89
AQS	AI (sec)	1.03	1.04	0.88	1.01	1.05	1.12	1.18	1.20	0.05	0.08	0.17	0.88	0.90	0.88	0.95
	QS (sec)	>10m	>10m	>10m	>10m	>10m	>10m	>10m	>10m	6	19	6	>10m	>10m	>10m	>10m

more tuples that need to be ranked, so the final ranking rate is high.

Exp-3: Precision@l of IDE: Figure 7 shows the *precision@50* of the IDE problems $Q_1 - Q_{15}$ by DExPLORER-IQS+, DExPLORER-IQS, DExPLORER-AQS, SSY, LTR1, LTR2 and LTR3 respectively. Since SSY only returns **ORDER BY** clauses for $Q_1, Q_6, Q_{11}, Q_{12}, Q_{13}$, we only report *precision@50* of SSY for $Q_1, Q_6, Q_{11}, Q_{12}, Q_{13}$. Figure 7 tells us that: DExPLORER-IQS+ and DExPLORER-AQS perform similarly, and they finally achieve 0.939 and 0.931 *precision@50* on average after asking 20 questions, while LTR-based solutions have poor performance for finding the top-l tuples. SSY can only capture simple ranking clauses of $Q_1, Q_6, Q_{11}, Q_{12}, Q_{13}$, so it only has good performances in $Q_1, Q_6, Q_{11}, Q_{12}, Q_{13}$.

Exp-4: Treating IDE-Decision and IDE-Rankingseparately (denoted by DExPLORER-SEP*) versus holistically (denoted by* DExPLORER*)* in each round of user interaction. We show the number of user operations in Table 5 (i.e.,the number of "click" operation and "drag" operation) and *accuracy* of DExPLORER when it gets converge (i.e.,the *accuracy* difference of two consecutive rounds is less than 1%). For DExPLORER-SEP, we show users k ($k = 10$) tuples in each round. In the first few rounds, users only perform click operations, and in the later rounds, users only perform drag operations on the positive tuples inferred by the first few rounds. The question selection algorithm of DExPLORER-SEP is shown in Section 5.3. For ease of comparison, we perform the same number of "click" operations as DExPLORER, then perform more (or same) number of "drag" operations than DExPLORER, and report the *accuracy* of DExPLORER-SEP. We can see that the *accuracy* of DExPLORER is much higher than DExPLORER-SEP: DExPLORER averagely achieves 0.96 *accuracy* on 15 problems, while DExPLORER-SEP averagely achieves 0.79 *accuracy*, that is, DExPLORER outperforms DExPLORER-SEP by 22%. This is because DExPLORER-SEP only asks questions related to IDE-Decision problem in the first few rounds, without considering the IDE-Rankingproblem, thus obtaining worse results.

7.1.2 Efficiency

Exp-5: Efficiency We also test the efficiency of DExPLORER on IDE problems $Q_1 - Q_{15}$. We repeat all experiments ten times to compute the average results. Table 6 shows the running time of answer inference and question selection by algorithms IQS+, IQS and AQS.

We make the following observations: (1) IQS+ and IQS significantly outperform the baseline—AQS on all IDE problems (i.e.,Q_1 - Q_{15}). This observation also matches the result of the time complexity discussion in Section 5. (2) The results on answer inference and question selection of the tested algo-

Table 7 User study (min: minute; sec: second)

User	#-Question	Total time (min)	Average time (sec)	Accuracy
1	5	2.9	35	0.99
2	10	9.7	58	0.91
3	15	14.0	56	0.89
4	14	14.7	63	0.90
5	9	10.1	67	0.91
6	13	9.8	45	0.92
7	6	3.9	39	0.96
8	14	12.1	52	0.92
9	11	8.8	48	0.92
10	8	6.0	45	0.95

rithms do not vary a lot for different IDE problems on the same dataset. The reason is that the algorithms are independent of the complexity of the SQL queries. However, the complex SQL queries usually take more rounds to converge. (3) It takes less than 2 seconds on datasets Car and TPCH to infer answers and select questions for the next round using IQS+, which is feasible in practice, while it takes more than 10 minutes for AQS on the two datasets. For the small dataset Publication, it only takes ∼0.1 second.

