Fast Diversified Top-k Rule Discovery via User-Guided Embeddings

Ziyan Han, Wanjia Chen, Yunpeng Han, Rui Mao, Jianbin Qin*

***Abstract*—Rule discovery is a fundamental task in data analysis, with broad applications in data cleaning, knowledge extraction, and decision making. However, existing methods often generate a large number of functionally redundant rules, with a high time cost. To address this, a recent line of work, the first to introduce diversified top-k rule discovery, aims to identify a set of top-ranked rules that are both relevant and diverse. Despite this advancement, it still suffers from high user interaction overhead, computational inefficiency, and the inability to handle a common scenario of selecting a diverse subset from an existing rule set.**

In this paper, we propose a user-friendly and efficient framework for diversified top-k rule discovery. As a testbed, we consider Entity Enhancing Rules (REEs), which subsume common association rules and data quality rules as special cases. Our method allows users to specify lightweight preference templates, which are used to train a correlation model that captures user preferences and generates subjective embeddings for predicates and rules. Based on these embeddings, we define an objective function to jointly measure the relevance and diversity of rules in a unified vector space; moreover, we formulate and study two key problems: (i) selecting diversified top-k rules from an existing redundant rule set, and (ii) discovering diversified top-k rules directly from raw data. We prove that both problems are intractable and propose effective algorithms; in particular, the second problem is more challenging and thus we further optimize its solution with carefully designed pruning strategies and parallel optimization. Extensive evaluation on real-world datasets demonstrates that our algorithms consistently identify top-ranked relevant and diverse rules, achieving an average 14.4× speedup (up to 35.57×) over the state-of-the-art method.

***Index Terms*—Data mining, rule discovery, top-k, diversified.**

I. Introduction

In the realm of data-driven decision making, it is critical to extract meaningful patterns from datasets. As an interpretable form of knowledge representation, rules are empirically proven to be useful for representing complex patterns hidden in data across various scenarios, *e.g.,* finance [1], healthcare [2] and social media analytics [3]. In particular, rules can be used to improve the quality of data, *e.g.,* entity resolution (ER) rules can be used for deduplication by identifying tuples that refer to the same entity, and conflict resolution (CR) rules can be used to resolve conflicts in/among entities. There has been a large body of works on rules, *e.g.,* functional dependencies (FDs [4]), conditional functional dependencies (CFDs [5]), matching dependencies (MDs [6]), denial constraints (DCs [7]), and entity enhancing rules (REEs [8], [9]).

To make practical use of rules, it is essential to efficiently discover effective rules from real-world data. Traditional rule discovery approaches primarily rely on objective metrics such as *support* and *confidence* to measure rule relevance and identify all valid rules, which often leads to excessive irrelevant rules. While prior work, such as [10], was the first to incorporate subjective metrics into top-k relevant rule discovery, enabling user preferences to guide the ranking process, it requires users to label preferred rules through pairwise comparisons, which can be difficult or even infeasible in certain scenarios. Furthermore, the discovered rules may still be functionally similar or redundant, limiting their practical utility.

Example 1. *Table I illustrates a toy example with six bi-variable rules, discovered from a relation* Hospital *[10]–[14]. Each rule is denoted as $\varphi : X \rightarrow p_o$, where X is a conjunction of predicates and p_o is a predicate implied by X ($p_o \notin X$). When users are presented with rules φ_2 and φ_3 for pairwise comparison, it is challenging to determine which one is preferable, since the predicates in their X are different. Additionally, rules $\varphi_1 \sim \varphi_3$ (resp. $\varphi_4 \sim \varphi_6$) indicate the association between the predicates in X and the HospitalType-related (resp. EmergencyService-related) predicate p_o; we observe that rules $\varphi_1 \sim \varphi_3$ (resp. $\varphi_4 \sim \varphi_6$) are functionally similar, and thus two rules alone (selecting one from $\varphi_1 \sim \varphi_3$ and $\varphi_4 \sim \varphi_6$, respectively) may suffice in some application scenarios (*e.g., *missing value imputation on attributes in p_o).*

To mitigate this, subsequent work [15] formulated and explored diversified top-k rule discovery as a bi-objective optimization problem considering both relevance and diversity. It made the first attempt to discover diversified top-ranked, instead of all, rules from a dataset, leveraging a novel error detection-based relevance model and four proposed diversity measures. While this approach offers a promising direction, our experimental evaluation reveals that it still suffers from the following three limitations:

(1) It requires users to provide a clean version of each given dataset as input for training a relevance model, which is often difficult or even infeasible in many real-world scenarios. Constructing such clean datasets is non-trivial, as users may lack the necessary expertise, prior rules, or domain knowledge for accurate data cleaning. Moreover, relying on existing data cleaning tools may introduce additional errors, thereby compromising the quality of the labeled data used for training and causing error propagation in downstream tasks;

Ziyan Han and Wanjia Chen are with Shenzhen University, Shenzhen 518060, China (e-mail: hanzy@szu.edu.cn; chenwanjia@email.szu.edu.cn).

Yunpeng Han is with Louisiana State University, Baton Rouge, LA 70803, USA (e-mail: yhan28@lsu.edu).

Rui Mao and Jianbin Qin are with SICS, Shenzhen University, Shenzhen 518060, China (e-mail: mao@szu.edu.cn; qinjianbin@szu.edu.cn).

*Corresponding author

TABLE I
A TOY EXAMPLE OF A FEW REDUNDANT RULES DISCOVERED ON HOSPITAL VIA EXISTING RULE DISCOVERY APPROACHES

rid	rule
φ_1	Hospital(t) $\wedge$ Hospital(s) $\wedge$ t.City $=$ s.City $\wedge$ t.EmergencyService $=$ s.EmergencyService $\rightarrow$ t.HospitalType $=$ s.HospitalType
φ_2	Hospital(t) $\wedge$ Hospital(s) $\wedge$ t.State $=$ s.State $\wedge$ t.EmergencyService $=$ s.EmergencyService $\rightarrow$ t.HospitalType $=$ s.HospitalType
φ_3	Hospital(t) $\wedge$ Hospital(s) $\wedge$ t.CountyName $=$ s.CountyName $\wedge$ t.HospitalOwner $=$ s.HospitalOwner $\rightarrow$ t.HospitalType $=$ s.HospitalType
φ_4	Hospital(t) $\wedge$ Hospital(s) $\wedge$ t.PhoneNumber $=$ s.PhoneNumber $\rightarrow$ t.EmergencyService $=$ s.EmergencyService
φ_5	Hospital(t) $\wedge$ Hospital(s) $\wedge$ t.State $=$ TX $\wedge$ s.State $=$ TX $\wedge$ t.HospitalOwner $=$ s.HospitalOwner$\wedge$ t.HospitalOwner $=$ Voluntary non-profit-other $\rightarrow$ t.EmergencyService $=$ s.EmergencyService
φ_6	Hospital(t) $\wedge$ Hospital(s) $\wedge$ t.City $=$ s.City $\wedge$ t.Sample $=$ s.Sample$\wedge$ t.Condition $=$ Pneumonia $\wedge$ s.Condition $=$ Pneumonia $\rightarrow$ t.EmergencyService $=$ s.EmergencyService

(2) Most existing rule discovery methods (*e.g.,* [15]) evaluate rule relevance using traditional measures such as support and confidence. While widely adopted, these metrics are computationally expensive, particularly on large datasets, and often become the primary bottleneck, fundamentally limiting the scalability and efficiency; and

(3) The scope of the approach is limited to discovering diversified top-k rules directly from raw datasets. However, in many practical applications, a candidate rule set may already be available (*e.g.,* generated by an initial rule discovery phase), from which a diversified top-k subset needs to be selected. The method in [15] does not address this scenario, thereby restricting its applicability in modular or multi-stage rule mining pipelines.

This paper proposes a user-friendly and efficient framework for diversified top-k rule discovery that addresses the limitations of prior work. An overview of our framework is shown in Fig. 1. Rather than requiring clean data or intensive pairwise labeling, we design an intuitive interaction mechanism to effectively capture user preferences. Specifically, users only need to provide lightweight preference templates for their desired rules prior to rule discovery, thereby avoiding the heavy burden of extensive labeling work. From these templates, we train a correlation model that captures user preferences and generates subjective embeddings for predicates and rules. Both relevance and diversity are defined in a unified vector space based on these user-guided predicate/rule embeddings. This embedding-based formulation not only enables instant scoring without costly support/confidence calculations, which in our framework are used only as pre-filtering conditions, but also opens a promising direction for fast and scalable evaluation of rule quality. Following this idea, we develop two algorithms: (i) TopkDivSelector, for selecting diversified top-k rules from an existing rule set; and (ii) PTopkDivFinder, for discovering diversified top-k rules from scratch. The discovery process is further accelerated by carefully designed pruning and parallelization strategies.

