Advanced Datasets Discovery: When Multi-Query-Dataset Cardinality Estimation Matters

Tingting Wang*, Shixun Huang*, Zhifeng Bao*, J. Shane Culpepper†, Reza Arablouei‡, Volkan Dedeoglu‡
*RMIT University, *University of Wollongong, †The University of Queensland, ‡Data61, CSIRO

***Abstract*—As available data grows, so too does the demand for datasets discovery in order to meet the custom information needs of users. The information needs of any user can often be expressed via a query set. While the usefulness of a set of datasets may vary across tasks or scenarios, the number of distinct tuples resulting from running a query set on these datasets can serve as a task-agnostic indicator of their usefulness and offer valuable insights into targeted tasks. In this work, we refer to the task-agnostic usefulness of datasets as their *distinctiveness*. We introduce and study the problem of finding a set of datasets with maximum distinctiveness for a user's query set, while keeping the total price of the datasets within a user-defined budget. We first prove the NP-hardness of this problem. Then, we develop a greedy method that achieves an approximation factor of $(1-e^{-1})/2$. But this method is neither efficient nor scalable as it requires the frequent computation of the exact distinctiveness during datasets selection, involving checking every tuple for query result overlaps among multiple datasets. To address this limitation, we propose an efficient and effective machine-learning-based (ML-based) method for estimating the distinctiveness of a set of datasets without the need for tuple-wise checking. The proposed method is the first to enable cardinality estimation (CE) for a query set over multiple datasets, as previous studies only support CE for a single query over a single dataset and cannot effectively identify query result overlaps across multiple datasets. Extensive experiments using five real-world data pools demonstrate that the proposed greedy algorithm with ML-based distinctiveness estimation, outperforms all baselines in effectiveness and efficiency. We believe our problem and solution provide insights for practitioners who wish to further study data marketplaces and/or cardinality estimation.**

I. Introduction

Data, which is being created and collected at unprecedented scales, is an essential resource for informed decision making [1]. This is reinforced by advances in machine learning, whose success heavily relies on huge amounts of data for insights extraction [2]. Therefore, *data preparation*, the process of preparing datasets to meet users' information needs has garnered considerable attention [3].

As shown in Fig. 1, a common pipeline for data preparation consists of three stages – data cleaning, basic datasets discovery, and advanced datasets discovery [2]. At first, datasets are cleansed via a series of operations (e.g., error correction [4], duplicate removal [5], schema alignment [6] and entity matching [7]). Next, a user can perform basic and advanced datasets discovery to find datasets that meet her information needs. In basic discovery, the user finds relevant datasets by utilizing keywords (e.g., real estate in Australia) to indicate her information needs. As keyword-based results are usually coarse-grained, the user may seek to refine the search to a limited number of datasets that fulfill her fine-grained information needs [8]. This can be achieved through advanced datasets discovery, as illustrated in the example below.

Example 1: In a data market platform like Amazon AWS Marketplace [9] and Snowflake Data Marketplace [10], a user seeks to purchase real estate datasets. Through basic datasets discovery, she finds a set $D = \{d_1, d_2, d_3\}$ of candidate real estate datasets shown in Fig. 2. Then, the user aims to purchase data from D within a user-defined budget during advanced datasets discovery. Depending on whether the downstream task is specific or not, advanced datasets discovery can be categorized into task-agnostic discovery and task-specific discovery, as explained below:

• *Task-specific discovery*. The user provides a specific task (e.g., predicting apartment prices in Melbourne) to the platform to find and purchase a subset of D that can be used for the task. To deliver this, the platform uses a criterion (e.g., the model performance improvement for an ML task [11]) to evaluate the task-specific usefulness of datasets and identify a subset of D (e.g., d_1 as it has $A2$ and $A4$ describing apartments in Melbourne) that falls within the budget. As the purchased data is intended for a single task, the process needs to be repeated for each task when the user changes the task or adds new ones, potentially increasing computation-based budgets.

• *Task-agnostic discovery*. The user provides a query set $Q = \{q_1, q_2\}$ (see Fig. 2) to the platform to purchase a subset of D. The platform evaluates the task-agnostic usefulness (defined later) of datasets using a criterion and identifies a subset S of D within the budget (e.g., $S = \{d_1, d_3\}$ as it has the most distinct tuples returned by Q). In contrast to task-specific discovery, task-agnostic discovery emphasizes the task-agnostic usefulness of datasets instead of their usage for any specific task. Such discovery can be beneficial in several scenarios as follows. (1) The first scenario is when the user has an initial task, which she later refines [12]. For example, a user wants to predict property prices in Melbourne but does not define property types at the beginning, and later decides to predict apartment prices in Melbourne after purchasing data. (2) The second scenario is when tasks change over time [12]. For example, a user wants to predict apartment prices in Melbourne first and later wants to predict apartment prices in Sydney. (3) Another scenario is when there are multiple concurrent tasks. This may require different pieces of information from multiple datasets (e.g., one task is predicting apartment prices in Melbourne, and another task is predicting

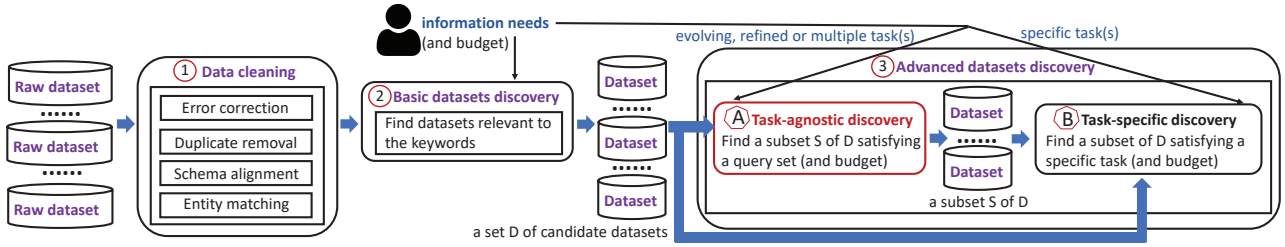
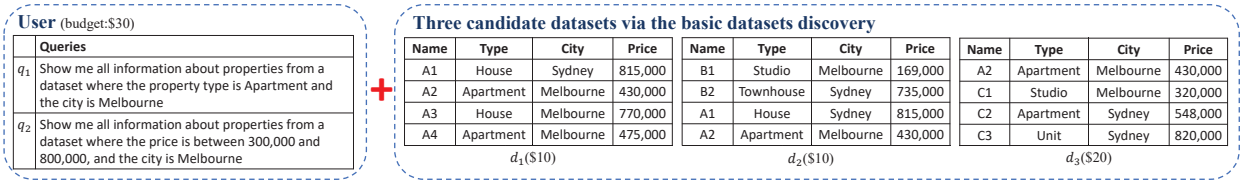


Fig. 1: The data preparation pipeline using a budget to purchase datasets.



User (budget:$30)

	Queries
q_1	Show me all information about properties from a dataset where the property type is Apartment and the city is Melbourne
q_2	Show me all information about properties from a dataset where the price is between 300,000 and 800,000, and the city is Melbourne

Three candidate datasets via the basic datasets discovery

Name	Type	City	Price
A1	House	Sydney	815,000
A2	Apartment	Melbourne	430,000
A3	House	Melbourne	770,000
A4	Apartment	Melbourne	475,000

d_1($10)

Name	Type	City	Price
B1	Studio	Melbourne	169,000
B2	Townhouse	Sydney	735,000
A1	House	Sydney	815,000
A2	Apartment	Melbourne	430,000

d_2($10)

Name	Type	City	Price
A2	Apartment	Melbourne	430,000
C1	Studio	Melbourne	320,000
C2	Apartment	Sydney	548,000
C3	Unit	Sydney	820,000

d_3($20)

Fig. 2: An illustrative example of Step A in Fig. 1.

apartment prices in Syndey), or common information across multiple datasets (e.g., one task is predicting apartment prices in Melbourne, and another task is predicting apartment demand in Melbourne, which is positively correlated with the apartment prices in Melbourne). In summary, task-agnostic discovery can be regarded as the first step of task-specific discovery. It provides general guidance on the usefulness of datasets, and thus saves budgets for repeatedly performing task-specific discovery/purchase.

To achieve this, we focus on task-agnostic discovery in this paper. We are interested in designing a criterion to evaluate the task-agnostic usefulness of datasets. Therefore, we express a user's fine-grained information needs as a query set and use *the number of distinct tuples in the results of the user's query set across multiple datasets* as the evaluation criterion. This is a natural indicator of the preliminary usefulness of the datasets [13]. Moreover, our criterion is more general and more beneficial to ML-based tasks, and potentially other applications that rely on the task-agnostic usefulness of datasets (e.g., epidemiological studies [14] and market strategy development [15]). As a result, we call this evaluation criterion the *distinctiveness* of the discovered datasets.

Next, we introduce and study a problem called *distinctiveness maximization for advanced datasets discovery*: given fine-grained information needs of a user (in the form of a query set), a budget, and the candidate datasets discovered during basic datasets discovery, our goal is to select a subset of candidate datasets for the user to purchase, such that the overall distinctiveness of the subset is maximized while the total price does not exceed the user budget.

We prove that obtaining an exact solution for this problem is NP-hard. Moreover, the distinctiveness computation requires every tuple in the relevant datasets to be checked for given queries. This can limit the scalability of obtaining an exact solution. Hence, we start by developing a greedy method that relies on the exact distinctiveness computation to approximate an optimal solution. We prove that the greedy method can achieve an approximation factor of $\frac{1-1/e}{2}$. However, its reliance on exact distinctiveness computation limits its efficiency and scalability.

Our answer to this limitation is to devise an efficient method to estimate the distinctiveness instead of computing the exact distinctiveness. By definition, estimating the distinctiveness of a set of datasets w.r.t. a query set is to estimate the number of distinct tuples returned for a query set over a set of datasets, a generalized version of the classical cardinality estimation (CE) problem. We call such a CE problem as *Multi-Query-Dataset Cardinality Estimation* (MCE). The MCE problem is different from the existing CE problem [16]–[21], which *estimates the cardinality of a single query over a single dataset*. To better distinguish these two problems, we call the latter as *Single-Query-Dataset Cardinality Estimation* (SCE). The MCE problem cannot easily be addressed using existing solutions for the SCE problem. The key reason is that no previous work has considered how to effectively and efficiently identify any *overlap in results for a query set over multiple datasets* (e.g., three tuples are returned by issuing $\{q_1, q_2\}$ on $\{d_1, d_2\}$ in Fig. 2 since d_1 and d_2 share two common tuples).

Therefore, we propose a novel machine-learning-based (ML-based) method to estimate the distinctiveness of a set of datasets w.r.t. a query set. Specifically, we adopt a pre-trained model to transform the input data summary of a dataset into embeddings that capture useful information about the dataset and given queries as well as connections between them. We predict the distinctiveness of a dataset based on the generated embeddings. We also propose a learned function that generates a data summary for a set of datasets by merging data summaries from individual datasets, so as to predict the distinctiveness of a set of datasets using the pre-trained model. In summary, we make the following technical contributions:

- We introduce a novel distinctiveness maximization problem for advanced datasets discovery and prove its NP-hardness (Section II). To solve this problem, we propose a greedy algorithm using exact distinctiveness computation (`Exact-Greedy` for short) to achieve an approximation factor of $\frac{1-1/e}{2}$. (Section III)
- To avoid expensive exact distinctiveness computation in `Exact-Greedy`, we cast the distinctiveness estimation

problem as the MCE problem, and we propose an ML method to address the MCE problem. Then, we propose a greedy algorithm using ML-based distinctiveness estimation (`ML-Greedy` for short) to solve the distinctiveness maximization problem. (Section IV)

- For evaluation purposes, we develop several strategies to prepare candidate datasets and query sets (based on five real-world data pools), and control the number of overlapping tuples and query results for the candidate datasets in order to reflect various practical test cases. We believe this setup will be useful in future work on this emerging topic. (Section V)
- We conduct an extensive evaluation to demonstrate: 1) In terms of distinctiveness estimation accuracy, our ML-based distinctiveness estimation method is significantly more effective compared to extending state-of-the-art methods for the SCE problem. 2) In terms of distinctiveness maximization, our `ML-Greedy` is competitive than `Exact-Greedy`, and also significantly outperforms all other baselines. 3) In terms of efficiency, our `ML-Greedy` achieves up to four orders of magnitude speedup over `Exact-Greedy`, and three orders of magnitude improvement over the fastest baseline method. (Section VI)

II. Problem Formulation

In this section, we present preliminaries, our problem formulation, and hardness analysis of the problem. We summarize the frequently used notations in Table I.