7.1.3 User study

Exp-6: User study The above experiments are conducted by a simulated user, but in real scenarios, users may make mistakes while providing feedback, so we conduct user studies to further evaluate the usability (effectiveness and efficiency) of DExPlorer.

We employ 10 university students[10] from a crowdsourcing [5–7,36] platform Appen.[11] The native languages of these workers are English and their Human Intelligence Task (HIT) approval ratings are bigger than 90%. We ask these users to use our system DExPlorer to find their ranked desired cars on the Car dataset. DExPlorer shows the user 10 tuples (i.e.,a question) in each round, and the user labels these tuples as *true/false*, and drags the positive tuples to sort them. When the user is satisfied with the inferred ranked desired result R or loses patience, the user can terminate the exploration process. We report the number of rounds (i.e.,the number of questions) and the total time that the user labels tuples when the exploration process is terminated in Table 7. Besides, we also report the *accuracy* of real user feedback when the exploration process is terminated. To get the *accuracy* of real user feedback, we need user's feedback on R, because we do not know the user's real intent (i.e.,we do not have the ground truth). To get the ground truth, we sample 50 tuples for the user to label (because it is impractical for the user to label all tuples): 25 tuples are sampled from R and 25 tuples are sampled from $T \setminus R$ (if we randomly sample 50 tuples from T, we may not sample the user's desired tuples). We ask the user to label the 50 tuples as *true/false* and drag the positive tuples to sort them. Then we can get the ground truth of the 50 tuples and calculate the final *accuracy* (the calculation method of *accuracy* is described in Section 7.1.1). The results are shown in Table 7.

From Table 7, we can know that: (1) it takes $5-15$ rounds for the 10 users to find their ranked desired tuples by DExPlorer, with an average of 10.5; (2) the time that the 10 users find their ranked desired tuples by DExPlorer varies from 2.9 minutes to 14.7 minutes, with an average of 9.2 minutes; the average time that the 10 users label tuples in each round varies from 35 seconds to 67 seconds, with an average of 50.8 seconds. (3) the *accuracy* varies from 0.89 to 0.99, with an average of 0.93. The experiment results in Table 7 show that: DExPlorer can help users find their ranked desired tuples effectively (the average *accuracy* is 0.93) in reasonable time ($<$ 15 minutes, the data exploration system AIDE [14] find users' desired tuples in $7.9-39.7$ minutes, with an average of 28.3 minutes).

7.2 Effectiveness of IDE-Decision problem

Metrics The IDE-Decision problem is actually a classification problem: the tuples in the inferred answer R of the exploration system are classified as positive, and others as negative. Thus, we use F1-score to evaluate the effectiveness of the IDE-Decision problem, which is defined as: $F1\text{-}score = \frac{2\times p\times r}{p+r}$, where $p = \frac{TP}{TP+FP}, r = \frac{TP}{TP+FN}, TP = |R \cap Q(T)|, FP = |R \cap Q'(T)|, FN = |R' \cap Q(T)|$.

Comparisons We compare (1) DExPlorer using the decision model in Section 4.2 for decision answer inference and IQS+ in Section 5.3 for question selection, (2) AIDE [14]

[10] We tell the workers that we need university students to participate in a user study, and ask them to fill in their ".edu" mails. We then send emails to these ".edu" mails with the link to the user study. Users can use DExPlorer to find their ranked desired tuples in this link.

[11] https://appen.com.