We use entity enhancing rules (REEs) [8], [9] as our testbed. REEs are a powerful class of rules used in an industrial system Rock, which subsume common association rules and data quality rules, such as FDs, CFDs, MDs and DCs, as special cases.

Contributions & Organization. The main contributions and structure of this paper are presented as follows.

(1) User-guided embeddings and objective function (Section III). We propose a novel and low-cost interaction mechanism in which users specify preference templates for their desired rules. These templates are used to train a correlation model $\mathcal{M}_{\mathsf{corr}}$ that captures user preferences and generates corresponding user-guided embeddings for predicates and rules. We define both relevance and diversity measures in a unified vector space, and further formulate an embedding-based objective function that jointly optimizes relevance and diversity.

(2) Selecting the diversified top-k rules from a given rule set (Section IV). We formulate a problem that, given a set Σ of discovered rules, a set of user-specified preference templates, a set $\mathcal{A}$ of attributes of users' interests, and a set C of user-accumulated rules, selects a set Σ_D of diversified top-k rules from Σ such that rules in Σ_D (a) are relevant to $\mathcal{A}$, (b) unexpected with respect to C, and at the same time, (c) diverse from each other. We show that this problem is intractable. This said, we propose an efficient approach, denoted by TopkDivSelector; we prove that it takes time $\mathcal{O}(kn(|C| + |\mathcal{A}|))$ with *approximation ratio* 2.

(3) Discovering the diversified top-k rules from scratch (Section V). We formulate and explore another problem that, given the set of preference templates, a set $\mathcal{A}$ of attributes of users' interests, and a set C of user-accumulated rules, discovers a set Σ of diversified top-k rules from scratch such that rules in Σ (a) are relevance to $\mathcal{A}$, (b) unexpected with respect to C, and (c) diverse from each other. Unfortunately, this problem is also intractable. Nevertheless, we propose an effective approach, denoted by PTopkDivFinder; in particular, we greatly reduce the time cost of the proposed approach via some carefully designed pruning strategies and parallel optimization.

(4) Experimental evaluation (Section VI). We conducted extensive experiments on real-world datasets to evaluate the performance of the proposed approaches in tackling the two studied problems. We empirically find the following. (a) Based on $\mathcal{M}_{\mathsf{corr}}$, TopkDivSelector and PTopkDivFinder are able to obtain a set of diversified top-k rules, consistently outperforming the baselines on normalized $F(\cdot)$ and rule coverage $rCov(\cdot)$. (b) TopkDivSelector is efficient, it is at least 3060X faster than finding the optimal results, on average. (c) PTopkDivFinder is efficient, *e.g.,* it is $>100\times$ faster than methods that mine the entire rule set; with a user-guided embedding-based objective function, it speeds up the method of [15] by 14.4$\times$ on average, up to 35.57$\times$. (d) The proposed pruning strategies are effective, *e.g.,* with optimization, PTopkDivFinder improves by 24.1$\times$ on average, up to 54$\times$.

In addition, we present the preliminaries in Section II, discuss the related work in Section VII, and conclude this paper in Section VIII.

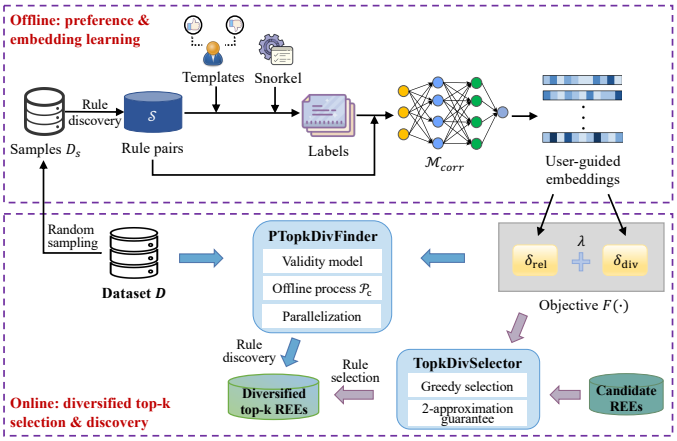


Fig. 1. Overview of user-guided diversified top-k rule discovery framework.

II. Preliminaries

In this section, we introduce relation, predicates, entity enhancing rules (REEs) [8], [9], and some constraints of rules.

Definition 1 (Relation). *A database schema* $\mathcal{R} = (R_1, \ldots, R_m)$ *consists of multiple relation schema* R_j*, where each* $R_j = R(A_1 : \tau_1, \ldots, A_n : \tau_n)$ *contains some attributes* A_i *of types* τ_i*. A database instance* $\mathcal{D}$ *of* $\mathcal{R}$ *is* $(D_1, \ldots, D_m)$*, where each* D_j *is a relation instance (or simple relation) of* R_j*.*

Definition 2 (Predicates). *Following [8], [9], we study the following types of predicates over a schema* $\mathcal{R}$*,*

$$p ::= R(t) \mid t.A \oplus c \mid t.A \oplus s.B \mid \mathcal{M}(t[\bar{A}], s[\bar{B}]),$$

where $R(t)$ *is a* relation atom *over schema* $\mathcal{R}$*,* $\oplus$ *is the operator* $=$ *or* $\neq$*,* c *is a constant,* $t.A$ *(resp.* $s.B$*) is the value on attribute* A *(resp.* B*) of a tuple* t *(resp.* s*),* $t.A \oplus c$ *(resp.* $t.A \oplus s.B$*) is a constant (resp. non-constant) predicate, and* $\mathcal{M}(t[\bar{A}], s[\bar{B}])$ *is an ML predicate checking whether the values of attributes* $\bar{A}$ *in tuple* t *are similar to those of* $\bar{B}$ *in* s*.*

Definition 3 (REEs). *An* entity enhancing rule *(*REE*) [8], [9]* φ *over a schema* $\mathcal{R}$ *is defined as*

$$\varphi : X \rightarrow p_o,$$

where X *is the* precondition *that contains a conjunction of predicates over* $\mathcal{R}$*, and* p_o *(*$p_o \notin X$*) is the single-predicate* consequence*; all tuple variables in* φ *are bounded in* X*.*

Semantics. Consider an instance $\mathcal{D}$ of $\mathcal{R}$. A valuation h of φ is a mapping that instantiates $t \in R(t)$ with a tuple in a relation $D \in \mathcal{D}$. We denote $h \models p$ when h satisfies a predicate p (*e.g.,* p_o); we denote $h \models X$ (resp. $h \models \varphi$) if $h \models p'$ holds for each $p' \in X$ (resp. $X \cup p_o$). Moreover, we denote $\mathcal{D} \models \varphi$ if $h \models \varphi$ holds for all valuations h of the tuple variables of φ in $\mathcal{D}$. Further, we denote $\mathcal{D} \models \Sigma$ if $\mathcal{D}$ *satisfies* a set Σ of REEs, *i.e.,* $\mathcal{D} \models \varphi$ holds for all $\varphi \in \Sigma$.

Due to space limitations, we briefly discuss the basic idea of support and confidence, and recommend interested readers to existing works (*e.g.,* [8], [9], [15]) for more details.

TABLE II
Notations

symbols	description
$\mathcal{D}$, $\mathcal{D}_s$	dataset; samples of the dataset
φ	a rule expression $X \rightarrow p_o$
Σ, Σ_{all}	top-k diversified rule set; set of all minimal rules
$\delta_{\mathsf{rel}}(\varphi)$, $\delta_{\mathsf{div}}(\Sigma)$	relevance of rule φ; diversity of rule set Σ
$\delta_{\mathsf{dist}}(\varphi_i, \varphi_j)$, $\delta^{\mathsf{w}}_{\mathsf{dist}}(\varphi_i, \varphi_j)$	pairwise distance; weighted pairwise distance
$F(\Sigma)$	objective score for rule set Σ
λ	trade-off between relevance and diversity
$\mathcal{A}$, C	user-selected attributes; user-accumulated rule set
σ, γ	support threshold; confidence threshold
k	target number of selected or discovered rules

Definition 4 (Support). *Denote by* $\mathsf{supp}(\varphi, \mathcal{D})$ *the* support *of a rule* φ *on an instance* $\mathcal{D}$*, which is the size of all tuple pairs in* $\mathcal{D}$ *satisfying the rule* φ*.*

Definition 5 (σ-frequent). *For a threshold* σ*, we say a rule* φ *is* σ-frequent *on an instance* $\mathcal{D}$ *if* $\mathsf{supp}(\varphi, \mathcal{D}) \geq \sigma$*.*

Definition 6 (Confidence). *Denote by* $\mathsf{conf}(\varphi, \mathcal{D})$ *the* confidence *of a rule* $\varphi : X \rightarrow p_o$ *on an instance* $\mathcal{D}$*;* $\mathsf{conf}(\varphi, \mathcal{D}) = n_2/n_1$*, where* n_1 *(resp.* n_2*) is the size of all tuple pairs in* $\mathcal{D}$ *satisfying the precondition* X *(resp.* $X \cup p_o$*).*

Definition 7 (γ-confident). *For a threshold* γ*, we say a rule* φ *is* γ-confident *on an instance* $\mathcal{D}$ *if* $\mathsf{conf}(\varphi, \mathcal{D}) \geq \gamma$*.*

Definition 8 (Minimality). *For a rule* $\varphi : X \rightarrow p_o$*, we say it is* minimal *on an instance* $\mathcal{D}$ *if (a)* φ *is* σ*-frequent, (b)* φ *is* γ*-confident, and (c) there exists no other rules* $\varphi' : X' \rightarrow p_o$ *where* φ' *satisfies conditions (a)∼(b) above and* $X' \subseteq X$*.*

The key notations of this paper are summarized in Table II.