We consider a set $D = \{d_1, d_2, ..., d_{|D|}\}$ of candidate datasets, resulting from basic datasets discovery, and a query set $Q = \{q_1, q_2, ..., q_{|Q|}\}$ indicating the user-defined information needs. Here, each dataset $d \in D$ is assigned a price $p(d) \in \mathbb{R}^+$, which is determined by any pricing function p of choice (e.g., a tuple-based pricing function [22] or a usage-based pricing function [9], [10]). We denote the price of D as $p(D) = \sum_{d \in D} p(d)$.

When deciding which datasets to purchase, a user is interested in determining how many distinct tuples are returned from all datasets in D based on the query set Q. We refer to this as the *distinctiveness* of the datasets w.r.t. the user's information needs.

Definition 1 (Distinctiveness): Suppose $q(d)$ is a tuple set returned by query q on a dataset d, and $Q(d)$ is the union of all tuple sets returned by all $q \in Q$ on d (i.e., $Q(d) = \cup_{q \in Q} q(d)$). The *distinctiveness* $\mathscr{D}(S, Q)$ for a set S of datasets is the size of the union of $Q(d)$ over all $d \in S$, that is, $\mathscr{D}(S, Q) = \left|\bigcup_{d \in S} Q(d)\right|$.

As mentioned in Sec. I, the distinctiveness estimation problem is essentially the MCE problem, as formalized below.

Definition 2 (Multi-query-dataset cardinality estimation (MCE)): Given a set D of datasets and a query set Q, the MCE problem is to estimate the cardinality of Q over D, i.e., the number of distinct tuples returned by Q over D.

In the rest of this paper, we use distinctiveness estimation and MCE interchangeably. Notably, when Q has only one query ($|Q| = 1$) and S contains only one dataset ($|S| = 1$), the distinctiveness estimation is equivalent to the well-known SCE problem [16]–[19], [23].

TABLE I: Summary of notation.

notation	description
d, D, S	a dataset, a set of datasets, a subset of D
$p(d), p(S)$	the price of a dataset d, the price of a set S of datasets
B	a budget
q, Q	a query, a query set
$q(d)$	a tuple set returned by q on d
$Q(d)$	the union of tuple sets returned by each $q \in Q$ on d
$\mathscr{D}(S, Q)$	the distinctiveness of a set S of datasets over Q

Definition 3 (Distinctiveness Maximization (DM)): Given a user budget B, a query set Q, a set D of candidate datasets, and a pricing function p, the DM problem determines a subset $S^* \subseteq D$ that has the maximum distinctiveness for the given budget B. Formally, we have

$$S^* = \text{argmax}_{S \subseteq D}\, \mathscr{D}(S, Q) \text{ s.t.} \sum_{d \in S} p(d) \leq B. \quad (1)$$

Remark. We accept any type of query that can be transformed into SQL queries as input for our problem. The user queries can be converted to SQL queries at an early stage of the pipeline. For example, natural language queries can be converted to SQL queries via NL2SQL techniques [24]. An SQL query can be represented as follows: SELECT * FROM d WHERE ... AND $l_c \leq c \leq u_c$ AND ..., where d is a dataset and c is a query column. The query column c may contain numerical or categorical data. For categorical data, l_c is equal to u_c. For example, the query q_1 in Fig. 2 can be transformed to SELECT * FROM d WHERE Type = 'Apartment' and City = 'Melbourne', and the query q_2 in Fig. 2 can be transformed to SELECT * FROM d WHERE 300,000 $<$ Price $<$ 800,000 and City = 'Melbourne'.

We prove the NP-hardness of the DM problem via a reduction from the maximum coverage (MC) problem [25].

Theorem 1: The DM problem is NP-hard.

Proof 1: We present a polynomial time reduction for any instance of the MC problem to an instance of the DM problem. Given any instance of MC, we first map each element $e \in V$ to a tuple of e. If the elements are categorical, we give each one a unique index and then map each element e to a tuple of the index. Next, we map each set $S' \subseteq V$ in the MC problem to a dataset $d \in D$ in the DM problem. So, V can be mapped to the universe $T_D = \bigcup_{d \in D} d$ of all tuples. By setting $p(d) = 1$, K is mapped to B. Next, we define a query set $Q = \{q\}$ where q = SELECT * FROM d WHERE $min(T_D) \leq c \leq max(T_D)$. Here, $min(T_D)$ and $max(T_D)$ denote the minimum and maximum value of elements in T, respectively. Therefore, $Q(d)$ contains all the tuples of a dataset d. Thus, we can establish the connection $S' \equiv Q(d)$, and hence the objectives (i.e., maximizing $|\cup_{S' \in \mathcal{S}} S'|$ and $|\bigcup_{d \in S} Q(d)|$), are equivalent.

Based on this reduction, the optimal solution of the DM problem is also an optimal solution for the MC problem. Since the time complexity of the reduction construction is clearly polynomial and MC is NP-hard, the DM problem must also be NP-hard.

Algorithm 1 `Exact-Greedy`

Input: a set D of datasets, a query set Q and a budget B;
Output: a subset $S \subseteq D$ of datasets with its distinctiveness;
1: $S \leftarrow \emptyset$, $\mathcal{T} \leftarrow \emptyset$, $T_S \leftarrow \emptyset$;
2: **for** $d \in D$ **do**
3: $Q(d) \leftarrow \emptyset$;
4: **for** $q \in Q$ **do**
5: $q(d) \leftarrow$ ExecuteQueries(d, q), $Q(d) \leftarrow Q(d) \cup q(d)$;
6: $\mathcal{T}[d] \leftarrow Q(d)$;
7: **while** $D \neq \emptyset$ **do**
8: $g^* \leftarrow 0$, $d^* \leftarrow \emptyset$, $T^* \leftarrow \emptyset$;
9: **for** $d \in D$ **do**
10: $T \leftarrow T_S \cup \mathcal{T}[d]$, $g \leftarrow (|T| - |T_S|)/p(d)$;
11: **if** $g > g^*$ **then** $g^* \leftarrow g$, $d^* \leftarrow d$, $T^* \leftarrow T$;
12: **if** $p(S) + p(d^*) \leq B$ **then**
13: $S \leftarrow S \cup d^*$, $T_S \leftarrow T^*$;
14: $D \leftarrow D \setminus d^*$;
15: $d_t \leftarrow \arg\max_{d \in D \wedge p(d) \leq B} |\mathcal{T}[d]|$;
16: **if** $|T_S| < |\mathcal{T}[d_t]|$ **return** $\{d_t\}$ and $|\mathcal{T}[d_t]|$;
17: **return** S and $|T_S|$;

III. Greedy Algorithm with Exact Distinctiveness Computation

Given the NP-hardness of the DM problem, obtaining an optimal solution is computationally prohibitive, even for datasets of moderate size. Therefore, we propose a greedy algorithm using exact distinctiveness computation (`Exact-Greedy`) to find approximate solutions with theoretical guarantees. The pseudocode for `Exact-Greedy` is given in Alg. 1. The underlying intuition is to progressively select a better solution from a set S of datasets (Lines 7 to 14) and also for a single dataset d_t (Line 15). Specifically, we first execute each query $q \in Q$ on each dataset $d \in D$ and merge the tuple sets $q(d)$ obtained by the query $q \in Q$ to obtain the union $Q(d)$ of $q(d)$ (Lines 4-5). To illustrate this, we introduce the notation $\mathcal{T}[d]$ to record $Q(d)$ (Line 6). At each iteration, we add the dataset d^* with the greatest marginal gain g^* into S until the budget is exhausted (Lines 9-13). In Line 15, we find a single dataset d_t with the maximum distinctiveness $|\mathcal{T}[d_t]|$. We present its approximation guarantee in Theorem 2.

Theorem 2: `Exact-Greedy` (Alg. 1) can achieve an approximation factor of $\frac{1-1/e}{2}$ for the DM problem.

Proof 2: Let S^* denote the optimal set of datasets, S_m denotes the set of datasets with size m added to S in the first l iterations, d_{m+1} denotes the first dataset considered from S^*, but not added to S due a budget constraint violation, and $S_{m+1} = \{d_{m+1}\} \cup S_m$. By applying Lemma 2 of [26], we see that $\mathscr{D}(S_{m+1}, Q) \geq (1 - (1/e)) \cdot \mathscr{D}(S^*, Q)$ in Alg. 1. Let $\Delta\mathscr{D}$ be the distinctiveness increase after adding d_{m+1} to S_m. So, $\mathscr{D}(S_{m+1}, Q) = \mathscr{D}(S_m, Q) + \Delta\mathscr{D} \geq (1 - \frac{1}{e}) \cdot \mathscr{D}(S^*, Q)$. Since $\Delta\mathscr{D}$ cannot be greater than $\mathscr{D}(\{d_t\}, Q)$ for a single best dataset d_t, $\mathscr{D}(S_m, Q) + \mathscr{D}(\{d_t\}, Q) \geq \mathscr{D}(S_m, Q) + \Delta\mathscr{D} \geq (1 - \frac{1}{e}) \cdot \mathscr{D}(S^*, Q)$. Therefore, either $\mathscr{D}(S_m, Q)$ or $\mathscr{D}(\{d_t\}, Q)$ is not smaller than $\frac{(1-1/e)}{2} \cdot \mathscr{D}(S^*, Q)$. Thus, `Exact-Greedy` achieves an approximation ratio of $\frac{(1-1/e)}{2}$.

Time complexity analysis. Assume that d is a dataset in D with the most tuples. The algorithm requires $\mathcal{O}(|d||Q|)$ time to execute each query $q \in Q$, and requires $\mathcal{O}(|d|^2|Q|)$ time to merge the tuple set $q(d)$ obtained from each query $q \in Q$ (Lines 4-5). Therefore, constructing $\mathcal{T}[d]$ requires $\mathcal{O}(|d|^2|Q||D|)$ time (Lines 2-6). Computing the marginal gain for a dataset d requires $\mathcal{O}(|d||D||\bigcup_{d \in D} d|)$ time, and $\mathcal{O}(|d||T_D||D|^3)$ time is required to find the set S of the most relevant datasets (lines 7-14). Hence, the total time complexity of Alg. 1 is $\mathcal{O}(|d|^2|Q||D| + |d||\bigcup_{d \in D} d||D|^3)$.

IV. Greedy Algorithm with ML-based Distinctiveness Estimation

Although `Exact-Greedy` is able to achieve a $\frac{1-1/e}{2}$ approximation ratio, it is computationally inefficient due to the exact computation of the marginal gain in Line 10 of Alg. 1. In our experiments, the exact distinctiveness computation takes one hour with 20 datasets (100k tuples in a dataset) and 10 queries. This is impractical since a production data discovery system must deliver timely feedback to meet the user's information needs and prevent user abandonment.