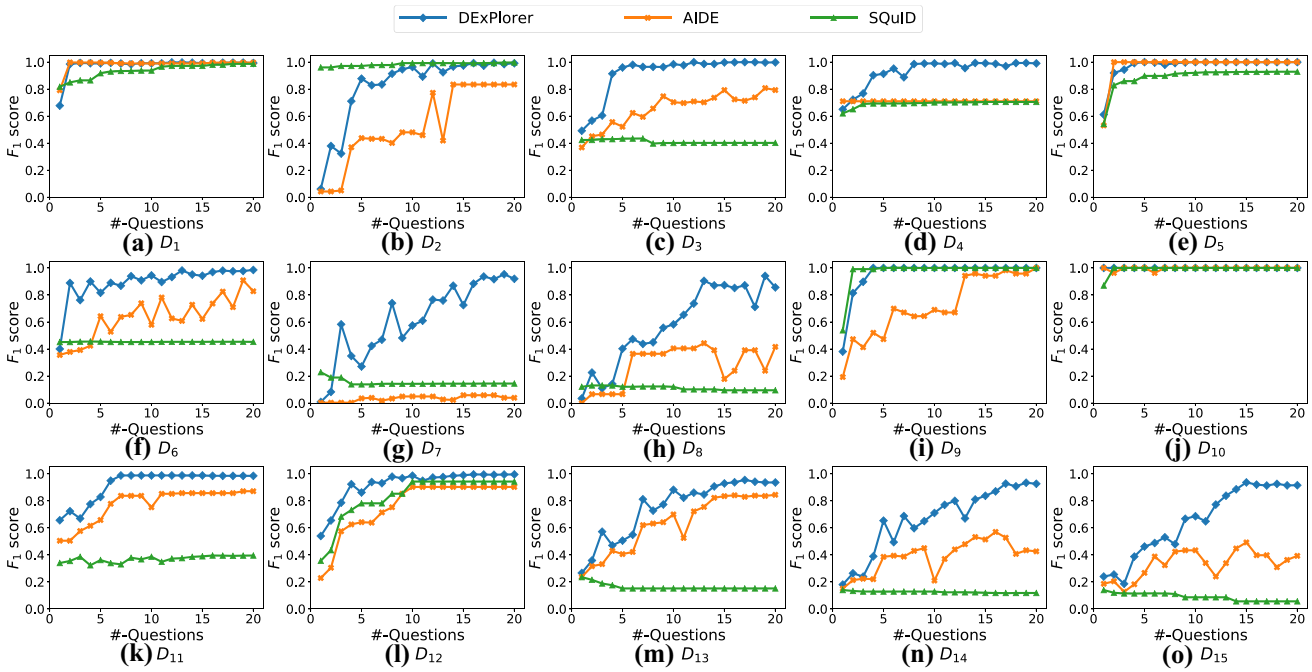


Fig. 8 Effectiveness Study for IDE-Decision Problem (x-axis: #-Questions; y-axis: F1-score)