III. User-guided embedding and objective function

In this section, we first present how user-provided preference templates are used to train a correlation model that learns user preferences and generates user-guided embeddings of predicates and rules (Section III-A). Based on the learned embeddings, we then define an objective function that jointly optimizes relevance and diversity (Section III-B).

A. User preference–driven embedding learning

We design a correlation model $\mathcal{M}_{\mathsf{corr}}$ that takes a set $\mathcal{S}$ of labelled rule pairs as input and outputs the embeddings of predicates and rules in $\mathcal{S}$, where each rule pair in $\mathcal{S}$ is automatically labelled by a user-friendly labeling scheme that incorporates user preferences. We present the model structure in Section III-A1, followed by the labeling scheme in Section III-A2.

1) Model structure: Following [10], [15], we design $\mathcal{M}_{\mathsf{corr}}$ as a lightweight model, where $\mathcal{M}_{\mathsf{corr}}$ utilizes a Siamese network architecture [16]. Rather than asking users to label a large number of rule pairs as in [10] or provide a clean version of the given dataset as in [15], we only need users to provide certain preference templates of the desired rules (see Section III-A2). The $\mathcal{M}_{\mathsf{corr}}$ consists of two key components: rule embeddings and loss function.

Rule embeddings. Given a rule $\varphi : X \rightarrow p_o$, we denote E_p the embedding of a predicate p (*e.g.,* p_o), where $E_p \in \mathbb{R}^{d\times 1}$; for the precondition X, following [10], denoted by E_X^T, we obtain its embedding by concatenating the embeddings of its predicates $p \in X$, *i.e.,* $E_X^T = \rho(\sum_{p\in X} \Phi(E_p))$, where $E_X \in \mathbb{R}^{d\times 1}$, and ρ and Φ indicate two linear layers without activation functions. We embed φ as $E_\varphi \in \mathbb{R}^{2d\times 1}$ by concatenating the embeddings of X and p_o, *i.e.,* $E_\varphi = [E_X^T; E_{p_o}^T]^T$.

Loss function. We adopt contrastive loss function [17] to minimize the distance between embeddings of functionally similar rule pairs (with labels 1), and at the same time maximize the distance between embeddings of dissimilar rule pairs (with labels 0). For a rule pair (φ_i, φ_j), we denote its label as y (= $\{0, 1\}$). The loss $\mathcal{L}$ for the (φ_i, φ_j) is defined as:

$$\mathcal{L}(\varphi_i, \varphi_j; y) = y \cdot \frac{1}{2} d(\varphi_i, \varphi_j)^2 + (1-y) \cdot \frac{1}{2} \max(0, m - d(\varphi_i, \varphi_j))^2, \quad (1)$$

where $d(\varphi_i, \varphi_j) = ||E_{\varphi_i}, E_{\varphi_j}||_2$ is the Euclidean distance between the embeddings of φ_i and φ_j, and m is a margin parameter denoting the minimum distance of dissimilar rule pairs.

Note that the loss function treats each rule pair equally. If users want to give more weights $w(\varphi_i, \varphi_j)$ on some rule pairs (φ_i, φ_j) in $\mathcal{S}$, one can optimize $\mathcal{L}(\varphi_i, \varphi_j, y)$ as $w(\varphi_i, \varphi_j) \cdot \mathcal{L}(\varphi_i, \varphi_j, y)$. It enables $\mathcal{M}_{\mathsf{corr}}$ to prioritize learning from more critical rule pairs; we leave this optimization as future work.

2) Labeling scheme: We first discuss how to obtain a set $\mathcal{S}$ of unlabeled rule pairs. Then, we define the required form of the user's preference templates for labeling the rule pairs in $\mathcal{S}$, followed by some proposed function templates for reference. Note that we assume the subjective criterion of users is to determine whether a rule pair in $\mathcal{S}$ is similar or dissimilar.

Rule pair generation. Consider an instance $\mathcal{D}$ over schema $\mathcal{R}$. We first randomly extract a sample data $\mathcal{D}_s$ with a medium-size (*e.g.,* 20K tuples) from $\mathcal{D}$, on which we discover a set $\mathcal{S}$ of valid rule pairs via the commonly used rule discovery algorithms (*e.g.,* [14]) for subsequent labeling.

Criterion form. Given a rule pair (φ_i, φ_j) in $\mathcal{S}$, we define the user's subjective criterion as a function f that takes the (φ_i, φ_j) as input and outputs a scalar in $[0, 1]$. A user may define multiple subjective functions; we denote $\mathcal{F}$ the set of all functions/criteria defined by users. Intuitively, given a function $f \in \mathcal{F}$ and the (φ_i, φ_j), we say that φ_i and φ_j are similar (resp. dissimilar) if $f(\varphi_i, \varphi_j) > 0.5$ (resp. ≤ 0.5), where the pair is automatically labeled as 1 (resp. 0). Given the label of each $f \in \mathcal{F}$ on (φ_i, φ_j), one can decide its final label by using *e.g.,* majority voting. In this paper, we leverage Snorkel [18], an automatic labeling tool using weak supervision, to aggregate the labels from each criterion.

Preference templates. To guide users in defining their criteria following the form above, we provide five function templates as references.

Predicate overlap. Given a rule pair (φ_i, φ_j), we say φ_i and φ_j are similar (resp. dissimilar) *w.r.t.* predicates if they share a large (resp. small) number of predicates, *i.e.,* $f(\varphi_i, \varphi_j) = \frac{|\mathcal{P}_i \cap \mathcal{P}_j|}{|\mathcal{P}_i \cup \mathcal{P}_j|} > 0.5$ (resp. ≤ 0.5), where $\mathcal{P}_i$ (resp. $\mathcal{P}_j$) denotes the set of all predicates in φ_i (resp. φ_j).

Covered tuples overlap. Given a rule pair (φ_i, φ_j), we say they are similar (resp. dissimilar) *w.r.t.* covered tuples if their valuations in $\mathcal{D}$ cover a large (resp. small) number of tuples, *i.e.,* $f(\varphi_i, \varphi_j) = \frac{|\mathsf{cov}(\varphi_i) \cap \mathsf{cov}(\varphi_j)|}{|\mathsf{cov}(\varphi_i) \cup \mathsf{cov}(\varphi_j)|} > 0.5$ (resp. ≤ 0.5), where $\mathsf{cov}(\varphi_i) = \{t | t \in h \wedge h \models \varphi_i\}$ denotes the tuples covered in all valuations h of tuple variables of φ_i in $\mathcal{D}$; similar for $\mathsf{cov}(\varphi_j)$.

Same consequence. Given a rule pair (φ_i, φ_j), we say φ_i and φ_j are similar (resp. dissimilar) *w.r.t.* consequence if they contain the same (resp. different) consequence, *i.e.,* $f(\varphi_i, \varphi_j) = \mathbb{1}[p_o^i$ equals to $p_o^j] = 1$ (resp. 0), where p_o^i (resp. p_o^j) is the consequence of the rule φ_i (resp. φ_j).

Semantic correlation. Given a rule pair (φ_i, φ_j), we say φ_i and φ_j are semantically similar (resp. dissimilar) if they are judged to be similar (resp. dissimilar) by a reliable language model LM, *i.e.,* $f(\varphi_i, \varphi_j) = \mathsf{LM}(\varphi_i, \varphi_j) > 0.5$ (resp. ≤ 0.5), where $\mathsf{LM}(\varphi_i, \varphi_j)$ is the correlation score between φ_i and φ_j output by the LM (*e.g.,* SentenceBert [19]).

Structural correlation. We borrow the idea from *graph neural network* [20] to catch the semantics inherent in the underlying data. To be specific, we convert $\mathcal{D}_\mathsf{s}$ into a graph G_s where attributes and values in $\mathcal{D}_\mathsf{s}$ refer to nodes in G_s, and their relationships are depicted as edges, such that we can employ any reliable graph embedding tools (*e.g.,* EmbDI [21]) to learn the embeddings of each node in G_s. Given a rule pair (φ_i, φ_j), we say φ_i and φ_j are structurally similar (resp. dissimilar) if the graph embeddings of φ_i and φ_j have small (resp. large) distance, *i.e.,* $f(\varphi_i, \varphi_j) = 1/(1 + ||E_{\varphi_i}, E_{\varphi_j}||_2) > 0.5$ (resp. ≤ 0.5), where E_{φ_i} denotes the graph embeddings of φ_i; similar for E_{φ_j}.