Therefore, a natural question arises: is it possible to efficiently estimate the distinctiveness while maintaining high accuracy? Our answer is affirmative. According to Definition 2, distinctiveness estimation is essentially an MCE problem, where the goal is to estimate the cardinality of a query set over a set of datasets. To address the MCE problem, we begin by examining the solution of the SCE problem, which involves estimating the cardinality of a single query over a single dataset. A state-of-the-art SCE method is `IRIS` [16], which uses a pre-training summarization approach to generate data summaries and utilize them to efficiently obtain approximate SCE solutions. Inspired by `IRIS`, we develop a machine-learning-based (ML-based) estimation method to solve the MCE problem. This method consists of five components, namely *Component 1: data summary generation*, *Component 2: query-aware dataset embedding generation*, *Component 3: query-set embedding generation*, *Component 4: distinctiveness estimation* and *Component 5: merging data summaries*, as shown in Fig. 3.

Note that `IRIS` addresses the SCE problem and hence cannot detect overlaps in results for a query set over multiple datasets, which is unique to the MCE problem. Hence, while Component 1 is adopted from `IRIS`, we design the remaining four components entirely from the ground up to tackle the challenges specific to the MCE problem.

We employ Component 1 to generate the data summary of the considered dataset. We then utilize Component 2 to transform the data summary into a query-aware dataset embedding by capturing the connections between the query set and the data summary. Next, we use Component 3 to generate a query-set embedding by merging the information of the queries in the query set. Finally, we use Component 4 to estimate the distinctiveness of a dataset w.r.t. a query set while utilizing the two embeddings generated by the previous components. To estimate the distinctiveness of a set of datasets w.r.t. a query set via the above components, we need to first compute the data summary of the set of datasets. The data summary of a set of datasets can be computed by iteratively merging the current data summary with the data summary of a new dataset, starting

with a single dataset within the set. Hence, our ML-based method uses Component 5 containing a new learned function to efficiently generate the data summary of a set of datasets by merging the data summaries of individual datasets. We will elaborate on these components in Sec. IV-A - Sec. IV-E. We describe our proposed greedy algorithm employing ML-based distinctiveness estimation (`ML-Greedy`) in Sec. IV-F, which combines all of the five components.

A. Component 1: Data Summary Generation

We adopt `IRIS` [16] for our data summary generation (see the left part of Fig. 3). That is because, compared to other approaches (histograms [27], sketches [28]), such a pre-training summarization approach can avoid per-dataset training and reduce the cost to construct summaries while achieving high accuracy in answering the SCE problem.

To generate the data summary of a dataset d, `IRIS` requires two steps. The first step is *identifying column sets*, which identifies a set $\mathcal{C}_d$ of column sets that have high correlations to the dataset d. The second step is *generating column set embeddings*, which uses pre-trained models to learn a column set embedding e_C for each column set $C \in \mathcal{C}_d$ and uses column set embeddings as the data summary E_C^d ($E_C^d = \{..., e_{C_i}, ...\}$, where $C_i \in \mathcal{C}_d$). Next, we describe these two steps in detail.

Identifying Column Sets. For each dataset $d \in D$, we randomly select a small set of tuples from d to estimate the correlation scores between all pairs of columns using CORDS [29]. With these correlation scores, a graph is constructed where columns are nodes and edges are weighted according to the respective correlation score. Using this weighted graph, a set of nodes is iteratively selected as a column set. In each iteration, among all the possible cliques containing no more than κ (2 in our experiments) nodes, a clique that has the maximum sum of edge weights is selected as a column set, and then all of the selected edges are removed from the graph.

Let $\Pi_C d$ denote a projection of a dataset d on columns in C. The column set selection procedure is repeated until the total storage usage of $\Pi_C d$ for all column sets reaches a predefined threshold (4kb in our experiments). This results in a candidate set $\mathcal{C}_d$ of column sets. For example, in Fig. 2, $\mathcal{C}_{d_1}$ of d_1 can be $\{C_1 = \{\mathsf{Name}, \mathsf{Type}\}, C_2 = \{\mathsf{Type}, \mathsf{City}\}, C_3 = \{\mathsf{City}, \mathsf{Price}\}\}$.

Generating Column Set Embeddings. For each column set $C \in \mathcal{C}_d$ and the corresponding rows $\Pi_C d$ in the dataset $d \in D$, a row embedding is learned for each row $r \in \Pi_C d$, and then the average of all such row embeddings for C is produced, which is the column set embedding e_C of C, computed as:

$$e_C \triangleq \frac{1}{|\Pi_C d|} \sum_{i=1}^{|\Pi_C d|} \psi(r_i). \tag{2}$$

Here, $\psi(\cdot)$ is a learned function (model) for a row embedding. It is computed using the quantization function $\mathbf{q}(\cdot)$ proposed by [16], a column embedding function $\phi_c(\cdot)$, a row embedding function $\phi_r(\cdot)$, and a ReLU activation function $\delta(\cdot)$, expressed as:

$$\psi(r_i) \triangleq \delta(\phi_r(\mathrm{Reorder}([\cdots, \phi_c(\mathbf{q}(c_i^j)), \cdots]))), \tag{3}$$

where $\phi_c : \mathbb{R}^\xi \to \mathbb{R}^{64}$, $\phi_r : \mathbb{R}^\ell \to \mathbb{R}^\eta$, and c_i^j are values of the j-th column in C for row r_i. The $\mathrm{Reorder}$ function reorders the values according to the number of bits assigned such that the most distinct columns are aligned at the same input position for ϕ_r.

The quantization function $\mathbf{q}(\cdot)$ replaces the value of each column with an identifier containing at most ξ bits while $\phi_c(\mathbf{q}(c_i^j))$ encodes the identifier $\mathbf{q}(c_i^j)$ by an embedding layer. Let v_i be a vector that concatenates the reordered outputs of ϕ_c. The function $\phi_r(v_i)$ generates a row embedding of $r_i \in \Pi_C d$ using a fully-connected neural-network (NN) layer. The function $\phi_r(v_i)$ inputs ℓ bits and outputs η bits (the size of the row embedding). Finally, $\delta(\phi_r(v_i))$ is the ReLU activation function applied to $\phi_r(v_i)$. Using Eq. 2, a column set embedding e_C of C is computed with the size of η.

Example 2: Consider the column set $C = \{\mathsf{City}, \mathsf{Price}\}$ from dataset d_1 in Fig. 2. We allocate $\ell = 3$ bits in total to the two columns, with 2 and 4 distinct values in each. Therefore, using the quantization method from [16], we assign $[1, 2]$ bits to the columns. Using a simple embedding method, we have $\phi_c(\mathbf{q}(c_1^1)) = [0]$, $\phi_c(\mathbf{q}(c_1^2)) = [1, 0]$, and $\mathrm{Reorder}[\phi_c(\mathbf{q}(c_1^1)), \phi_c(\mathbf{q}(c_1^2))] = [\phi_c(\mathbf{q}(c_1^2)) \ , \phi_c(\mathbf{q}(c_1^1))] = [1, 0, 0]$ for $r_1 \in \Pi_C d$. Thus, $\psi(r_1) = \delta(\phi_r([1, 0, 0]))$.

B. Component 2: Query-aware Dataset Embedding Generation

In order to capture the connections between the queries in Q and the data summary E_C^d of a dataset d, we generate a query-aware dataset embedding e_d^Q by using a learned model $\phi_d(\cdot)$. This process is shown in the query-aware dataset embedding portion of Fig. 3.

To achieve a more efficient computation, we construct a lookup table $\mathcal{C}_Q$ which maintains the column sets associated with each query $q \in Q$, i.e., $\mathcal{C}_Q[q] = \{C | C \in \cap_{d \in D} \mathcal{C}_d$ and $C \cap \mathrm{ColsOf}(q)\}$. Here, $\mathrm{ColsOf}(q)$ refers to query columns. Each column set in $\mathcal{C}_q$ is also a column set for a dataset in D. Given a data summary E_C^d for a dataset d and $\mathcal{C}_Q[q]$, for each query $q \in Q$, a dataset embedding, e_d^q, is constructed based on q, by concatenating all the column set embeddings in E_C^d which are associated with $\mathcal{C}_Q[q]$, such as $e_d^q = [..., e_{C_i}, ...]$ where $e_{C_i} \in E_C^d$ and $C_i \in \mathcal{C}_Q[q]$.

Next, $\phi_d : \mathbb{R}^{2\eta x} \to \mathbb{R}^{\eta x}$ is constructed, which is a fully-connected NN layer followed by a ReLU activation function $\delta(\cdot)$, to learn a query-aware dataset embedding for a dataset d iteratively:

$$\begin{aligned}
&e_d^{q_1, q_2} = \delta(\phi_d([e_d^{q_1}, e_d^{q_2}])) \\
&... \\
&e_d^{q_1, ..., q_j} = \delta(\phi_d([e_d^{q_j}, e_d^{q_1, ..., q_{j-1}}])) \\
&... \\
&e_d^{Q} = \delta(\phi_d([e_d^{q_{|Q|}}, e_d^{q_1, ..., q_{|Q|-1}}])).
\end{aligned} \tag{4}$$

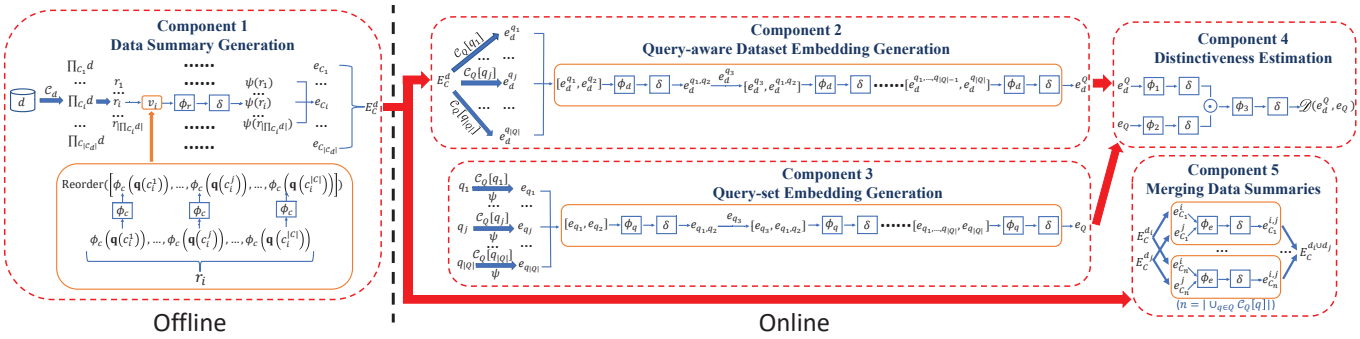


Fig. 3: The ML-based distinctiveness estimation method.

For the first query $q_1 \in Q$ and the second query $q_2 \in Q$, the two dataset embeddings, $e_d^{q_1}$ (based on q_1) and $e_d^{q_2}$ (based on q_2), are concatenated and used as the current input for ϕ_d. The output embedding $e_d^{q_1,q_2}$ combines the information of both $e_d^{q_1}$ and $e_d^{q_2}$. Similarly, the embedding $e_d^{q_1,\ldots,q_j}$ is generated using $e_d^{q_j}$ and $e_d^{q_1,\ldots,q_{j-1}}$. At the end of all iterations, $e_d^Q = e_d^{q_1,\ldots,q_{|Q|}}$ represents the final query-aware dataset embedding.

Note that, since the size of a column set embedding is η, the input size of $\phi_d(\cdot)$ is fixed to $2\eta x$ and the output size of $\phi_d(\cdot)$ to ηx. If the input size is smaller than $2\eta x$, the input is padded with zeros to ensure that the size is $2\eta x$.

C. Component 3: Query-set Embedding Generation

In this section, we generate a query-set embedding e_Q to merge the information of queries in a query set Q. This process is summarized in the query-set embedding portion of Fig. 3.