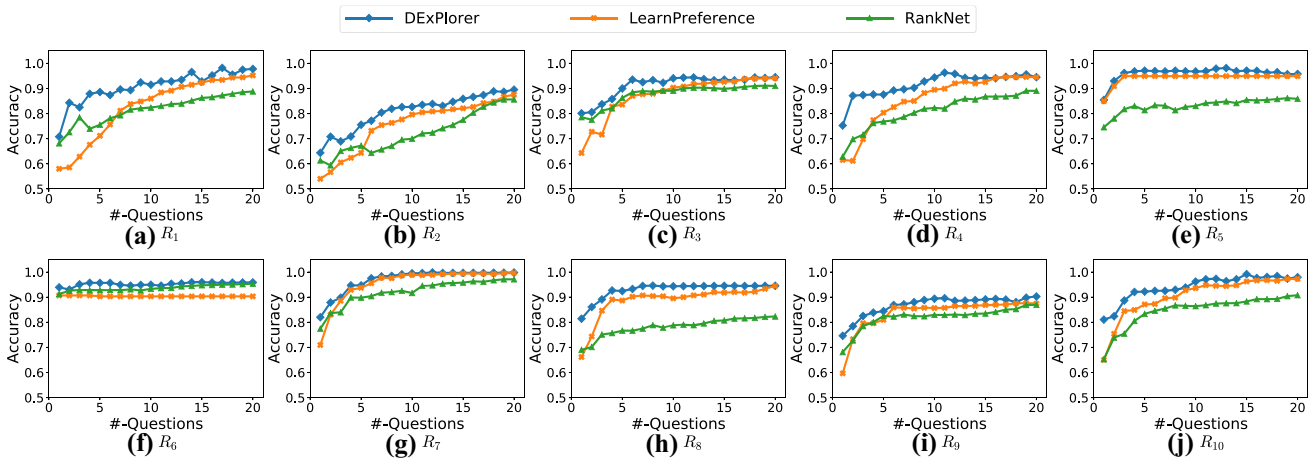


Fig. 9 Effectiveness study for IDE-Rankingproblem (x-axis: #-Questions; y-axis: Accuracy)

uses a decision tree for answer inference and selects k tuples from diverse data areas to the user for question selection, and (3) SQUID [17] is a state-of-the-art SQL query intent discovery system, which takes a set of positive tuples as input examples and outputs the inferred SQL whose query result possibly includes these tuples. We select questions by randomly choosing k positive tuples for SQUID.

Exp-7: Effectiveness on IDE-Decision: We test DEXPLORER, AIDE, SQUID on $D_1 - D_{15}$. Figure 8 shows the results and tells us the followings:

(1) DEXPLORER outperforms AIDE and SQUID, among all IDE-Decision problems. DEXPLORER averagely achieves 0.966 F1-score with 20 questions as training data, while AIDE and SQUID only averagely achieve 0.737 and 0.558 F1-score, respectively. In summary, DEXPLORER outperforms AIDE and SQUID by 31% and 73%, respectively.

(2) AIDE shows poor performance on complex queries (i.e.,D_7, D_8, D_{14}, D_{15}), as mentioned in Section 4.2, which is due to the use of decision tree. What's more, AIDE only captures the diversity between selected tuples but without uncertainty, resulting in a poor performance.

(3) SQUID can only support the AND SQL queries. Thus, it performs well in D_1, D_2, D_9, D_{12}. But for the OR, NO, MIXED SQL queries, its performance (F1-score) does not improve even with the increase of #-tuples for training.

7.3 Effectiveness of IDE-Rankingproblem

Metrics We also use *accuracy* defined in Section 7.1, with the only difference that all tuple pairs need to be ranked.

Comparisons We compare (1) DExPLORER using the ranking model in Section 4.2 for ranking answer inference and IQS+ in Sect. 5.3 for question selection, (2) LearnPreference [56] uses ranking SVM for answer inference and we apply our question selection algorithm (i.e.,Algorithm 4) because our hybrid ranking algorithm is essentially ranking SVM, and LambdaMART is only used for feature enrichment, and (3) RankNet [1] is a learning-to-rank algorithm. We use RankNet for answer inference and select questions by randomly selecting k tuples for RankNet.

Exp-8: Effectiveness on IDE-Ranking: Figure 9 shows the results of different approaches on $R_1 - R_{10}$, where x-axis denotes the number of questions answered by users, and y-axis denotes *accuracy*. We make the following observations:

(1) DExPLORER outperforms LearnPreference and RankNet among all IDE-Rankingproblems, and they finally achieve 0.951, 0.935 and 0.893 *accuracy*, respectively. In summary, DExPLORER outperforms LearnPreference and RankNet by 1.7% and 6.5%.

(2) DExPLORER and LearnPreference finally achieve similar *accuracy* after several rounds of labeling, but DExPLORER outperforms LearnPreference in the first few rounds. Hence, the LambdaMART algorithm can help to solve the cold start problem of Ranking SVM.

(3) RankNet behaves better than LearnPreference in the first few rounds, but it is likely to overfit with the number of labeled tuples increasing, such that the accuracy is lower than LearnPreference in the following rounds.