In a typical configuration, a set $\mathcal{S}$ of a few thousand (*e.g.,* 3K) rule pairs is constructed from a medium-sized sample $\mathcal{D}_\mathsf{s}$ (*e.g.,* 20K tuples) drawn from the given dataset. Labeling $\mathcal{S}$ with a small number of preference templates (*e.g.,* 5) is generally sufficient to provide the training data for $\mathcal{M}_{\mathsf{corr}}$.

Example 2. *We illustrate how preference templates generate training data for $\mathcal{M}_{\mathsf{corr}}$. Consider a similar pair of rules (φ_1, φ_2) from Table I and the five preference templates. Let $\mathcal{P}_1$ (resp. $\mathcal{P}_2$) be the set of all predicates in φ_1 (resp. φ_2), including the consequence. The* predicate overlap *template computes $f_{\mathsf{overlap}}(\varphi_1, \varphi_2) = \frac{|\mathcal{P}_1 \cap \mathcal{P}_2|}{|\mathcal{P}_1 \cup \mathcal{P}_2|}$ and, since the two rules share many predicates, we have $f_{\mathsf{overlap}}(\varphi_1, \varphi_2) > 0.5$, so this template outputs a positive weak label ($\ell_{\mathsf{overlap}} = 1$). Together with the other four templates, we may obtain weak labels $(\ell_{\mathsf{overlap}}, \ell_{\mathsf{cov}}, \ell_{\mathsf{cons}}, \ell_{\mathsf{sema}}, \ell_{\mathsf{stru}}) = (1, 0, 1, 1, 0)$ for (φ_1, φ_2), and $(0, 1, 0, 0, 0)$ for a dissimilar pair (φ_1, φ_5). Snorkel aggregates these weak labels into a single label $y \in \{0, 1\}$, yielding $y = 1$ for (φ_1, φ_2) and $y = 0$ for (φ_1, φ_5), which constitute positive and negative training pairs for $\mathcal{M}_{\mathsf{corr}}$.*

User effort. For experienced users, authoring a preference template is lightweight; each template is a small function that can be written or adapted in a few minutes. For novices, we enable five built-in templates by default, covering both semantic (*predicate overlap*, *same consequence*, *semantic correlation*) and structural relations (*covered tuples overlap*, *structural*

correlation). Thus, the pipeline can run end-to-end without user-authored templates; custom templates are only needed for application-specific preferences beyond these defaults.

Remark. $\mathcal{M}_{\mathsf{corr}}$ is trained once for each dataset and reused for subsequent diversified top-k rule selection and discovery on the same dataset without retraining. We do not reuse a single $\mathcal{M}_{\mathsf{corr}}$ across datasets, even if they share similar schemas, because different schemas induce different predicate spaces and candidate rules for training $\mathcal{M}_{\mathsf{corr}}$, and distribution shifts can further change the rule space and training data.

B. Embedding-based objective function

Assume that we have a set Σ of discovered rules over schema $\mathcal{R}$, a set $\mathcal{A}$ of user-interested attributes in $\mathcal{R}$, and a set C of user-accumulated rules. Based on the learned subjective embeddings of predicates/rules via proposed correlation model $\mathcal{M}_{\mathsf{corr}}$, we define two measures: *relevance* and *diversity*. The relevance is to evaluate (a) whether a rule in Σ has a high correlation with the attributes in $\mathcal{A}$, while at the same time (b) is unexpected to the rules in C; the diversity is to evaluate (c) whether the rules in Σ are different from each other. Based on these two measures, we propose an objective function that considers both the relevance and diversity of rules.

Relevance measure. We first define two sub-measures, fitness and unexpectedness, to evaluate the (a) and (b) above, respectively, based on which we present the relevance measure.

Fitness. In addition to the preference templates in Section III-A2, we additionally ask users to select a set $\mathcal{A}$ of interested attributes from the schema $\mathcal{R}$, which is easy to do.

Definition 9. *Given a rule φ in Σ and a set $\mathcal{A}$ of user-selected attributes, we define the* fitness *of φ* w.r.t. *$\mathcal{A}$ as:*

$$\delta_{\mathsf{fit}}(\varphi, \mathcal{A}) = \min_{p_i \in \varphi} \max_{p_j \in \mathcal{P}^{\mathcal{A}}_{\mathsf{user}}} \mathsf{corr}(p_i, p_j), \tag{2}$$

where $\mathsf{corr}(p_i, p_j) = 1/(1 + ||E_{p_i}, E_{p_j}||_2)$ is the correlation between learned embeddings of predicates p_i and p_j, output by $\mathcal{M}_{\mathsf{corr}}$; $\mathcal{P}^{\mathcal{A}}_{\mathsf{user}}$ contains predicates involving the attributes in $\mathcal{A}$.

It utilizes the minimum of the maximal correlation between any pair (p_i, p_j) of predicates from φ and $\mathcal{P}^{\mathcal{A}}_{\mathsf{user}}$; intuitively, this indicates how rule φ meets the user's need w.r.t attributes $\mathcal{A}$.

Unexpectedness. It is common for users to have accumulated effective rules in practice. In such cases, they may have more expectations of reliable yet unexpected rules than common ones. To echo this, following [22], we define the *unexpectedness* to quantify the interestingness of newly discovered rules.

Definition 10. *Given a rule φ in Σ and a user-accumulated rule set C, we define the* unexpectedness *of φ* w.r.t. *C as:*

$$\delta_{\mathsf{unexp}}(\varphi, C) = \frac{1}{|C|} \left[\mathsf{corr}(C \cup \{\varphi\}) - \mathsf{corr}(C)\right], \tag{3}$$

where $\mathsf{corr}(C) = \sum_{\varphi_i, \varphi_j \in C, i<j} 1/(1 + ||E_{\varphi_i}, E_{\varphi_j}||_2)$ is the sum of correlation scores of all rule pairs in rule set C; similar for $\mathsf{corr}(C \cup \{\varphi\})$. When $C = \emptyset$, $\delta_{\mathsf{unexp}}(\varphi, C) = 0$.

Intuitively, *unexpectedness* reflects how "surprising" the rule φ is to the users, by checking to what extent φ contradicts the experience of users, *i.e.,* how does the addition of φ in C broaden the usage of C.

Relevance. Putting these together, we define the *relevance* of a rule φ as

$$\delta_{\mathsf{rel}}(\varphi, \mathcal{A}, C) = \delta_{\mathsf{fit}}(\varphi, \mathcal{A}) + \delta_{\mathsf{unexp}}(\varphi, C). \tag{4}$$

This said, rules with high relevance are ones that closely align with what the user cares about (attributes in $\mathcal{A}$), and at the same time offer novel, complementary knowledge beyond what the user already knows (rules in C). When the context is clear, we use $\delta_{\mathsf{rel}}(\varphi)$ to denote $\delta_{\mathsf{rel}}(\varphi, \mathcal{A}, C)$ for simplicity.

User effort. Experienced users can explicitly select an interested attribute set $\mathcal{A}$ from the schema $\mathcal{R}$ and maintain an accumulated rule set C (previously accepted rules) for fine-grained ranking and diversification. For non-experts, specifying a small set $\mathcal{A}$ of attributes they care about (*e.g.,* outcome) is usually straightforward. To further reduce the burden of specifying C, our system automatically suggests a small set of simple FDs with $|X| = 1$ (*e.g.,* ZipCode $\rightarrow$ City) that have high support and confidence; users only need to select the obviously valid ones to populate C as their accumulated experience. Novices may also leave C empty, in which case rule discovery proceeds based on $\mathcal{A}$ only. Together with the default preference templates in Section III-A, this enables a practical cold-start setting without user-authored templates or prior rules.

Diversity measure. We next define a diversity measure to evaluate the (c) mentioned above. Given a set Σ of rules, we define the *diversity* of Σ, denoted by $\delta_{\mathsf{div}}(\Sigma)$, as:

$$\delta_{\mathsf{div}}(\Sigma) = \sum_{\varphi_i, \varphi_j \in \Sigma, i<j} \delta_{\mathsf{dist}}(\varphi_i, \varphi_j), \tag{5}$$

where $\delta_{\mathsf{dist}}(\varphi_i, \varphi_j)$ is the *Euclidean distance* between the subjective embedding of rule φ_i and that of φ_j, output by $\mathcal{M}_{\mathsf{corr}}$. It reflects the diversity between rule pairs in Σ at a set level.