For each query $q \in Q$, $C \in \mathcal{C}_Q[q]$ indicates a column set where columns appear in the WHERE clause of q. In general, each column c is associated with a query range whose minimal value is l_c and maximal value is u_c, as discussed in Sec. II. As a result, we can rewrite the query as $q = [q_l^C, q_h^C]$, where $q_l^C = [l_{c_1}, ..., l_{c_{|C|}}]$ and $q_h^C = [u_{c_1}, ..., u_{c_{|C|}}]$. The same row embedding function $\psi(\cdot)$ (see Eq. 3) is employed on the minimal and maximal value of the columns for a query, q_l^C and q_h^C, to generate two embeddings of q for C, $\psi(q_l^C)$ and $\psi(q_h^C)$. Then, a query embedding e_q is constructed using q by concatenating all the embeddings of q for $C \in \mathcal{C}_Q[q]$, i.e., $e_q = [..., \psi(q_l^{C_i}), \psi(q_h^{C_i}), ...]$ where $C_i \in \mathcal{C}_Q[q]$.

Example 3: Consider the column set $C = \{\text{City}, \text{Price}\}$ in Example 2 and the query q = SELECT Count(*) FROM d WHERE 300,000 $\leq$ Price $\leq$ 800,000. We have $\phi_c(\mathbf{q}(\text{Melbourne})) = [0]$ and $\phi_c(\mathbf{q}(\text{Sydney})) = [1]$, indicating the range of all distinct values in City. We also have $\phi_c(\mathbf{q}(300,000)) = [0,0]$ and $\phi_c(\mathbf{q}(800,000)) = [1,0]$. Therefore, $\text{Reorder}[\phi_c(\mathbf{q}(\text{Melbourne})), \phi_c(\mathbf{q}(300,000))] = [0,0,0]$ and $\text{Reorder}[\phi_c(\mathbf{q}(\text{Sydney})), \phi_c(\mathbf{q}(800,000))] = [1,0,1]$. So, we have a query embedding $[\psi(q_l^C), \psi(q_h^C)] = [\delta(\phi_r([0,0,0])), \delta(\phi_r(1,0,1))]$ to represent q.

After obtaining all of the query embeddings, the iterative procedure described in Section IV-B is used to generate the query-set embedding as shown below:

$$
\begin{aligned}
e_{q_1,q_2} &= \delta(\phi_p([e_{q_1}, e_{q_2}])) \\
&\ldots \\
e_{q_1,\ldots,q_j} &= \delta(\phi_p([e_{q_j}, e_{q_1,\ldots,q_{j-1}}])) \\
&\ldots \\
e_Q &= \delta(\phi_p([e_{q_{|Q|}}, e_{q_1,\ldots,q_{|Q|-1}}])).
\end{aligned} \tag{5}
$$

Here, $\phi_q : \mathbb{R}^{4\eta x} \rightarrow \mathbb{R}^{2\eta x}$ is a fully-connected NN layer and $\delta(\cdot)$ is a ReLU activation function. Since the size of a column set embedding for a query is 2η, we fix the input size of $\phi_q(\cdot)$ to $4\eta x$ and the output size of $\phi_q(\cdot)$ to $2\eta x$. If the input size is smaller than $4\eta x$, the embedding is padded with zeros to ensure that the input size is $4\eta x$.

D. Component 4: Distinctiveness Estimation

Using our query-aware dataset embedding e_d^Q and query-set embedding e_Q as the input, three learned models are created, $\phi_1(\cdot)$, $\phi_2(\cdot)$, and $\phi_3(\cdot)$, which estimate the distinctiveness of each dataset d as

$$
\mathscr{D}(e_d^Q, e_Q) = \delta(\phi_3(\delta(\phi_1(e_d^Q)) \odot \delta(\phi_2(e_Q)))). \tag{6}
$$

Here, $\phi_1 : \mathbb{R}^{\eta x} \rightarrow \mathbb{R}^{\eta}$ is a fully-connected NN layer, $\phi_2 : \mathbb{R}^{2\eta x} \rightarrow \mathbb{R}^{\eta}$ is another fully-connected NN layer, $\phi_3 : \mathbb{R}^{\eta} \rightarrow \mathbb{R}^1$ is a multilayer perceptron with one hidden layers, and $\delta(\cdot)$ is a ReLU activation function.

Distinctiveness Estimation Process. As shown in Fig. 3, our distinctiveness estimation process involves Component 1 - Component 4, each associated with one or more models. Specifically, ϕ_c and ϕ_r are used to generate a data summary, ϕ_d is used to generate a query-aware dataset embedding, and ϕ_q is used to generate a query-set embedding, with ϕ_1, ϕ_2, and ϕ_3 being used to estimate the distinctiveness. These models are pre-trained end-to-end, with no hand-engineered intermediate features required [30]. The pseudocode for this process is presented in Alg. 2. First, we find the dataset embedding e_d^q for each query $q \in Q$, and a query embedding e_q for each query $q \in Q$ using $\mathcal{C}_Q[q]$ and E_C^d (Lines 2-6). Then, a query-aware dataset embedding e_d^Q is generated in Line 7, and a query-set embedding e_Q is generated in Line 8. Finally estimated the distinctiveness for dataset d is computed in Line 9. Note that we adopt an iterative manner to generate the query-aware dataset embedding and the query-set embedding, but the order of queries has almost no impact on the final result.

Algorithm 2 Distinctiveness($Q, E_C^d, n, \mathcal{C}_Q$)

Input: the data summary E_C^d of d, a query set Q, the number of tuples n in d, a lookup table $\mathcal{C}_Q$ to maintain column sets associated with each query $q \in Q$;
Output: estimated distinctiveness $\mathscr{D}$;
1: $\mathscr{D} \leftarrow 0$, $E_d \leftarrow \emptyset$, $E_Q \leftarrow \emptyset$
2: **for** $q \in Q$ **do**
3: $\quad e_d^q \leftarrow \emptyset$, $e_q \leftarrow \emptyset$;
4: $\quad$ **for** $e_C \in E_C^d$ and $C \in \mathcal{C}_Q[q]$ **do**
5: $\quad\quad e_d^q \leftarrow [e_d^q, e_C]$, $e_q \leftarrow [e_q, \psi(q_l^C), \psi(q_h^C)]$;
6: $\quad E_d \leftarrow E_d \cup e_d^q$, $E_Q \leftarrow E_Q \cup e_q$;
7: Generate e_d^Q by each $e_d^q \in E_d$ using Eq. 4;
8: Generate e_Q by $e_q \in E_Q$ using Eq. 5;
9: $\mathscr{D} \leftarrow \mathscr{D}(e_d^Q, e_Q) \times n$ using Eq. 6;
10: **return** u;

The training loss function. Consider a set $\mathcal{D}$ of training pairs where each training pair $(\widetilde{\mathscr{D}}(d,Q), \mathscr{D}(d,Q))$ contains the estimated distinctiveness $\widetilde{\mathscr{D}}(d,Q)$ for d using Q, and a true distinctiveness $\mathscr{D}(d,Q)$ for d using Q, the mean squared error (MSE) can be as the loss function: $\text{loss} = \frac{1}{|\mathcal{D}|}\sum_{(\widetilde{\mathscr{D}}(d,Q),\mathscr{D}(d,Q))\in\mathcal{D}}(\widetilde{\mathscr{D}}(d,Q) - \mathscr{D}(d,Q))^2$.

E. Component 5: Merging Data Summaries

This component is proposed to compute the data summary $E_C^{S\cup d}$ of $S\cup d$ by merging the data summary E_C^S of the current set S of datasets and the data summary E_C^d of a dataset d, so as to estimate the distinctiveness of $S \cup d$ w.r.t Q using the above estimation process (Alg. 2).

Note that `IRIS` [16] has proposed a method to incrementally update the data summary E_C^d of a dataset d when inserting new rows into d. Specifically, for each columnset C of d, new rows in C denoted as R_{new}, a new column set embedding can be obtained as $e'_C = \frac{ne_C + \sum_{r\in R_{new}} \psi(r)}{n+|R_{new}|}$. Here, e_C is the column set embedding for C in E_C^d and $\psi(\cdot)$ is the row embedding function shown in Eq. 3. Subsequently, the column set embedding e_C in E_C^d is replaced with e'_C.

However, the above method is computationally expensive when updating the data summary E_C^S of the current set S of datasets using new rows from a dataset d. In order to support updates, a check must be performed to determine if a row in d exists that is already referenced in S.

In such cases, a new trainable model is used to learn column set embeddings for a data summary. This process is summarized in the merging data summaries portion of Fig. 3. Given two datasets d_i and d_j, $d_{i,j}$ is the dataset created by merging d_i and d_j. Let e_C^i be a column set embedding for d_i, e_C^j a column set embedding for d_j, and $e_C^{i,j}$ a column embedding for $d_{i,j}$, where $C \in \cup_{q\in Q}\mathcal{C}_Q[q]$. A new embedding e is learned from e_C^i and e_C^j, and is a good approximation for $e_C^{i,j}$, computed as

$$\delta(\phi_e([e_C^i, e_C^i])) \approx e_C^{i,j} \tag{7}$$

where $\phi_e : \mathbb{R}^{2\eta} \rightarrow \mathbb{R}^{\eta}$ is a fully-connected NN layer. As before, δ is the ReLU activation function.

Merging data summaries. The procedure for merging data summaries is shown in Alg. 3. Given a data summary E_C^d for the dataset d, a data summary E_C^S for the set S of datasets,

Algorithm 3 MergeEmbeddings($E_C^d, E_C^S, \mathcal{C}_Q$)

Input: data summary E_C^d of dataset d, data summary E_C^S of the set S of datasets, and a lookup table $\mathcal{C}_Q$ to maintain column sets associated with each query $q \in Q$;
Output: data summaries $E_C^{S\cup d}$;
1: $E_C^{S\cup d} \leftarrow \emptyset$;
2: **for** $e_C \in E_C^S$ **do**
3: $\quad$ **if** $C \in \cup_{q\in Q}\mathcal{C}_Q[q]$ **then**
4: $\quad\quad$ find column set embedding e'_C of C from E_C^d;
5: $\quad\quad e \leftarrow \delta(\phi_e([e_C, e'_C]))$;
6: $\quad\quad e_C \leftarrow e$;
7: $\quad E_C^{S\cup d} \leftarrow E_C^{S\cup d} \cup e_C$
return $E_C^{S\cup d}$;

and a lookup table $\mathcal{C}_Q$ which maps column sets $\mathcal{C}_q$ associated with each query $q \in Q$, we update the column set embeddings in E_C^S, whose column set is in $\cup_{q\in Q}\mathcal{C}_q$ using Eq. 7.

The training loss function. Consider a set $\mathcal{S}$ of training pairs where each training pair $(\widetilde{e}_C^{i,j}, e_C^{i,j})$ contains an estimated column pair embedding $\widetilde{e}_C^{i,j}$ of a column set C on a dataset created by merging d_i and d_j and the corresponding ground-truth column pair embedding $e_C^{i,j}$. We use the MSE as the loss function, i.e., $\text{loss} = \frac{1}{|\mathcal{S}|}\sum_{(\widetilde{e}_C^{i,j}, e_C^{i,j})\in\mathcal{S}}(\widetilde{e}_C^{i,j} - e_C^{i,j})^2$.

F. A Complete Algorithm

With all the five components introduced (Sec. IV-A-IV-E), we are in a position to present a complete procedure for the proposed greedy algorithm using ML-based distinctiveness estimation (`ML-Greedy`), as shown in Alg. 4. First, an offline process (Line 1) generates the set of column sets $\mathcal{C}_d$ and the data summary E_C^d for each dataset $d \in D$. Then, all the remaining operations between Lines 2-22 are an online process. A lookup table $\mathcal{C}_Q$ is constructed to maintain a set of column sets associated with each query $q \in Q$ (Lines 3-4). Then, the best solution is found for the datasets in S (Lines 6-21), or for a single dataset d_t (Line 22). Specifically, in the first iteration, the distinctiveness $\mathscr{D}(\{d\}, Q)$ of any single dataset d is calculated using Alg. 2 and is maintained in $\mathbb{D}$ (Line 11). In the other iterations, the marginal gain $g = \mathscr{D}(S \cup d, Q) - \mathscr{D}(S, Q)$ of d w.r.t S is calculated using Alg. 2, after merging the data summary for S and the data summary for d with Alg. 3 (Lines 14-16). The set S is iteratively updated by adding the dataset d^* with the greatest marginal gain g^* until the budget is exceeded (Lines 18-20). In Line 22, we find a single dataset d_t with the maximum distinctiveness $\mathbb{D}[d]$.