8 Conclusion

We have built a data exploration system DExPLORER. We have implemented a user-friendly front-end that allows the user to select and rank tuples. On the back-end, we have developed a well-performed answer inference model, based on which we select a set of tuples to be answered by the user. We have proved that the optimal question selection problem is NP-hard and proposed an efficient and effective heuristic algorithm. We have also conducted extensive experiments to show the effectiveness of DExPLORER.

Acknowledgements This work is supported by NSF of China (61925205, 61632016, 62102215), Huawei, TAL education, China National Postdoctoral Program for Innovative Talents (BX2021155), China Postdoctoral Science Foundation (2021M691784), Shuimu Tsinghua Scholar and Zhejiang Lab's International Talent Fund for Young Professionals.

References

1. Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., Hullender, G.N.: Learning to rank using gradient descent. In: ICML, pp. 89–96 (2005)
2. Chai, C., Li, G., Li, J., Deng, D., Feng, J.: Cost–effective crowdsourced entity resolution: a partial-order approach. In: Özcan, F., Koutrika, G., Madden, S. (eds.) Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, 26 June–1 July 2016, pp. 969–984. ACM (2016). https://doi.org/10.1145/2882903.2915252
3. Chai, C., Li, G., Li, J., Deng, D., Feng, J.: A partial–order–based framework for cost–effective crowdsourced entity resolution. VLDB J. **27**(6), 745–770 (2018). https://doi.org/10.1007/s00778-018-0509-6
4. Chai, C., Fan, J., Li, G., Wang, J., Zheng, Y.: Crowd–powered data mining. CoRR (2018). arXiv:1806.04968
5. Chai, C., Fan, J., Li, G., Wang, J., Zheng, Y.: Crowdsourcing database systems: overview and challenges. In: 35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, 8–11 April 2019, pp. 2052–2055. IEEE (2019). https://doi.org/10.1109/ICDE.2019.00237
6. Chai, C., Li, G., Fan, J., Luo, Y.: Crowdsourcing-based data extraction from visualization charts. In: 36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, 20–24 April 2020, pp. 1814–1817. IEEE (2020). https://doi.org/10.1109/ICDE48307.2020.00177
7. Chai, C., Cao, L., Li, G., Li, J., Luo, Y., Madden, S.: Human-in-the-loop outlier detection. In: Maier, D., Pottinger, R., Doan, A.H., Tan, W.-C., Alawini, A., Ngo, H.Q. (eds.) Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, Portland, OR, USA, 14–19 June 2020, pp. 19–33. ACM (2020). https://doi.org/10.1145/3318464.3389772
8. Chai, C., Li, G., Fan, J., Luo, Y.: CrowdChart: crowdsourced data extraction from visualization charts. IEEE Trans. Knowl. Data Eng. **33**(11), 3537–3549 (2021). https://doi.org/10.1109/TKDE.2020.2972543
9. Chaudhuri, S., Das, G., Hristidis, V., Weikum, G.: Probabilistic ranking of database query results. In: VLDB, pp. 888–899 (2004)
10. Chu, W., Ghahramani, Z.: Extensions of gaussian processes for ranking: semisupervised and active learning. Learning to Rank, 29 (2005)
11. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
12. Dai, X., Yan, X., Zhou, K., Wang, Y., Yang, H., Cheng, J.: Convolutional embedding for edit distance. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 599–608 (2020)
13. Diaconis, P.: Group representations in probability and statistics. IMS Lecture Notes-monograph **72**(2), 7–108 (1988)
14. Dimitriadou, K., Papaemmanouil, O., Diao, Y.: Explore-by-example: an automatic query steering framework for interactive data exploration. In: SIGMOD, pp. 517–528 (2014)
15. Dwork, C., Kumar, R., Naor, M., Sivakumar, D.: Rank aggregation methods for the web. In: WWW (2001)
16. Fagin, R., Kumar, R., Sivakumar, D.: Comparing top k lists. SIAM J. Discrete Math. **17**(1), 134–160 (2003)
17. Fariha, A., Meliou, A.: Example-driven query intent discovery: abductive reasoning using semantic similarity. PVLDB **12**(11), 1262–1275 (2019)
18. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Ann. Stat. **29**, 1189–1232 (2001)
19. Gharibshah, Z., Zhu, X., Hainline, A., Conway, M.: Deep learning for user interest and response prediction in online display advertising. Data Sci. Eng. **5**(1), 12–26 (2020)