Objective function. We propose a max-sum objective function to combine $\delta_{\mathsf{rel}}(\cdot)$ and $\delta_{\mathsf{div}}(\cdot)$, whose goal is to evaluate the relevance and diversity of rules in a rule set. Following [23]–[25], given a set Σ of rules, $F(\Sigma)$ can be defined as:

$$F(\Sigma) = (1 - \lambda) \cdot (|\Sigma| - 1) \cdot \sum_{\varphi \in \Sigma} \delta_{\mathsf{rel}}(\varphi) + 2\lambda \cdot \delta_{\mathsf{div}}(\Sigma), \tag{6}$$

where the $\delta_{\mathsf{rel}}(\cdot)$ and $\delta_{\mathsf{div}}(\cdot)$ are defined above, and $\lambda \in [0, 1]$ is a parameter set by users to balance relevance and diversity. Smaller λ (*e.g.,* 0.3) emphasizes relevance, while larger λ (*e.g.,* 0.7) emphasizes diversity. When users have no strong preference, we use $\lambda = 0.5$ as a balanced default in our experiments.

An equivalent variant. The above $F(\Sigma)$ can be rewritten as

$$F(\Sigma) = \sum_{\varphi_i, \varphi_j \in \Sigma, i<j} \delta^{\mathsf{w}}_{\mathsf{dist}}(\varphi_i, \varphi_j), \tag{7}$$

where $\delta^{\mathsf{w}}_{\mathsf{dist}}(\varphi_i, \varphi_j) = (1 - \lambda) \cdot (\delta_{\mathsf{rel}}(\varphi_i) + \delta_{\mathsf{rel}}(\varphi_j)) + 2\lambda \cdot \delta_{\mathsf{dist}}(\varphi_i, \varphi_j)$ is a *weighted distance* between rules φ_i and φ_j [23], [24]. This equivalent transition holds, since $\sum_{\varphi_i, \varphi_j \in \Sigma, i<j} \delta^{\mathsf{w}}_{\mathsf{dist}}(\varphi_i, \varphi_j) = \sum_{\varphi_i, \varphi_j \in \Sigma, i<j} (1 - \lambda) \cdot (\delta_{\mathsf{rel}}(\varphi_i) + \delta_{\mathsf{rel}}(\varphi_j)) + \sum_{\varphi_i, \varphi_j \in \Sigma, i<j} 2\lambda \cdot \delta_{\mathsf{dist}}(\varphi_i, \varphi_j) = (1 - \lambda) \cdot (|\Sigma| - 1) \cdot \sum_{\varphi \in \Sigma} \delta_{\mathsf{rel}}(\varphi) + 2\lambda \cdot \delta_{\mathsf{div}}(\Sigma)$. We adopt this variant

in our solution for two tasks: (i) selecting diversified top-k rules from a given rule set (Section IV), and (ii) discovering diversified top-k rules directly from raw data (Section V).

Theorem 1. *The weighted distance $\delta^{w}_{\mathsf{dist}}(\cdot,\cdot)$ is non-negative and satisfies the triangle inequality.*

Proof. The $\delta^{\mathsf{w}}_{\mathsf{dist}}(\cdot,\cdot)$ is non-negative since $\lambda \in [0,1]$, $\delta_{\mathsf{rel}}(\cdot) = \delta_{\mathsf{fit}}(\cdot) + \delta_{\mathsf{unexp}}(\cdot) \geq 0$, and $\delta_{\mathsf{dist}}(\cdot,\cdot) \geq 0$ is the Euclidean distance between rule pairs. For any rules φ_i, φ_j and φ_k, $\delta^{\mathsf{w}}_{\mathsf{dist}}(\varphi_i,\varphi_k) = (1-\lambda)\big(\delta_{\mathsf{rel}}(\varphi_i)+\delta_{\mathsf{rel}}(\varphi_k)\big)+2\lambda\,\delta_{\mathsf{dist}}(\varphi_i,\varphi_k) \leq (1-\lambda)\big(\delta_{\mathsf{rel}}(\varphi_i)+\delta_{\mathsf{rel}}(\varphi_k)\big)+2\lambda\big(\delta_{\mathsf{dist}}(\varphi_i,\varphi_j)+\delta_{\mathsf{dist}}(\varphi_j,\varphi_k)\big)$, using the triangle inequality of $\delta_{\mathsf{dist}}(\cdot,\cdot)$. Since $(1-\lambda)\delta_{\mathsf{rel}}(\varphi_j) \geq 0$, adding $2(1-\lambda)\delta_{\mathsf{rel}}(\varphi_j)$ to the right-hand side yields $\delta^{\mathsf{w}}_{\mathsf{dist}}(\varphi_i,\varphi_k) \leq \delta^{\mathsf{w}}_{\mathsf{dist}}(\varphi_i,\varphi_j) + \delta^{\mathsf{w}}_{\mathsf{dist}}(\varphi_j,\varphi_k)$. Thus, $\delta^{\mathsf{w}}_{\mathsf{dist}}(\cdot,\cdot)$ satisfies the triangle inequality. □

IV. Selecting diversified top-k rules

This section formulates the problem (TopkDivSP) of selecting diversified top-k rules from a given set of discovered rules. We show that it is intractable and propose an effective method.

Problem definition (TopkDivSP). Based on the above objective function, we next state the problem TopkDivSP as follows:

- *Input*: A set Σ_{cand} of discovered rules over schema $\mathcal{R}$, a set C of user-accumulated rules, a set $\mathcal{A} \subseteq \mathcal{R}$ of user-specified attributes, an integer k, and a parameter λ.
- *Output*: A set Σ of diversified top-k rules from Σ_{cand}, such that (a) $|\Sigma| = k$; and (b) $F(\Sigma)$ is maximized, *i.e.,* $F(\Sigma) = \max_{\Sigma' \subseteq \Sigma_{\mathsf{cand}}} F(\Sigma')$ for any k-element set $\Sigma' \subseteq \Sigma_{\mathsf{cand}}$, where $F(\cdot)$ is defined in Section III-B.

In this paper, we assume that $|\Sigma_{\mathsf{cand}}| > k$.

Theorem 2. *The* TopkDivSP *is NP-hard.*

Proof. We prove its NP-hardness by a reduction from the *maximum dispersion problem* (MAXDP) [26], which is proved to be NP-hard. The MAXDP identifies a complete subgraph G_s of k nodes in a given graph G with n nodes $\{v_i | i \in [1,n]\}$, such that the sum of edge weights in G_s is maximized, where the edge weights satisfy the triangle inequality. We can construct a TopkDivSP instance by treating (1) each rule φ_i in Σ as a node $v_i \in G$, and (2) each weighted distance $\delta^{\mathsf{w}}_{\mathsf{dist}}(\varphi_i,\varphi_j)$ of a rule pair (φ_i,φ_j) (see Section III-B) as the weight of an edge in G, where the $\delta^{\mathsf{w}}_{\mathsf{dist}}(\varphi_i,\varphi_j) \geq 0$ satisfies the triangle inequality; this reduction takes polynomial time. Thus, the TopkDivSP is NP-hard. □

Basic idea. Given a set Σ_{cand} of rules and the initially empty Σ, inspired by the d-Greedy Augment [27], we first randomly select a rule from Σ_{cand} and add it to Σ; we then adopt a greedy strategy to select a rule φ' from $\Sigma_{\mathsf{cand}} \setminus \Sigma$ whose embedding is farthest away from the embeddings of other rules in Σ; we add the φ' into Σ. Repeat this process until k rules are selected (*i.e.,* $|\Sigma| = k$).

Algorithm. We propose an algorithm TopkDivSelector for TopkDivSP, shown in Algorithm 1. In line 1, it first trains a correlation model $\mathcal{M}_{\mathsf{corr}}$ and generates the subjective embeddings of predicates/rules (Section III-A), based on which we can obtain the objective function $F(\cdot)$, considering relevance metric $\delta_{\mathsf{rel}}(\cdot)$ and diversity metric $\delta_{\mathsf{div}}(\cdot)$ (Section III-B). In line 2, it randomly selects a rule φ_{rand} from Σ_{cand} and add it to Σ; it also initializes the φ_l to record the latest rule added to Σ, and the ranking score $RS(\varphi)$ for each $\varphi \in \Sigma_{\mathsf{cand}}$. In lines 3-9, it updates the Σ with more $(k-1)$ rules. In particular, for each $\varphi \in \Sigma_{\mathsf{cand}} \setminus \Sigma$, we update its ranking score based on the weighted distance $\delta^{\mathsf{w}}_{\mathsf{dist}}(\cdot,\cdot)$ between the rule pair (φ,φ_l) (lines 4-6), where the $\delta^{\mathsf{w}}_{\mathsf{dist}}(\cdot,\cdot)$ is introduced in Section III-B. Then, we select the rule φ' from $\Sigma_{\mathsf{cand}} \setminus \Sigma$ with the highest ranking score (line 7); we update the Σ and φ_l (line 8). Finally, the Σ is returned as the set of diversified top-k rules.