Time complexity analysis. It takes $\mathcal{O}(|\mathcal{C}_Q[q]||Q|)$ time to construct the dataset embedding and the query embedding, for each query $q \in Q$ (Lines 2-6 in Alg. 2) and $\mathcal{O}(8\eta^2x^2|Q|)$ time to generate the query-aware dataset embedding and the query-set embedding (Lines 7-8 of Alg. 2). Additionally, it takes $\mathcal{O}(|\mathcal{C}_Q[q]||Q|+8\eta^2x^2|Q|)$ time to estimate the utility for a dataset (Lines 1-9 in Alg. 2). Finally, it takes $\mathcal{O}(\eta^2|\cup_{q\in Q}\mathcal{C}_Q[q]|)$ time to merge the data summary E_C^d for a dataset d with a data summary E_c^S for the datasets S (Lines 1-7 in Alg. 3). So, it takes $\mathcal{O}(|\mathcal{C}_Q[q]||Q|+8\eta^2x^2|Q|+\eta^2|\cup_{q\in Q}\mathcal{C}_Q[q]|)$ time to compute the marginal gain for a dataset d w.r.t the set S of datasets (Lines 14-16 in Alg. 4). Therefore, the total time

Algorithm 4 ML-Greedy

Input: a set of datasets D, a query set Q, the budget B;
Output: a set $S \subseteq D$ of datasets with its distinctiveness;
Generate a set $\mathcal{C}_d$ of column sets and data summary E_C^d for each $d \in D$;
/* offline process */
$\mathcal{C}_Q \leftarrow \emptyset$;
for $q \in Q$ **do**
    $\mathcal{C}_Q[q] \leftarrow \{C | C \in \cap_{d \in D} \mathcal{C}_d$ and $C \cap \text{ColsOf}(q)\}$;
$S \leftarrow \emptyset$, $\mathbb{D} \leftarrow \emptyset$, $E_C^S \leftarrow \emptyset$, $n^* \leftarrow 0$, $\mathscr{D}^* \leftarrow 0$;
while $D \neq \emptyset$ **do**
    $g^* \leftarrow 0$, $d^* \leftarrow \emptyset$, $E^* \leftarrow \emptyset$;
    for $d \in D$ **do**
        if $S == \emptyset$ **then**
            $\mathscr{D} \leftarrow \text{Distinctiveness}(Q, E_C^d, |d|, \mathcal{C}_Q)$;
            $g \leftarrow \frac{\mathscr{D}}{p(d)}$, $\mathbb{D}[d] \leftarrow \mathscr{D}$;
            if $g > g^*$ **then** $d^* \leftarrow d$, $g^* \leftarrow g$, $E^* \leftarrow E_C^d$;
        else
            $E_C^{S \cup d} \leftarrow \text{MergeEmbeddings}(E_d, E_C^S, \mathcal{C}_Q)$
            $\mathscr{D} \leftarrow \text{Distinctiveness}(Q, E_C^{S \cup d}, n^* + |d|, \mathcal{C}_Q)$;
            $g \leftarrow \frac{\mathscr{D} - \mathscr{D}^*}{p(d)}$;
            if $g > g^*$ **then** $d^* \leftarrow d$, $g^* \leftarrow g$, $E^* \leftarrow E_C^{S \cup d}$;
    if $p(d^*) + p(S) \leq B$ **then**
        $E_C^S \leftarrow E^*$, $S \leftarrow S \cup d^*$;
        $\mathscr{D}^* \leftarrow \mathscr{D}^* + g^* \times p(d^*)$, $n^* \leftarrow n^* + |d^*|$;
    $D \leftarrow D \setminus d^*$;
$d_t \leftarrow \arg\max_{d \in D \wedge p(d) \leq B} \mathbb{D}[d]$;
If $\mathscr{D}^* < \mathbb{D}[d_t]$ **return** $\{d_t\}$ and $\mathbb{D}[d_t]$;
return S and $\mathscr{D}^*$;

complexity for our greedy algorithm using ML-based utility estimation is $\mathcal{O}((8\eta^2 x^2 |Q| + \eta^2 |\cup_{q \in Q} \mathcal{C}_Q[q]|)|D|^2)$. Note that in model pre-training, we set $\eta = 128$ (column set embedding size) and $x = 4$ (i.e., a query corresponds to at most 4 column sets).

Note that $|d|$ is generally in the order of millions, and thus the empirical efficiency improvement provided by ML-Greedy is potentially even more substantial. This can be evidenced by the fact that ML-Greedy can be up to four orders of magnitude faster than Exact-Greedy as shown in our experiments.

V. Datasets and Queries Preparation

A. Dataset Preparation

We use five real-world data pools with different sizes and column types (see Table II), none of which are used in pre-training. Following existing studies [31], [32], we sample tuples from a data pool d_{pool} to generate datasets, which together constitute a set D of candidate datasets.

As discussed in Sec. I, the distinctiveness of datasets is critical in advanced datasets discovery, so it is important to control overlapping tuples during candidate dataset generation. To achieve this, we set two parameters, s_{min} and s_{max}, to control the lower and upper bound ($s_{min} \cdot |D|$ and $s_{max} \cdot |D|$) of the expected number of times that a tuple should appear in D. Specifically, we set a sampling rate s from $[s_{min}, s_{max}]$ at random, and then randomly sample $s \cdot |d_{pool}|$ tuples from d_{pool} to construct the candidate dataset d. Then, we generate multiple datasets to form a set D of candidate datasets.

In our experiments, we set $|D|$, the size of D, as 20 by default. We also fix $s_{min} = 1/|D| = 0.05$, which indicates a tuple is expected to *appear in at least one dataset*. We set $s_{max} = 2/|D| = 0.1$ (by default), indicating that a tuple is expected to *appear in at most two datasets*. This ensures that the datasets in D have a realistic overlap of tuples but meanwhile are not too similar to each other.

TABLE II: Data pools that are not used in model pre-training.

Name	# of cols (R/C)*	# of records
TPCH-LineItem [33]	11/4	6M
DMV [34]	7/13	11M
IMDB-CastInfo [35]	6/1	36M
Airline-OnTime [36]	66/17	440K
RealEstate [37]	21/14	1.4M

Price and budget. For simplicity, we set $p(d) = w \times |d|$ for each dataset d where w is chosen randomly from $(0, 1]$, and follow existing pricing strategies [22], which prices a dataset based on the number of tuples. To better control budget variations, we propose a *B*-ratio, which is the ratio of the budget B and the total price of the datasets in D, $\sum_{d \in D} p(d)$, set to 0.5 by default. Notably, our proposed algorithms can work with any pricing strategy.

B. Query Set Preparation

Intuitively, users would like to issue queries with unique constraints to find distinct tuples for each query. That is, they expect query results returned by different queries to be diverse. Thus, *it is important to control the number of overlapping tuples returned by different queries*. To this end, we set a parameter ol that controls the *minimum overlap ratio* between the tuples returned by a pair of queries for a dataset. The constraint ol uses a similar idea from [16] to generate queries. To simplify the configuration, at most one categorical column is used by a query.

First, the number of categorical columns is determined for a query, i.e., $k_c = 0$ or $k_c = 1$, uniformly at random, and then a query is generated based on two cases:

- If $k_c = 1$, a categorical column c is randomly selected from the datasets in D. We set v_c for c by sampling c from the datasets in D such that, for any dataset $d \in D$, the number of tuples in d that satisfy $c = v_c$ is larger than $ol \times |d|$. For each dataset $d \in D$, the tuples satisfying $c = v_c$ are located and merged into a dataset d_{sp}. Next, we choose k_r real-valued columns randomly from d_{sp}. For each real-valued column c in the k_r columns, the range c is set as $\min(c) \leq c \leq \max(c)$ based on d_{sp}. Here, $\min(c)$ is the minimum value for d_{sp} in c and $\max(c)$ is the maximum value of d_{sp} in c.
- If $k_c = 0$, a query contains real-valued columns only. For each dataset $d \in D$, a total of $ol \times |d|$ records are randomly sampled and merged into a dataset d_{sp}. k_r real-valued columns are randomly sampled from d_{sp}. For each real-valued column c in the k_r columns, the range of c, $\min(c) \leq c \leq \max(c)$, is set based on d_{sp}, where $\min(c)$ is the minimum value for d_{sp} in c and $\max(c)$ is the maximum value for d_{sp} in c.

After generating q using the above techniques, another query q' is generated by selecting real-valued columns from

*R indicates real-valued columns and C indicates categorical columns.

the sampled dataset d_{sp} that was used to generate q. Such a query pair (q, q') should adhere to the following condition: the minimum overlap ratio between tuples returned by each $d \in D$ is ol. The resulting query pair (q, q') is added to a query pool. When a query pool with 100 query pairs is created, the process stops. Query pairs are randomly sampled from the query pool to obtain a query set Q used for D.

By default, the number of queries $|Q|$ is set to 20. Based on the default size of D, ol is set to 5% by default to ensure the size of the sampled dataset d_{sp} is less than the number of tuples in a dataset $d \in D$, i.e., $|d_{sp}| \leq |D| \times ol \times |d| \leq |d|$, $ol \leq 0.05$. This avoids the case where a query returns all records from a dataset $d \in D$.

VI. Experiments

We conduct extensive experiments to verify that our greedy algorithm with ML-based distinctiveness estimation is:

- effective in distinctiveness estimation (Section VI-B);
- competitive with the greedy method with exact distinctiveness computation, and outperforms all other baselines under a variety of test cases and parameter settings (Section VI-C);
- efficient and scalable – the runtime is several orders of magnitude lower compared to the greedy algorithm using exact distinctiveness computation and other baselines (Section VI-D).

A. Experimental setup

Methods for comparison. Recall that the core components of our ML-based distinctiveness estimation method are (1) distinctiveness estimation and (2) updating data summaries. To demonstrate the superiority of our method, several alternative methods for these two components were explored. For component (1), we consider two implementations: `DE` – our distinctiveness estimation method as described in Alg. 2; `IRIS` – a state-of-the-art cardinality estimation method [16] for the single-query-dataset cardinality estimation (SCE) problem, which can then be aggregated across all queries to indicate the estimated distinctiveness for a query set. For component (2), we also consider two implementations: `MS` – our approach to merge data summaries, as described in Alg. 3; `IU` – a method to first find new tuples from a new dataset that does not appear in the current set of datasets selected and then update column set embeddings using new tuples, by following the equation in Sec. IV-E. This results in four methods for comparison.

- `ML-Greedy` – Our greedy algorithm using ML-based distinctiveness estimation as described in Alg. 4.
- `(DE+IU)-ML-Greedy` – A greedy algorithm that uses `DE` for distinctiveness estimation and `IU` for updating data summaries.
- `(IRIS+IU)-ML-Greedy` – A greedy algorithm that uses `IRIS` for distinctiveness estimation and `IU` for updating data summaries.
- `Exact-Greedy` – The greedy algorithm using exact distinctiveness computation as described in Alg. 1.

Parameter settings. Key parameters introduced in Sec. V are summarized in Table III. Data summaries are generated using three parameters: ξ (each column has at most ξ bits), ℓ (each row has ℓ bits), η (column set embedding size), the same as those used in `IRIS` [16]. We set $\xi = 128$, $\ell = 2048$, $\eta = 128$ for `DE` and `IRIS`, which are also the default settings for [16]. Note that, `DE` has an additional parameter, x, to indicate whether a query corresponds to at most x column sets in the query-aware dataset embedding generation and query-set embedding generation, which is set to $x = 4$.