20. Gollapudi, S., Sharma, A.: An axiomatic approach for result diversification. In: WWW, pp. 381–390 (2009)
21. Hassin, R., Rubinstein, S., Tamir, A.: Approximation algorithms for maximum dispersion. Oper. Res. Lett. **21**(3), 133–137 (1997)
22. Haveliwala, T.H.: Topic-sensitive pagerank. In: WWW, pp. 517–526. ACM (2002)
23. Hazelwood, K., Bird, S., Brooks, D., Chintala, S., Diril, U., Dzhulgakov, D., Fawzy, M., Jia, B., Jia, Y., Kalro, A., Law, J., Lee, K., Lu, J., Noordhuis, P., Smelyanskiy, M., Xiong, L., Wang, X.: Applied machine learning at facebook: a datacenter infrastructure perspective. In: HPCA (2018)
24. He, C., Wang, C., Zhong, Y.-X., Li, R.-F.: A survey on learning to rank. In: 2008 International Conference on Machine Learning and Cybernetics, vol. 3, pp. 1734–1739. IEEEE (2008)
25. He, X., Pan, J., Jin, O., Xu, T., Liu, B., Xu, T., Shi, Y., Atallah, A., Herbrich, R., Bowers, S., Candela, J. Q.: Practical lessons from predicting clicks on ads at facebook. In: ADKDD, pp. 5:1–5:9 (2014)
26. Hristidis, V., Gravano, L., Papakonstantinou, Y.: Efficient ir-style keyword search over relational databases. In: VLDB, pp. 850–861 (2003)
27. Hristidis, V., Papakonstantinou, Y.: Discover: keyword search in relational databases. In: VLDB, pp. 670–681 (2002)
28. Huang, E., Peng, L., Palma, L.D., Abdelkafi, A., Liu, A., Diao, Y.: Optimization for active learning-based interactive database exploration. PVLDB **12**(1), 71–84 (2018)
29. Jamieson, K.G., Nowak, R.D.: Active ranking using pairwise comparisons. arXiv preprint arXiv:1109.3701 (2011)
30. Joachims, T.: Training linear svms in linear time. In: SIGKDD, pp. 217–226 (2006)
31. Kalashnikov, D.V., Lakshmanan, L.V., Srivastava, D.: Fastqre: Fast query reverse engineering. In: Proceedings of the 2018 International Conference on Management of Data, pp. 337–350 (2018)
32. Lewis, D.D., Catlett, J.: Heterogeneous uncertainty sampling for supervised learning. In: Machine Learning Proceedings 1994, pp. 148–156. Elsevier (1994)
33. Lewis, D.D., Gale, W.A.: A sequential algorithm for training text classifiers. In: SIGIR'94, pp. 3–12. Springer (1994)
34. Li, H., Chan, C.-Y., Maier, D.: Query from examples: an iterative, data-driven approach to query construction. Proc. VLDB Endow. **8**(13), 2158–2169 (2015)
35. Li, G., Chai, C., Fan, J., Weng, X., Li, J., Zheng, Y., Li, Y., Yu, X., Zhang, X., Yuan, H.: CDB: optimizing queries with crowd–based selections and joins. In: Salihoglu, S., Zhou, W., Chirkova, R., Yang, J., Suciu, D. (eds.) Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, 14–19 May 2017, pp. 146–1478. ACM (2017). https://doi.org/10.1145/3035918.3064036
36. Li, G., Chai, C., Fan, J., Weng, X., Li, J., Zheng, Y., Li, Y., Yu, X., Zhang, X., Yuan, H.: CDB: a crowd–powered database system. Proc. VLDB Endow. **11**(12), 1926–1929 (2018). https://doi.org/10.14778/3229863.3236226
37. Li, M., Wang, H., Li, J.: Mining conditional functional dependency rules on big data. Big Data Min. Anal. **03**(01), 68 (2020)
38. Liaw, A., Wiener, M., et al.: Classification and regression by randomforest. R News **2**(3), 18–22 (2002)
39. Liu, F., Yu, C., Meng, W., Chowdhury, A.: Effective keyword search in relational databases. In: SIGMOD, pp. 563–574 (2006)
40. Luo, Y., Chai, C., Qin, X., Tang, N., Li, G.: Interactive cleaning for progressive visualization through composite questions. In: ICDE, pp. 733–744 (2020)
41. Luo, Y., Qin, X., Tang, N., Li, G.: Deepeye: towards automatic data visualization. In: ICDE, pp. 101–112 (2018)