Algorithm 1 Algorithm TopkDivSelector

Input: A set Σ_{cand} of candidate REEs, a set C of user-accumulated rules, a set $\mathcal{A}$ of user-specified attributes, an integer k, and a parameter λ.
Output: A set $\Sigma \subset \Sigma_{\mathsf{cand}}$ of REEs s.t. $|\Sigma| = k$ and $F(\Sigma)$ is maximized.

1: Train a correlation model $\mathcal{M}_{\mathsf{corr}}$ and obtain the objective function $F(\cdot)$;
2: $\Sigma := \{\varphi_{\mathsf{rand}}\}$; $\varphi_l := \varphi_{\mathsf{rand}}$; $RS(\varphi) := 0$ for each $\varphi \in \Sigma_{\mathsf{cand}}$;
3: **while** $|\Sigma| < k$ **do**
4: **for** each $\varphi \in \Sigma_{\mathsf{cand}} \setminus \Sigma$ **do**
5: $RS(\varphi) := RS(\varphi) + \delta^{\mathsf{w}}_{\mathsf{dist}}(\varphi, \varphi_l)$;
6: **end for**
7: $\varphi' :=$ the rule φ in $\Sigma_{\mathsf{cand}} \setminus \Sigma$ with the highest $RS(\varphi)$;
8: $\Sigma := \Sigma \cup \{\varphi'\}$; $\varphi_l := \varphi'$;
9: **end while**
10: **return** Σ;

Complexity. TopkDivSelector takes $O(c_{\mathcal{M}_{\mathsf{corr}}} + kn(|C|+|\mathcal{A}|))$ to identify the Σ from Σ_{cand}, where $n = |\Sigma_{\mathsf{cand}}|$, $c_{\mathcal{M}_{\mathsf{corr}}}$ is the (one-time) cost of training $\mathcal{M}_{\mathsf{corr}}$ via user-defined criteria, and $|C| + |\mathcal{A}|$ is the cost of computing the ranking score of a rule pair *w.r.t.* the C and $\mathcal{A}$ in $\delta^{\mathsf{w}}_{\mathsf{dist}}(\cdot,\cdot)$. Note that, $\mathcal{M}_{\mathsf{corr}}$ is implemented as a lightweight Siamese network with only a few linear layers, trained on a small number of automatically labeled rule pairs. The model is trained offline once for each dataset and then can be reused to select the diversified top-k rules from any Σ_{cand} without further retraining. Thus, the online complexity of TopkDivSelector reduces to $O(kn(|C|+|\mathcal{A}|)$ when a $\mathcal{M}_{\mathsf{corr}}$ is already available.

Theorem 3. *Algorithm* TopkDivSelector *approximates the* TopkDivSP *problem with approximation ratio 2.*

Proof. We have proven that the MAXDP can be reduced to the TopkDivSP, where the MAXDP is also known as the *remote clique* problem [27], [28]. Thus, the approximate algorithm for TopkDivSP implies an approximation algorithm for the remote clique problem. In particular, TopkDivSelector simulates an instance of the d-Greedy Augment in [27] when $d = 1$; it returns a set Σ of k rules such that $F(\Sigma) \geq (k+d-2)/(2k-2) \cdot F(\Sigma^*) = \frac{1}{2} \cdot F(\Sigma^*)$, where Σ^* is the optimal diversified top-k rules, *i.e.,* TopkDivSelector approximates TopkDivSP with approximation ratio 2. Moreover, the approximation ratio

$(2k-2)/(k+d-2)$ of d-Greedy Augment for the remote clique problem has been proved to be *worst-case tight* [27]. Since TopkDivSelector is the $d=1$ instance of d-Greedy Augment on our rule graph, the above factor-2 bound is also a worst-case tight approximation ratio for TopkDivSP. □

V. Discovering diversified top-k rules

This section formulates another problem (TopkDivFP) of discovering diversified top-k rules in an instance $\mathcal{D}$ from scratch. We show it is also intractable and propose an effective solution with pruning strategies and parallel optimization.

Problem statement (TopkDivFP). Based on the proposed objective function, we state the problem TopkDivFP as follows:

- An instance $\mathcal{D}$ over schema $\mathcal{R}$, a support threshold σ, a confidence threshold γ, the maximum level ℓ_{max} of rule discovery, a set C of user-accumulated rules, a set $\mathcal{A}$ of user-specified attributes, an integer k, and a parameter λ.
- A set Σ of diversified top-k rules from $\mathcal{D}$, such that (a) $|\Sigma| = k$; (b) $F(\Sigma)$ is maximized, *i.e.,* $F(\Sigma) = \max_{\Sigma' \subseteq \Sigma_{\mathsf{all}}} F(\Sigma')$ among all minimal k-element rule sets in Σ_{all}, where $F(\cdot)$ is defined in Section III-B; and (c) for each $\varphi \in \Sigma$, φ is minimal, *i.e.,* σ-frequent and γ-confident (discussed in Section II).

Theorem 4. *The* TopkDivFP *is NP-hard.*

Proof. To solve the TopkDivFP, one needs to first (1) identify a set Σ_{all} of all minimal rules in $\mathcal{D}$, and then (2) identify a set Σ of rules from Σ_{all}, such that $|\Sigma| = k$ and $F(\Sigma)$ is maximized. Note that, the step (2) above is actually the TopkDivSP, which is proved to be NP-hard in Section IV. That is, the TopkDivFP is NP-hard. □

A. Algorithm

Basic idea. We propose a solution to enumerate candidate rules in a traditional *levelwise* manner. In particular, given a rule set Σ with k rules, we introduce a *border rule* $\varphi_{\mathsf{min}}(\Sigma)$ (see below) in Σ that refers to rule in Σ to be replaced when a better one φ' emerges. Thus, given the Σ with $|\Sigma| = k$, in the process of rule discovery, when a newly discovered φ' emerges, it updates the Σ by deleting the $\varphi_{\mathsf{min}}(\Sigma)$ and inserting the φ', through a *swapping* operation.

Definition 11. *Given a set Σ of k rules, we define the border rule of Σ as:*

$$\varphi_{\mathsf{min}}(\Sigma) = \arg\min_{\varphi \in \Sigma} \{ \sum_{\varphi' \in \Sigma, \varphi' \neq \varphi} \delta^{\mathsf{w}}_{\mathsf{dist}}(\varphi, \varphi') \}, \qquad (8)$$

where $\delta^{\mathsf{w}}_{\mathsf{dist}}(\cdot,\cdot)$ is a weighted distance metric, defined in Section III-B. Intuitively, $\varphi_{\mathsf{min}}(\Sigma)$ is the rule that, when removed from Σ, results in the least drop of the $F(\Sigma)$ score.

Algorithm. We propose a *swapping-based levelwise search* algorithm TopkDivFinder for TopkDivFP, shown in Algorithm 2. Denote by $\mathcal{P}_{\mathsf{all}}$ the set of all predicates in rule discovery; for a rule $\varphi : X \rightarrow p_o$, denote by $\mathcal{P}_{\mathsf{sel}}$ (resp. $\mathcal{P}_{\mathsf{re}}$) the set of selected (resp. candidate) predicates in $\mathcal{P}_{\mathsf{all}}$ to constitute X; denote by $\mathcal{Q}$ (resp. $\mathcal{Q}_{\mathsf{next}}$) a queue that stores the candidate REEs

Algorithm 2 TopkDivFinder

Input: An instance $\mathcal{D}$ over schema $\mathcal{R}$, a support threshold σ, a confidence threshold γ, the maximal level ℓ_{max} of rule discovery, a set C of user-accumulated rules, a set $\mathcal{A} \in \mathcal{R}$ of user-interested attributes, an integer k and a parameter λ.
Output: A set Σ of minimal REEs s.t. $|\Sigma| = k$ and $F(\Sigma)$ is maximized.

1: Train a correlation model $\mathcal{M}_{\mathsf{corr}}$ and obtain the objective function $F(\cdot)$;
2: Build auxiliary structures;
3: $\mathcal{P}_{\mathsf{sel}} := \emptyset$; $\mathcal{P}_{\mathsf{re}} := \mathcal{P}_{\mathsf{all}}$; $\ell := 0$; $\Sigma := \emptyset$; $\mathcal{Q} := \emptyset$;
4: $\mathcal{Q}.\mathsf{add}(\langle \mathcal{P}_{\mathsf{sel}}, \mathcal{P}_{\mathsf{re}}, p_o \rangle)$ for each $p_o \in \mathcal{P}_{\mathsf{all}}$;
5: **while** $\ell \leq \ell_{\mathsf{max}}$ **do**
6: $\quad \langle \mathcal{Q}_{\mathsf{next}}, \Sigma \rangle := \mathsf{Expand}(\mathcal{Q}, k, \sigma, \gamma, \lambda, \mathcal{A}, C, \Sigma)$;
7: $\quad \ell := \ell + 1$; $\mathcal{Q} := \mathcal{Q}_{\mathsf{next}}$;
8: **end while**
9: **return** Σ;

Procedure Expand
Input: $\mathcal{Q}$, k, σ, γ, λ, $\mathcal{A}$, C, and current set Σ of mined REEs.
Output: (1) The $\mathcal{Q}_{\mathsf{next}}$ of candidate REEs for the next level; and (2) the updated Σ.