TABLE III: Parameter settings (default value in **bold**).

Parameter	Value
sample rate s_{max}	**0.1**, 0.2, 0.3, 0.4, 0.5
$\|D\|$	10, **20**, 40, 80, 160
overlap ratio ol	1%, 3%, **5%**, 7%, 9%
$\|Q\|$	10, **20**, 40, 80, 160
$B - ratio$	0.1, 0.3, **0.5**, 0.7, 0.9

TABLE IV: Datasets used for model pretraining.

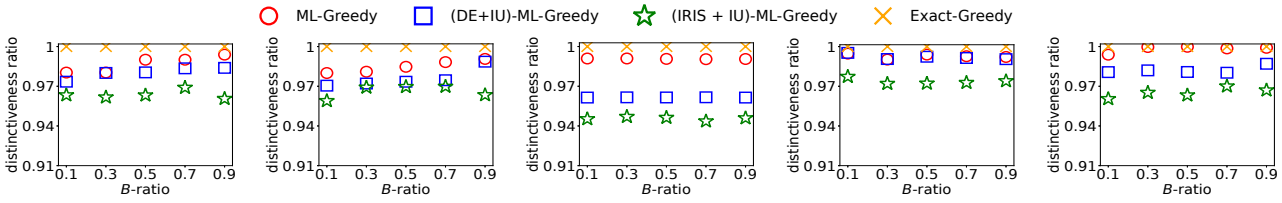
Name	# cols (R/C)*	Name	# cols (R/C)*	Name	# cols R/C
Higgs	28/0	KDD99	34/5	SUSY	19/0
PRSA	16/2	Gasmeth	18/0	Retail	5/3
Gastemp	20/0	Covtype	10/9	hepmass	29/0
Sgemm	10/4	Weather	7/0	Adult	6/8
PAMPA2	54/0	YearPred	90/0	HTsensor	10/0
Power	7/0	WECs	49/0	Census90	0/50
GasCO	18/0				

Training set generation. We pretrain our models using 19 datasets, as listed in Table IV. These datasets were released by [16] and used for pretraining cardinality estimation models.

To train models for distinctiveness estimation, we follow the approach of [16] to select 5 column sets with the highest correlation score, calculated with CORDS [29], from the training datasets and 300 randomly generated queries. For each column set, we maintain the top-10 queries by cardinality. For the column sets drawn from the same training dataset, every two column sets are combined to generate query pairs. Hence, we generate 100 query pairs for any two column pairs, to create a total of 19,000 query pairs. We then sample 80% of the query pairs as the training set, and 20% as the validation set. The batch size is set to 256. To train the model for merging data summaries, 10,000 column pairs are randomly picked from the datasets in Table IV. For each training dataset, two datasets d_1 and d_2 are sampled with a sample rate of 0.5. For each column pair in the training dataset, Eq. 2 is used to generate the column set embeddings e_1 and e_2 on d_1 and d_2, respectively. Correspondingly, a column set embedding e of the dataset created as a result of merging d_1 and d_2 by Eq. 2 is generated. We sample 80% column pairs as the training set and 20% as the validation set. The batch size is set to 256.

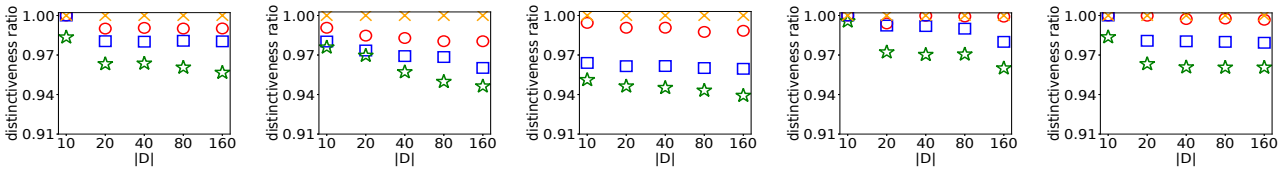
Evaluation Measures. We use the following measures:

- *Distinctiveness ratio* $\mathscr{D}/\mathscr{D}_{gt}$ is the ratio of the distinctiveness $\mathscr{D}$ calculated by a specific method to the distinctiveness $\mathscr{D}_{gt}$ calculated by the greedy algorithm using exact distinctiveness computation (i.e. the ground truth). It measures how close the performance of this method is to the ground truth.
- *Runtime* averaged over five independent runs. Due to significant differences in the runtime between each algorithm, the figures are presented using a log scale, which makes



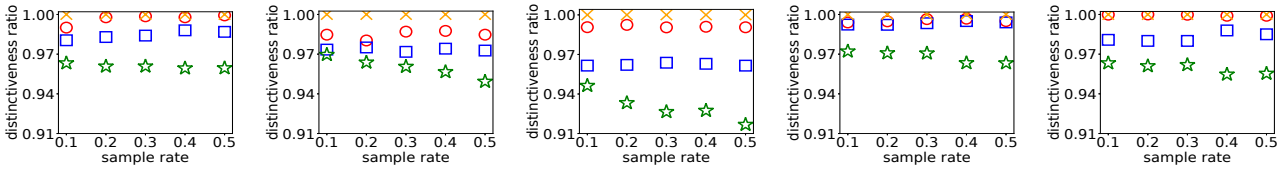
(a) TPCH (b) DMV (c) IMDB (d) Airline (e) RealEstate

Fig. 4: The impact of budget B on the distinctiveness ratio of each algorithm. (line chart is not used since cases are independent)



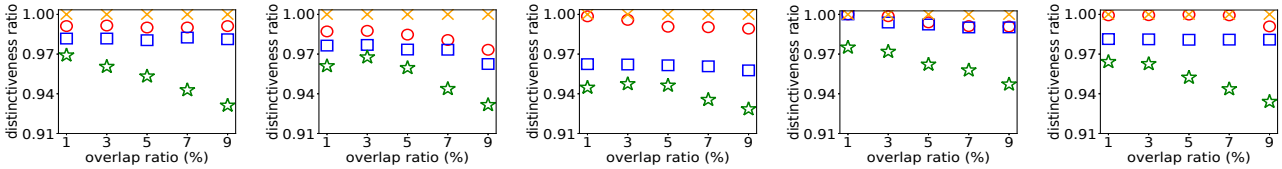
(a) TPCH (b) DMV (c) IMDB (d) Airline (e) RealEstate

Fig. 5: The impact of $|D|$ (the number of datasets) on the distinctiveness ratio of each algorithm.



(a) TPCH (b) DMV (c) IMDB (d) Airline (e) RealEstate

Fig. 6: The impact of the sampling rate upper bound $s_{\max}$ on the distinctiveness ratio of each algorithm.



(a) TPCH (b) DMV (c) IMDB (d) Airline (e) RealEstate

Fig. 7: The impact of the minimum overlap ratio ol between query pairs on the distinctiveness ratio of each algorithm.

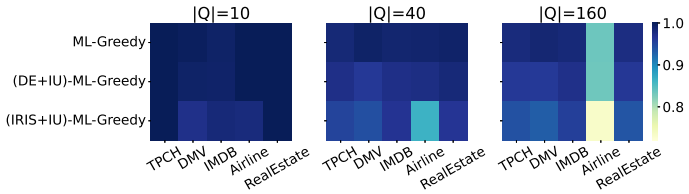


Fig. 8: The impact of $|Q|$ (the number of queries) on the distinctiveness ratio of each algorithm.

it impossible to display the confidence intervals for each point in the figures. The variation observed in each set of five runs is not significant when compared to the scale, with an average macro variance of 5.7 across all points.

Implementation We perform all experiments on a server running Red Hat Enterprise Linux with an Intel® Xeon® CPU@2.60GHz and 512GB memory, and two Nvidia Tesla P100 GPUs, each with 16GB memory. All algorithms were implemented in Python and the code is available at [38].

B. Accuracy of Distinctiveness Estimation

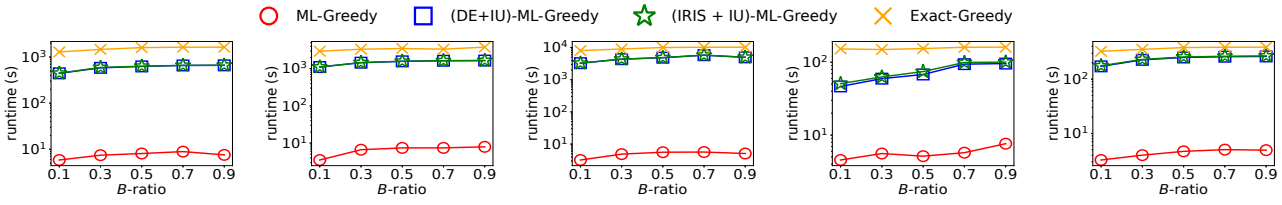
We first verify the effectiveness of our distinctiveness estimation method `DE`. As discussed in Sec. VI-A, two different implementations were considered – our distinctiveness estimation method `DE`, and the state-of-the-art cardinality estimation method `IRIS`. We compare the effectiveness using q-error, which is widely used in other cardinality estimation studies [16], [17], [21], [39]. Using the default setting, we estimate the distinctiveness for each dataset in D, and then compute the q-error for all of the datasets. Table V shows the performance comparison when using a different percentile of test cases. Observe that the q-error of `DE` is more than an order of magnitude less than `IRIS`, which shows that our method (Alg. 2) is much more effective.

TABLE V: q-error of `DE` and `IRIS` at different percentiles of test cases.

Dataset	Method	q-error				
		0th	25th	50th	75th	100th
TPCH	DE	1.602	1.661	1.669	1.682	1.801
	IRIS	11.629	11.713	11.938	11.981	12.423
DMV	DE	1.139	1.171	1.189	1.212	1.237
	IRIS	12.105	12.145	12.173	12.456	12.658
IMDB	DE	1.69	1.694	1.701	1.705	1.762
	IRIS	12.17	12.229	12.245	12.276	12.322
Airline	DE	1.725	1.840	1.85	1.858	1.876
	IRIS	12.07	12.087	12.103	12.134	12.18
RealEstate	DE	1.579	1.643	1.673	1.706	1.756
	IRIS	12.186	12.22	12.251	12.313	12.445

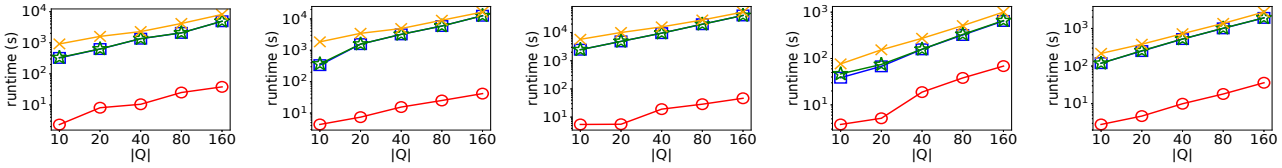
C. Effectiveness Study

Impact of the budget B. We study how varying the budget impacts the distinctiveness ratio for each algorithm, as shown in Fig. 4(a) -Fig. 4(e). Observe that: (1) When the B-ratio (budget) increases, the distinctiveness ratio of



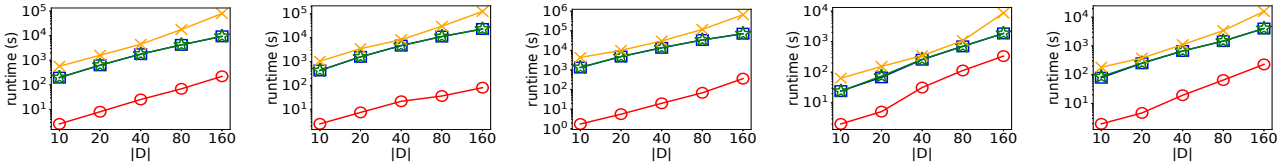
(a) TPCH (b) DMV (c) IMDB (d) Airline (e) RealEstate

Fig. 9: The impact of the budget B on the runtime of each algorithm.