42. Luo, Y., Qin, X., Tang, N., Li, G., Wang, X.: DeepEye: Creating Good Data Visualizations by Keyword Search. In: Das, G., Jermaine, C.M., Bernstein, P.A. (eds.) Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, 10–15 June 2018, pp. 1733–1736. ACM (2018). https://doi.org/10.1145/3183713.3193545
43. Luo, Y., Chai, C., Qin, X., Tang, N., Li, G.: VisClean: interactive cleaning for progressive visualization. Proc. VLDB Endow. **13**(12), 2821–2824 (2020). https://doi.org/10.14778/3415478.3415484
44. Luo, Y., Tang, N., Li, G., Li, W., Zhao, T., Yu, X.: DeepEye: a data science system for monitoring and exploring COVID–19 data. IEEE Data Eng. Bull. **43**(2), 121–132 (2020)
45. Luo, Y., Li, W., Zhao, T., Yu, X., Zhang, L., Li, G., Tang, N.: DeepTrack: monitoring and exploring spatio-temporal data – a case of tracking COVID–19. Proc. VLDB Endow. **13**(12), 2841–2844 (2020). https://doi.org/10.14778/3415478.3415489
46. Luo, Y., Qin, X., Chai, C., Tang, N., Li, G., Li, W.: Steerable self–driving data visualization. IEEE Trans. Knowl. Data Eng. (2020). https://doi.org/10.1109/TKDE.2020.2981464
47. Luo, Y., Tang, N., Li, G., Tang, J., Chai, C., Qin, X.: Natural Language to visualization by neural machine translation. IEEE Trans. Vis. Comput. Graph. (2021). https://doi.org/10.1109/TVCG.2021.3114848
48. Luo, Y., Tang, N., Li, G., Chai, C., Li, W., Qin, X.: Synthesizing natural language to visualization (NL2VIS) benchmarks from NL2SQL benchmarks. In: SIGMOD, pp. 1235–1247 (2021)
49. Martins, D.M.L.: Reverse engineering database queries from examples: state-of-the-art, challenges, and research opportunities. Inf. Syst. **83**, 89–100 (2019)
50. Masermann, U, Vossen, G.: Design and implementation of a novel approach to keyword searching in relational databases. In: Current Issues in databases and information systems, pp. 171–184 (2000)
51. Mishra, C., Koudas, N.: Interactive query refinement. In: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, pp. 862–873 (2009)
52. Nanongkai, D., Lall, A., Sarma, A.D., Makino, K.: Interactive regret minimization, pp. 109–120 (2012)
53. Panev, K., Michel, S.: Reverse engineering top-k database queries with paleo. In: EDBT, pp. 113–124 (2016)
54. Panev, K., Michel, S., Milchevski, E., Pal, K.: Exploring databases via reverse engineering ranking queries with paleo. Proc. VLDB Endow. **9**(13), 1525–1528 (2016)
55. Psallidas, F., Ding, B., Chakrabarti, K., Chaudhuri, S.: S4: Top-k spreadsheet-style search for query discovery. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 2001–2016 (2015)
56. Qian, L., Gao, J., Jagadish, H.: Learning user preferences by adaptive pairwise comparison. PVLDB **8**(11), 1322–1333 (2015)
57. Qin, X., Chai, C., Luo, Y., Zhao, T., Tang, N., Li, G., Feng, J., Yu, X., Ouzzani, M.: Ranking desired tuples by database exploration. In: ICDE
58. Qin, X., Luo, Y., Tang, N., Li, G.: Deepeye: an automatic big data visualization framework. Big Data Min. Anal. **1**(1), 75–82 (2018)
59. Qin, X., Luo, Y., Tang, N., Li, G.: DeepEye: Visualizing Your Data by Keyword Search. In: Böhlen, M.H., Pichler, R., May, N., Rahm, E., Wu, S.-H., Hose, K. (eds.) Proceedings of the 21st International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, 26–29 March 2018, pp 441–444. OpenProceedings.org (2018). https://doi.org/10.5441/002/edbt.2018.42
60. Qin, X., Luo, Y., Tang, N., Li, G.: Making data visualization more efficient and effective: a survey. VLDB J. **29**(1), 93–117 (2020)
61. Settles, B.: Active learning literature survey (2009)
62. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**(3), 379–423 (1948)
63. Shen, Y., Chakrabarti, K., Chaudhuri, S., Ding, B., Novik, L.: Discovering queries based on example tuples. In: SIGMOD, pp. 493–504 (2014)