1: $\mathcal{Q}_{\mathsf{next}} := \emptyset$;
2: **for all** $\langle \mathcal{P}_{\mathsf{sel}}, \mathcal{P}_{\mathsf{re}}, p_o \rangle \in \mathcal{Q}$ **do**
3: $\quad \varphi := \mathcal{P}_{\mathsf{sel}} \rightarrow p_o$;
4: $\quad$ **if** $\exists \varphi' \in \Sigma$ s.t. $\varphi' \preceq \varphi$ or $\mathsf{supp}(\varphi) < \sigma$ **then**
5: $\quad\quad$ **continue**; // Pruning for early termination
6: $\quad$ **end if**
7: $\quad$ **if** φ is minimal (σ-frequent and γ-confident) **then**
8: $\quad\quad$ **if** $|\Sigma| < k$ **then**
9: $\quad\quad\quad \Sigma := \Sigma \cup \{\varphi\}$;
10: $\quad\quad$ **else if** $F(\Sigma \setminus \{\varphi_{\mathsf{min}}(\Sigma)\} \cup \{\varphi\}) > F(\Sigma)$ **then**
11: $\quad\quad\quad \Sigma := \Sigma \setminus \{\varphi_{\mathsf{min}}(\Sigma)\} \cup \{\varphi\}$; // Swapping
12: $\quad\quad$ **end if**
13: $\quad\quad$ **continue**;
14: $\quad$ **end if**
15: $\quad$ **for all** $p \in \mathcal{P}_{\mathsf{re}}$ **do**
16: $\quad\quad \mathcal{Q}_{\mathsf{next}}.\mathsf{add}(\langle \mathcal{P}_{\mathsf{sel}} \cup \{p\}, \mathcal{P}_{\mathsf{re}} \setminus \{p\}, p_o \rangle)$;
17: $\quad$ **end for**
18: **end for**
19: **return** $\langle \mathcal{Q}_{\mathsf{next}}, \Sigma \rangle$;

to be examined in the current (resp. next) level; and denote by ℓ the current level of rule discovery. Same to the algorithm TopkDivSelector, TopkDivFinder first trains the model $\mathcal{M}_{\mathsf{corr}}$ (if it is not available), generates the subjective embeddings of predicates/rules, and obtains the objective function $F(\cdot)$ in line 1. It builds auxiliary structures, *e.g.,* position list indexes (PLI) [29] for accelerating rule discovery (line 2). It initializes the $\mathcal{P}_{\mathsf{sel}}$, $\mathcal{P}_{\mathsf{re}}$, ℓ, Σ and $\mathcal{Q}$ in line 3. It updates the $\mathcal{Q}$ with $|\mathcal{P}_{\mathsf{all}}|$ candidate rules in the form of triples $\langle \mathcal{P}_{\mathsf{sel}}, \mathcal{P}_{\mathsf{re}}, p_o \rangle$ in line 4, where each triple is used to discover a candidate rule $\varphi : \mathcal{P}_{\mathsf{sel}} \rightarrow p_o$ with the *consequence* p_o. In lines 5-8, it traverses the search space level by level to expand the *precondition* $\mathcal{P}_{\mathsf{sel}}$ (*i.e.,* X) of φ via a procedure Expand (see below), where the Expand (a) updates the Σ of diversified top-k rules and (b)

updates the triples to be examined in the next level. When it reaches the maximal level of discovery, TopkDivFinder returns the Σ as the suggested diversified top-k rules from $\mathcal{D}$.

Procedure Expand. As shown in Algorithm 2, Expand initializes $\mathcal{Q}_{\mathsf{next}}$ in line 1. In lines 2-18, it pops triples from $\mathcal{Q}$, updates Σ through the *swapping* operation, expands triples into multiple new ones by adding different predicates from $\mathcal{P}_{\mathsf{re}}$ to $\mathcal{P}_{\mathsf{sel}}$, and adds these new triples in $\mathcal{Q}_{\mathsf{next}}$; in particular, Expand considers the minimality, relevance and diversity of rules and at level i, $|\mathcal{P}_{\mathsf{sel}}| = i$. To be specific, for each candidate rule $\varphi : \mathcal{P}_{\mathsf{sel}} \rightarrow p_o$ at level i, it first checks whether φ needs to be further explored: (1) it prunes φ if φ is non-minimal w.r.t φ' or with small support in lines 4-6, where $\varphi' \preceq \varphi$ indicates that there is a less restricted rule $\varphi' : X' \rightarrow p_o$ with $X' \subset X$; (2) when φ is minimal (*i.e.,* σ-frequent and γ-confident as discussed in Section II), it updates Σ with φ by (a) directly adding φ to Σ if $|\Sigma| < k$ (lines 8-9) or (b) replacing the border rule $\varphi_{\mathsf{min}}(\Sigma)$ in Σ with φ if this swapping can make the objective function $F(\cdot)$ larger (lines 10-12); and (3) it updates the $\mathcal{Q}_{\mathsf{next}}$ by further expanding the $\mathcal{P}_{\mathsf{sel}}$ of φ from level i to $i+1$ (lines 15-17) when φ is σ-frequent but with low confidence and thus needs to be further expanded. Finally, it returns the updated $\mathcal{Q}_{\mathsf{next}}$ and Σ to TopkDivFinder (line 19).

Complexity. The algorithm TopkDivFinder takes at most $\mathcal{O}(c_{\mathcal{M}_{\mathsf{corr}}} + \sum_{\varphi \in C(\mathcal{P}_{\mathsf{all}}) \times \mathcal{P}_{\mathsf{all}}} \mathcal{T}_{\mathsf{rel}}(\varphi) + k\mathcal{T}_{\mathsf{dist}}(\varphi, \varphi'))$ time, where $c_{\mathcal{M}_{\mathsf{corr}}}$ is the cost of training the correlation model $\mathcal{M}_{\mathsf{corr}}$ if it is not available, $C(\mathcal{P}_{\mathsf{all}})$ is the power set of $\mathcal{P}_{\mathsf{all}}$, and $\mathcal{T}_{\mathsf{rel}}(\cdot)$ (resp. $\mathcal{T}_{\mathsf{dist}}(\cdot,\cdot)$) is the time for computing the relevance of each rule (resp. the distance between a pair of rules). To be specific, $\mathcal{T}_{\mathsf{rel}}(\varphi) = \mathcal{O}((|\varphi||\mathcal{A}| + |C|)d)$ and $\mathcal{T}_{\mathsf{dist}}(\varphi, \varphi') = \mathcal{O}(d)$, where d is the dimension of predicate embeddings. Thus, TopkDivFinder takes at most $\mathcal{O}(c_{\mathcal{M}_{\mathsf{corr}}} + \sum_{\varphi \in C(\mathcal{P}_{\mathsf{all}}) \times \mathcal{P}_{\mathsf{all}}} (|\varphi||\mathcal{A}| + |C| + k)d)$ time. As discussed in Section IV, $c_{\mathcal{M}_{\mathsf{corr}}}$ is a relatively small one-time offline training cost per dataset and is reused in discovery without retraining. Thus, our per-run complexity focuses on the second term, *e.g.,* rule discovery over the full dataset $\mathcal{D}$, whose combinatorial search space dominates the end-to-end runtime in practice. Note that the above bound is worst-case; with the effective optimization strategies below, our algorithm is much faster in practice.

B. Optimization Strategies

We adopt three optimization strategies to further accelerate TopkDivFinder: we (a) train a validity model to avoid unnecessary computation of supports and confidences; (b) separate constant predicates from the expansion of $\mathcal{P}_{\mathsf{sel}}$ to reduce the enumeration cost; and (c) adopt a parallel strategy to speed up algorithm. Due to space limitations, we briefly outline their core idea and indicate corresponding modules in Algorithm 2.

Validity model. We train a 3-class model $\mathcal{M}_{\mathsf{valid}}$ that predicts which class a rule φ belongs to: (a) valid, (b) invalid due to small support or (c) invalid due to small confidence; it takes φ as input and outputs the predicted class along with the confidence. If $\mathcal{M}_{\mathsf{valid}}$ classifies φ as the case (a) above with high confidence, we need to check whether updating Σ with φ (lines 7-14 of Expand in Algorithm 2); if the classification is case (b) with high confidence, we can prune φ (*i.e.,* lines 4-6 of Expand) without calculating the exact support and confidence of φ (*i.e.,* the most time-consuming unit in rule discovery); and if the classification is case (c) with high confidence, we need to further expand φ (*i.e.,* lines 15-17 of Expand). Model $\mathcal{M}_{\mathsf{valid}}$ enables us to compute the exact $\mathsf{supp}(\varphi)$ and $\mathsf{conf}(\varphi)$ of a rule φ only when the confidence of $\mathcal{M}_{\mathsf{valid}}$'s predication of φ is low.