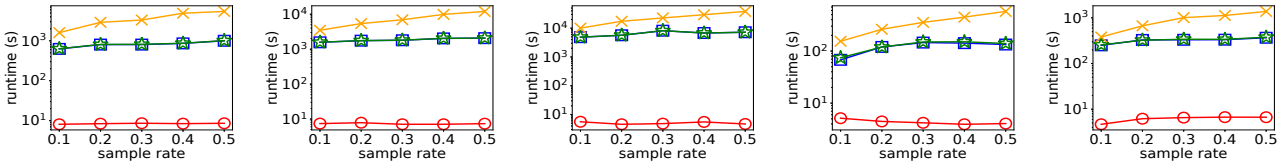
(a) TPCH (b) DMV (c) IMDB (d) Airline (e) RealEstate

Fig. 10: The impact of $|Q|$ (the number of queries) on the runtime of each algorithm.



(a) TPCH (b) DMV (c) IMDB (d) Airline (e) RealEstate

Fig. 11: The impact of $|D|$ (the number of datasets) on the runtime of each algorithm.



(a) TPCH (b) DMV (c) IMDB (d) Airline (e) RealEstate

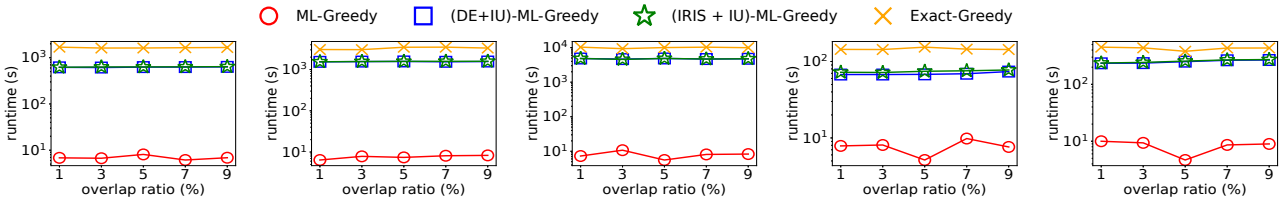
Fig. 12: The impact of the sampling rate upper bound $s_{\max}$ on the runtime of each algorithm.

`ML-Greedy` and `(DE+IU)-ML-Greedy` is 6% bigger than in `(IRIS+IU)-ML-Greedy`. Since $|d|$ is large in practice, a difference of 6% could be tens of thousands of differences in the number of tuples. This demonstrates that our distinctiveness estimation method `DE` can reliably estimate the distinctiveness, and the distinctiveness returned by `ML-Greedy` and `(DE+IU)-ML-Greedy` is very competitive with that of `Exact-Greedy`. (2) The distinctiveness ratio of `ML-Greedy` is 2% larger than `(DE+IU)-ML-Greedy`. This shows that our approach `MS` for updating data summaries is more effective.

Impact of $|D|$. We study how different sizes of datasets $|D|$ impact the distinctiveness ratio of each algorithm in Fig. 5(a) -Fig. 5(e). Observe that: (1) As $|D|$ increases, the distinctiveness ratios of `ML-Greedy` and `(DE+IU)-ML-Greedy` is greater than that of `(IRIS+IU)-ML-Greedy`. (2) The distinctiveness ratio of `(IRIS+IU)-ML-Greedy` decreases as $|D|$ increases, because more datasets are found when $|D|$ increases. This results in a larger bias when calculating the distinctiveness since `IRIS` is less accurate for distinctiveness estimation. (3) The distinctiveness ratio of `ML-Greedy` is more stable to changes in $|D|$ than `(DE+IU)-ML-Greedy`.

Impact of the sampling rate upper bound s_{max}. From the results in Fig. 6(a) -Fig. 6(e), we observe that: (1) When s_{max} increases, the distinctiveness ratio of `ML-Greedy` and `(DE+IU)-ML-Greedy` is 6% larger than `(IRIS+IU)-ML-Greedy`. (2) As s_{max} increases, the distinctiveness ratio of `(IRIS+IU)-ML-Greedy` decreases in certain cases. Since each dataset adds more tuples as s_{max} increases, the prediction capability of `IRIS` is worse than `DE`, and the distinctiveness is the product of Eq. 6 and the number of tuples in a dataset, which amplifies the errors when the number of tuples is larger. (3) Although datasets are likely to have more similar data distributions as s_{max} increases, the distinctiveness ratio of `ML-Greedy` is consistently larger than `(DE+IU)-ML-Greedy`. This shows that our approach to updating data summaries is effective regardless of whether the datasets distributions are similar.

Impact of the minimum overlap ratio ol between tuples returned by a query pair. From the results in Fig. 7(a) -Fig. 7(e), we observe that: (1) As ol increases, the distinctiveness ratio of `ML-Greedy` and `(DE+IU)-ML-Greedy` is 6% larger than `(IRIS+IU)-ML-Greedy`. (2) The distinctiveness ratio of `(IRIS+IU)-ML-Greedy` increases as ol increases, because the larger overlap ratio tends to produce more overlapping tuples for a query set. However, `IRIS` does not consider overlapping tuples across queries. (3) `ML-Greedy` always has the highest distinctiveness ratio, especially when ol is small. Recall Section V, in practice the overlap ratio is



(a) TPCH (b) DMV (c) IMDB (d) Airline (e) RealEstate

Fig. 13: The impact of the minimum overlap ratio ol between query pairs on the runtime of each algorithm.

usually small and hence the practical benefit of our method can be even greater.

Impact of the query set size $|Q|$. From Fig. 8, we observe that: (1) When $|Q|$ increases, the distinctiveness ratio of `ML-Greedy` and `(DE+IU)-ML-Greedy` is always greater than `(IRIS+IU)-ML-Greedy` – up to 13%. (2) The distinctiveness ratio of `(IRIS+IU)-ML-Greedy` decreases as $|Q|$ increases in every case. This is because more overlapping tuples tend to be returned as $|Q|$ increases while the overlap is ignored for the tuples returned by a query set in `(IRIS+IU)-ML-Greedy`. In certain cases, the distinctiveness ratio of `ML-Greedy` and `(DE+IU)-ML-Greedy` also decreases, but is still high ($> 83\%$). This shows the value of our distinctiveness estimation method `DE` when handling the overlapping tuples for a query set.

D. Efficiency and Scalability Study

Impact of the budget B. The results for various budget settings are shown in Fig. 9(a) -Fig. 9(e). Observe that: (1) When the B-ratio increases, the runtime of all algorithms increases since additional datasets are selected. However, `ML-Greedy` is at least two orders of magnitude faster than all other algorithms. This shows that `ML-Greedy`, our greedy algorithm using ML-based distinctiveness estimation, is efficient. (2) When compared against `Exact-Greedy`, which uses an exact distinctiveness computation, our distinctiveness estimation method `DE` is more than an order of magnitude faster. Also, `DE` is three times faster than `IRIS`. (3) Although `(DE+IU)-ML-Greedy` uses the same distinctiveness estimation method as `ML-Greedy`, it is at least an order of magnitude slower than `ML-Greedy`. This shows that our approach to updating data summaries, `MS`, is efficient.

Impact of the number of queries $|Q|$. We make the following observations from Fig. 10(a) -Fig. 10(e): (1) When $|Q|$ increases, the runtime in all algorithms increases because we have more queries that need to be processed when estimating distinctiveness. However, our algorithms are at least two orders of magnitude faster than the other algorithms. (2) Our distinctiveness estimation method `DE` is two orders of magnitude faster than `Exact-Greedy`, and is at least two times faster than `IRIS`. (3) Our method of updating data summaries, `MS`, is at least an order of magnitude faster than `IU`.

Impact of $|D|$. We make several observations from Fig. 11(a) -Fig. 11(e): (1) When $|D|$ increases, the runtime in all algorithms increases since the distinctiveness is being computed for more datasets. However, our greedy algorithm with ML-based distinctiveness estimation `ML-Greedy` is at least two orders of magnitude faster than the baselines, which translates to better scalability. (2) Our distinctiveness estimation method `DE` is three orders of magnitude faster than `Exact-Greedy`, and is four times faster than `IRIS`. (3) Our method of updating data summaries, `MS`, is two orders of magnitude faster than `IU`.

Impact of the sampling rate upper bound s_{max}. From Fig. 12(a) -Fig. 12(e), we observe that: (1) When s_{max} increases, our algorithm is at least two orders of magnitude faster than the baselines. (2) The runtime of `Exact-Greedy` increases since it takes more time to execute queries and calculate the marginal gain for a greater number of tuples returned from queries. In contrast, the other algorithms tend to be stable, hence confirming the advantage of using our distinctiveness estimation method. (3) Our distinctiveness estimation method `DE` is three orders of magnitude faster than `Exact-Greedy`, and is three times faster than `IRIS`. (4) Our method of updating data summaries, `MS`, is two orders of magnitude faster than `IU`.

Impact of the minimum overlap ratio ol **of tuples returned by a query pair**. From Fig. 13(a)-Fig. 13(e), we find: (1) When ol increases, the runtime of all algorithms is stable. This indicates that the number of overlapping tuples returned by queries rarely impact the runtime, even though our distinctiveness estimation method must consider overlaps. Meanwhile, `ML-Greedy` are up to four orders of magnitude faster than the competitors. (2) Our distinctiveness estimation method `DE` is at least two orders of magnitude faster than `Exact-Greedy`, and is two times faster than `IRIS`. (3) Our method of updating data summaries, `MS`, is two orders of magnitude faster than `IU`.

VII. Related Work

Datasets discovery. It can be broadly divided into basic datasets discovery and advanced datasets discovery.

Basic datasets discovery. One common way is to model a user experience over existing keyword-based information retrieval systems, where a user poses keywords and a list of relevant datasets is returned [40]. It can be used in free data portals (e.g., Google Dataset Search [41]) or commercial data market platforms (e.g., Amazon AWS Marketplace [9]). In fact, basic datasets discovery is the initial step in finding the most informative tuples and features [2].

Advanced datasets discovery. As described in Sec. I, advanced datasets discovery allows users to specify fine-grained information needs in a more expressive and concrete manner when finding desirable datasets. Recent studies [42], [43] express user information needs as specific attributes of the data. They focus on feature richness in the discovered datasets.

This may lead to too few data instances and redundant features during model training, which increases the risk of overfitting [44]. In contrast, our distinctiveness measure (see Definition 1) considers the distinctiveness of datasets, which can locate more distinct data instances. Another line of work [11], [45]–[49] expresses user information needs for a specific task (e.g., model training [11], causal inference in question answering [45]), and the usefulness of data is tested for a single task. In contrast, our distinctiveness measure provides guidance for the task-agnostic usefulness of datasets, as shown in Fig. 1; this is useful when tasks evolve over time or multiple tasks are processed at the same time.

Cardinality estimation. As mentioned in Sec. I, the problem of how to estimate the distinctiveness of a set of datasets w.r.t. a query set can be cast as the multi-query-dataset cardinality estimation (MCE) problem. Thus, we also review existing work on single-query-dataset cardinality estimation (SCE), which can be divided into query-driven methods and data-driven methods [20], [21]. Specifically, query-driven methods [23], [50]–[54] deploy discriminative models trained on a set of historically executed queries to predict the cardinality for a single query. Data-driven methods [16]–[19], [39], [55], [56] deploy generative models trained on data without query workloads. In general, query-driven methods are inflexible, especially when representative queries are not provided, and data-driven methods can achieve better performance than query-driven methods [20]. However, existing approaches for the SCE problem cannot address the MCE problem as they only estimate the cardinality of a single query over a single dataset while we estimate the cardinality of a query set over a set of datasets. Hence, it cannot address a unique challenge of the MCE problem – effectively identify the overlap in the results returned by a query set over multiple datasets.