64. Shen, L., Shen, Luo, Y., Yang, X., Hu, X., Zhang, X., Tai, Z., Wang, J.: Towards natural language interfaces for data visualization: a survey (2021). arXiv:2109.03506
65. Singh, R., Meduri, V.V., Elmagarmid, A.K., Madden, S., Papotti, P., Quiané-Ruiz, J., Solar-Lezama, A., Tang, N.: Synthesizing entity matching rules by examples. PVLDB **11**(2), 189–202 (2017)
66. Tian, S., Mo, S., Wang, L., Peng, Z.: Deep reinforcement learning-based approach to tackle topic-aware influence maximization. Data Sci. Eng. **5**(1), 1–11 (2020)
67. Tran, Q.T., Chan, C.-Y., Parthasarathy, S.: Query by output. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, pp. 535–548 (2009)
68. Tran, Q.T., Chan, C.-Y., Parthasarathy, S.: Query reverse engineering. VLDB J. **23**(5), 721–746 (2014)
69. Wang, Y., Yao, Y., Tong, H., Xu, F., Lu, J.: A brief review of network embedding. Big Data Min. Anal. **2**(1), 35 (2019)
70. Weiss, Y.Y., Cohen, S.: Reverse engineering spj-queries from examples. In: Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, pp. 151–166 (2017)
71. Wu, Q., Burges, C.J., Svore, K.M., Gao, J.: Adapting boosting for information retrieval measures. Inf. Retriev. **13**(3), 254–270 (2010)
72. Xie, M., Chen, T., Wong, R.C.-W.: Findyourfavorite: an interactive system for finding the user's favorite tuple in the database. In: SIGMOD, pp. 2017–2020 (2019)
73. Xie, M., Wong, R.C.-W., Lall, A.: Strongly truthful interactive regret minimization. In: SIGMOD, pp. 281–298 (2019)
74. Zhang, M., Elmeleegy, H., Procopiuc, C.M., Srivastava, D.: Reverse engineering complex join queries. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 809–820 (2013)
75. Zhang, S., Sun, Y.: Automatically synthesizing sql queries from input-output examples. In: ASE, pp. 224–234 (2013)