Maximum coverage. A classical problem in computational complexity theory, the maximum coverage (MC) problem [25], aims to cover as many elements as possible in a set. Our problem can be viewed as a variant of the MC problem, which can employ a greedy strategy to produce solutions with theoretical guarantees since the objective functions are monotone and submodular. However, our problem has a unique challenge: the difficulty of efficiently yet effectively estimating the distinctiveness of multiple datasets w.r.t. a query set. Specifically, it is difficult to identify the overlap between query set results for multiple datasets – a problem that has never been addressed in existing literature [16]–[19], [23].

VIII. Conclusion

We introduced and studied the problem of maximizing distinctiveness, which involves discovering a subset of candidate datasets with the highest distinctiveness given a query set and a budget provided by a user. To address this problem, we initially proposed a greedy solution that has a theoretical guarantee but requires the exact computation of distinctiveness for every candidate dataset, which is costly and impractical. Hence, we proposed an ML-based distinctiveness estimation method to effectively estimate the distinctiveness without the need to iterate over every tuple in each dataset. To the best of our knowledge, we are the first to explore the multi-query-dataset cardinality estimation problem and successfully address the unique challenge of identifying the overlap in multiple datasets for given a query set. Our extensive evaluation on five real-world data pools confirmed the effectiveness of our proposed ML-based method in diverse settings. Notably, we achieved multiple orders of magnitude improvement in computational efficiency compared to the closest competing methods.

References

[1] F. Schomm, F. Stahl, and G. Vossen, "Marketplaces for data: an initial survey," *SIGMOD Rec.*, vol. 42, no. 1, pp. 15–26, 2013.

[2] F. Nargesian, A. Asudeh, and H. V. Jagadish, "Responsible data integration: Next-generation challenges," in *SIGMOD*, 2022, pp. 2458–2464.

[3] J. Yang, Y. He, and S. Chaudhuri, "Auto-pipeline: synthesizing complex data pipelines by-target using reinforcement learning and search," *Proc. VLDB Endow.*, vol. 14, no. 11, pp. 2563–2575, 2021.

[4] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang, "Data cleaning: Overview and emerging challenges," in *SIGMOD*, 2016, pp. 2201–2206.

[5] S. Chaudhuri, A. D. Sarma, V. Ganti, and R. Kaushik, "Leveraging aggregate constraints for deduplication," in *SIGMOD*, 2007, pp. 437–448.

[6] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *VLDB J.*, vol. 10, no. 4, pp. 334–350, 2001.

[7] Y. Li, J. Li, Y. Suhara, A. Doan, and W. Tan, "Deep entity matching with pre-trained language models," *Proc. VLDB Endow.*, vol. 14, no. 1, pp. 50–60, 2020.

[8] Y. Gong, Z. Zhu, S. Galhotra, and R. C. Fernandez, "Ver: View discovery in the wild," *CoRR*, vol. abs/2106.01543, 2021.

[9] "Amazon aws marketplace," https://aws.amazon.com/marketplace, 2022.

[10] "Snowflake data marketplace," https://www.snowflake.com/data-marketplace/, 2022.

[11] C. Chai, J. Liu, N. Tang, G. Li, and Y. Luo, "Selective data acquisition in the wild for model charging," *Proc. VLDB Endow.*, vol. 15, no. 7, pp. 1466–1478, 2022.

[12] M. C. F. de Oliveira and H. Levkowitz, "From visual data exploration to visual data mining: A survey," *IEEE Trans. Vis. Comput. Graph.*, vol. 9, no. 3, pp. 378–394, 2003.

[13] B. Lin and D. Kifer, "On arbitrage-free pricing for general data queries," *Proc. VLDB Endow.*, vol. 7, no. 9, pp. 757–768, 2014.

[14] J. Faber and L. M. Fonseca, "How sample size influences research outcomes," *Dental press journal of orthodontics*, vol. 19, pp. 27–29, 2014.

[15] Chris, "What is a sample size? a guide to market research sample sizes," 2020. [Online]. Available: https://www.starlightanalytics.com/article/what-is-a-sample-size

[16] Y. Lu, S. Kandula, A. C. König, and S. Chaudhuri, "Pre-training summarization models of structured datasets for cardinality estimation," *Proc. VLDB Endow.*, vol. 15, no. 3, pp. 414–426, 2021.

[17] Z. Yang, A. Kamsetty, S. Luan, E. Liang, Y. Duan, X. Chen, and I. Stoica, "Neurocard: One cardinality estimator for all tables," *Proc. VLDB Endow.*, vol. 14, no. 1, pp. 61–73, 2020.

[18] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig, "Deepdb: Learn from data, not from queries!" *Proc. VLDB Endow.*, vol. 13, no. 7, pp. 992–1005, 2020.

[19] R. Zhu, Z. Wu, Y. Han, K. Zeng, A. Pfadler, Z. Qian, J. Zhou, and B. Cui, "FLAT: fast, lightweight and accurate method for cardinality estimation," *Proc. VLDB Endow.*, vol. 14, no. 9, pp. 1489–1502, 2021.

[20] Y. Han, Z. Wu, P. Wu, R. Zhu, J. Yang, L. W. Tan, K. Zeng, G. Cong, Y. Qin, A. Pfadler, Z. Qian, J. Zhou, J. Li, and B. Cui, "Cardinality estimation in DBMS: A comprehensive benchmark evaluation," *Proc. VLDB Endow.*, vol. 15, no. 4, pp. 752–765, 2021.

[21] K. Kim, J. Jung, I. Seo, W. Han, K. Choi, and J. Chong, "Learned cardinality estimation: An in-depth study," in *SIGMOD*, 2022, pp. 1214–1227.

[22] S. Mehta, M. Dawande, G. Janakiraman, and V. S. Mookerjee, "How to sell a dataset?: Pricing policies for data monetization," in *EC*, 2019, p. 679.

[23] S. Hasan, S. Thirumuruganathan, J. Augustine, N. Koudas, and G. Das, "Deep learning models for selectivity estimation of multi-attribute queries," in *SIGMOD*, 2020, pp. 1035–1050.

[24] N. Weir, P. Utama, A. Galakatos, A. Crotty, A. Ilkhechi, S. Ramaswamy, R. Bhushan, N. Geisler, B. Hättasch, S. Eger, U. Çetintemel, and C. Binnig, "Dbpal: A fully pluggable NL2SQL training pipeline," in *SIGMOD*, 2020, pp. 2347–2361.

[25] V. Nagarajan, "Approximation & online algorithms," *http://viswa.engin.umich.edu/wp-content/uploads/sites/169/2021/02/greedy.pdf*, 2021.

[26] S. Khuller, A. Moss, and J. Naor, "The budgeted maximum coverage problem," *Inf. Process. Lett.*, vol. 70, no. 1, pp. 39–45, 1999.

[27] Y. E. Ioannidis, "The history of histograms (abridged)," in *VLDB*. Morgan Kaufmann, 2003, pp. 19–30.

[28] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[29] I. F. Ilyas, V. Markl, P. J. Haas, P. Brown, and A. Aboulnaga, "CORDS: automatic discovery of correlations and soft functional dependencies," in *SIGMOD*, 2004, pp. 647–658.

[30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[31] F. Nargesian, A. Asudeh, and H. V. Jagadish, "Tailoring data source distributions for fairness-aware data integration," *Proc. VLDB Endow.*, vol. 14, no. 11, pp. 2519–2532, 2021.

[32] K. H. Tae and S. E. Whang, "Slice tuner: A selective data acquisition framework for accurate and fair machine learning models," in *SIGMOD*, 2021, pp. 1771–1783.

[33] "Tpc-h benchmark," http://www.tpc.org/tpch/, 2022.

[34] "State of new york vehicle, snowmobile, and boat registrations," https://catalog.data.gov/dataset/vehicle-snowmobile-and-boat-registrations, 2022.

[35] "Imdb dataset," https://https://homepages.cwi.nl/~boncz/job/imdb.tgz, 2022.

[36] "Airline dataset," https://relational.fit.cvut.cz/dataset/Airline, 2022.

[37] M. Li, Z. Bao, T. Sellis, S. Yan, and R. Zhang, "Homeseeker: A visual analytics system of real estate data," *J. Vis. Lang. Comput.*, vol. 45, pp. 1–16, 2018.

[38] "Source code," https://gitfront.io/r/user-3680909/fuDbUGQtSjjF/um/, 2023.

[39] Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, X. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan, and I. Stoica, "Deep unsupervised cardinality estimation," *Proc. VLDB Endow.*, vol. 13, no. 3, pp. 279–292, 2019.

[40] A. Chapman, E. Simperl, L. Koesten, G. Konstantinidis, L. Ibáñez, E. Kacprzak, and P. Groth, "Dataset search: a survey," *VLDB J.*, vol. 29, no. 1, pp. 251–272, 2020.

[41] D. Brickley, M. Burgess, and N. F. Noy, "Google dataset search: Building a search engine for datasets in an open web ecosystem," in *WWW*, 2019, pp. 1365–1375.

[42] Y. Li, H. Sun, B. Dong, and W. H. Wang, "Cost-efficient data acquisition on online data marketplaces for correlation analysis," *Proc. VLDB Endow.*, vol. 12, no. 4, pp. 362–375, 2018.

[43] A. Bogatu, A. A. A. Fernandes, N. W. Paton, and N. Konstantinou, "Dataset discovery in data lakes," in *ICDE*, 2020, pp. 709–720.

[44] D. Zha, Z. P. Bhat, K. Lai, F. Yang, Z. Jiang, S. Zhong, and X. Hu, "Data-centric artificial intelligence: A survey," *CoRR*, vol. abs/2303.10158, 2023.

[45] S. Galhotra, Y. Gong, and R. C. Fernandez, "Metam: Goal-oriented data discovery," in *ICDE*, 2023, pp. 2780–2793.

[46] Y. Li, X. Yu, and N. Koudas, "Data acquisition for improving machine learning models," *Proc. VLDB Endow.*, vol. 14, no. 10, pp. 1832–1844, 2021.

[47] Y. Chen, Y. Shen, and S. Zheng, "Truthful data acquisition via peer prediction," in *NeurIPS*, 2020, pp. 18 194–18 204.

[48] A. Agarwal, M. A. Dahleh, and T. Sarkar, "A marketplace for data: An algorithmic solution," in *EC*, 2019, pp. 701–726.

[49] J. Yoon, S. Ö. Arik, and T. Pfister, "Data valuation using reinforcement learning," in *ICML*, vol. 119, 2020, pp. 10 842–10 851.

[50] A. Dutt, C. Wang, A. Nazi, S. Kandula, V. R. Narasayya, and S. Chaudhuri, "Selectivity estimation for range predicates using lightweight models," *Proc. VLDB Endow.*, vol. 12, no. 9, pp. 1044–1057, 2019.

[51] A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper, "Learned cardinalities: Estimating correlated joins with deep learning," in *CIDR*, 2019.

[52] J. Sun and G. Li, "An end-to-end learning-based cost estimator," *Proc. VLDB Endow.*, vol. 13, no. 3, pp. 307–319, 2019.

[53] C. Wu, A. Jindal, S. Amizadeh, H. Patel, W. Le, S. Qiao, and S. Rao, "Towards a learning optimizer for shared clouds," *Proc. VLDB Endow.*, vol. 12, no. 3, pp. 210–222, 2018.

[54] Y. Park, S. Zhong, and B. Mozafari, "Quicksel: Quick selectivity learning with mixture models," in *SIGMOD*, 2020, pp. 1017–1033.

[55] Z. Wu, A. Shaikhha, R. Zhu, K. Zeng, Y. Han, and J. Zhou, "Bayescard: Revitilizing bayesian frameworks for cardinality estimation," *CoRR*, vol. abs/2012.14743, 2020.

[56] K. Tzoumas, A. Deshpande, and C. S. Jensen, "Efficiently adapting graphical models for selectivity estimation," *VLDB J.*, vol. 22, no. 1, pp. 3–27, 2013.