Note that $\mathcal{M}_{\mathsf{valid}}$ is trained per dataset, as is $\mathcal{M}_{\mathsf{corr}}$. For the same reasons in Section III-A, we do not reuse $\mathcal{M}_{\mathsf{valid}}$ across datasets even with similar schemas. When discovering Σ_s from $\mathcal{D}_{\mathsf{s}}$ to construct training data for $\mathcal{M}_{\mathsf{corr}}$, we simultaneously record 3-class labels for rules based on support/confidence thresholds. Thus, no extra mining is needed; a single discovery on $\mathcal{D}_{\mathsf{s}}$ per dataset suffices to train both $\mathcal{M}_{\mathsf{corr}}$ and $\mathcal{M}_{\mathsf{valid}}$.

Preprocessing constant predicates offline. We first give the motivation for this optimization strategy and then present its basic idea. This optimization is motivated by our preliminary experiments on three common datasets (Adult [10], [13], [14], [30], [31], Airport [10], [13], [14], [31] and Hospital [10]–[14]): given an instance $\mathcal{D}$, we discovered the rules in $\mathcal{D}$ by (a) only considering non-constant predicates *vs.* that by (b) considering both non-constant and constant ones (with frequency $\geq 5\%|\mathcal{D}|$). From the result (not shown), we found that case (a) was $100\times$ faster than case (b), on average.

Based on this observation, we propose to offline compute the frequency for all possible combinations of constant predicates via Apriori [32], and then expand $\mathcal{P}_{\mathsf{sel}}$ only with the non-constant predicates in Expand in Algorithm 2. To be specific, we split the predicates in $\mathcal{P}_{\mathsf{all}}$ (line 3 in TopkDivFinder) into constant and non-constant ones, denoted by $\mathcal{P}_c$ and $\mathcal{P}_v$, respectively. In line 3 in Expand, we expand the φ with multiple new rules φ_{new}, where each φ_{new} is in the form of $\mathcal{P}_{\mathsf{sel}} \cup \{\mathcal{P}'\} \rightarrow p_o$, and $\mathcal{P}'$ is a frequent constant predicate combination from $\mathcal{P}_c$. Then, for each φ_{new} (instead of φ), we execute the operation in lines 4-14. Another modification lies in lines 15-17 in Expand, we only consider non-constant (instead of all) predicates $p \in \mathcal{P}_{\mathsf{re}}$ for updating $\mathcal{Q}_{\mathsf{next}}$. With the modification above, we can greatly reduce the enumeration cost of expanding $\mathcal{P}_{\mathsf{sel}}$ by only considering non-constant predicates in the discovery process. In our implementation, frequent constant-predicate combinations are computed once per dataset offline and reused throughout rule discovery. This preprocessing is lightweight and incurs only a small overhead compared with the end-to-end runtime of PTopkDivFinder (Table V).

Parallelization. To further speed up the discovery process, we extend TopkDivFinder to PTopkDivFinder (not shown) through parallelization; in particular, it extends the Expand procedure of TopkDivFinder to a parallel version. The basic idea is as follows. It distributes the queue $\mathcal{Q}$ to multiple workers, where each worker concurrently handles a subset $\mathcal{Q}_{\mathsf{sub}}$ of candidate rules from $\mathcal{Q}$. Each worker needs to (a) verify the validity of candidate rules in $\mathcal{Q}_{\mathsf{sub}}$, (b) prune the invalid ones, and (c) process the expansion of rules for the next level. Moreover, the valid rules along with the candidate ones to be examined in the next level are sent to a coordinator; the coordinator aggregates all the results, updates the diversified top-k rule set Σ, and redistributes the candidate rules to be checked in

TABLE III
DATASET STATISTICS

Datasets	#Tuples	#Attributes	#Relations
Adult [10], [13]–[15], [30], [31]	32,561	15	1
Airport [10], [13]–[15], [31]	55,113	18	1
Hospital [10]–[15]	114,919	15	1
Inspection [10], [13]–[15], [33]	220,940	17	1
NCVoter [10], [13]–[15], [31]	1,681,617	12	1
DBLP [10], [14], [15], [34]	1,799,559	18	3

TABLE IV
ACCURACY OF $\mathcal{M}_{\mathsf{corr}}$ AND $\mathcal{M}_{\mathsf{valid}}$ ACROSS ALL DATASETS

Models	Adult	Airport	Hospital	Inspection	NCVoter	DBLP
$\mathcal{M}_{\mathsf{corr}}$	0.932	0.975	0.908	0.947	0.938	0.866
$\mathcal{M}_{\mathsf{valid}}$	0.952	0.913	0.858	0.937	0.85	0.912

the next level back to the workers for further processing. This process runs iteratively till (a) it reaches the maximal discovery level or (b) there are no more candidates to be evaluated.

VI. EXPERIMENTAL STUDY

Using real-life data, we experimentally evaluated the (1) effectiveness and (2) efficiency of the proposed approach of selecting and discovering diversified top-k rules for (a) different metrics and (b) adjustable parameters. The source code and tested datasets are available online [35].

A. Experimental settings

Datasets. We used six real-world datasets $\mathcal{D}$ (Table III). Adult, Airport, Hospital, Inspection and NCVoter are commonly used in existing studies, while DBLP is an academic dataset with multiple relations. We ran PTopkDivFinder on the full dataset $\mathcal{D}$ for rule discovery, and executed TopkDivSelector on the set Σ_{all} of all rules discovered by REEFinder [14]. For each dataset $\mathcal{D}$, we uniformly sampled 20K tuples as $\mathcal{D}_{\mathsf{s}}$ for training $\mathcal{M}_{\mathsf{corr}}$ (Section III-A) and $\mathcal{M}_{\mathsf{valid}}$ (Section V). Following prior work [15], we treat 20K as a medium-sized default. Table V shows that mining Σ_{s} from $\mathcal{D}_{\mathsf{s}}$ is already non-trivial; using a larger sample would substantially increase mining time and dominate preprocessing, as discovery cost grows exponentially with data size. Moreover, under uniform sampling, a moderate increase in $\mathcal{D}_{\mathsf{s}}$ mainly adds redundant evidence and is not expected to significantly change Σ_{s}. Thus, we fix $|\mathcal{D}_{\mathsf{s}}| = 20$K across all datasets. Additionally, we invited data-quality experts to specify the user-interested attributes $\mathcal{A}$, and provided a set of easy-to-understand FDs with $|X| = 1$ for them to select the obviously valid rules to constitute C (Section III-B).

ML models. (1) For the correlation model $\mathcal{M}_{\mathsf{corr}}$, we used 2 hidden layers with dimensions as 100, and set the size of its predicate embeddings to 768. We used Adam optimizer with a batch size of 128 and a learning rate of 0.001. To train $\mathcal{M}_{\mathsf{corr}}$, we limited the maximum number of rules Σ_{s} mined on $\mathcal{D}_{\mathsf{s}}$ to 1000, and randomly selected 3000 pairs of rules; we labeled these rule pairs by Snorkel [18] based on the five function templates in Section III-A; and we divided the labeled rule pairs into training and testing sets in a ratio of 8:2. We trained $\mathcal{M}_{\mathsf{corr}}$ with 400 epochs. (2) For the validity model $\mathcal{M}_{\mathsf{valid}}$, we employed XGBoost [36] and set (a) the number of boosting rounds to 100, (b) the maximum depth of each tree to 4, and (c) the learning rate to 0.01. We limited $|\Sigma_{\mathsf{s}}|$ mined on $\mathcal{D}_{\mathsf{s}}$ to be 3000, from which 1000 rules are for each type of classes (*i.e.,* valid, invalid due to small support and invalid due to small confidence). We encode the precondition and consequence of each rule as $|\mathcal{P}_{\mathsf{all}}|$-dimensional bit vectors [14], which are then concatenated to form the feature vector of the rule, fed into $\mathcal{M}_{\mathsf{valid}}$. (3) Moreover, we used two ML predicates of REEs in this paper: (a) default ditto [37] for ER; and (b) Bert [19] for textual attributes in $\mathcal{D}$.

TABLE V
OFFLINE PREPROCESSING TIME (SECONDS): TRAINING $\mathcal{M}_{\mathsf{corr}}$ AND $\mathcal{M}_{\mathsf{valid}}$, CONSTANT-PREDICATE PREPROCESSING ($\mathcal{P}_c$), AND MINING Σ_s FROM $\mathcal{D}_{\mathsf{s}}$

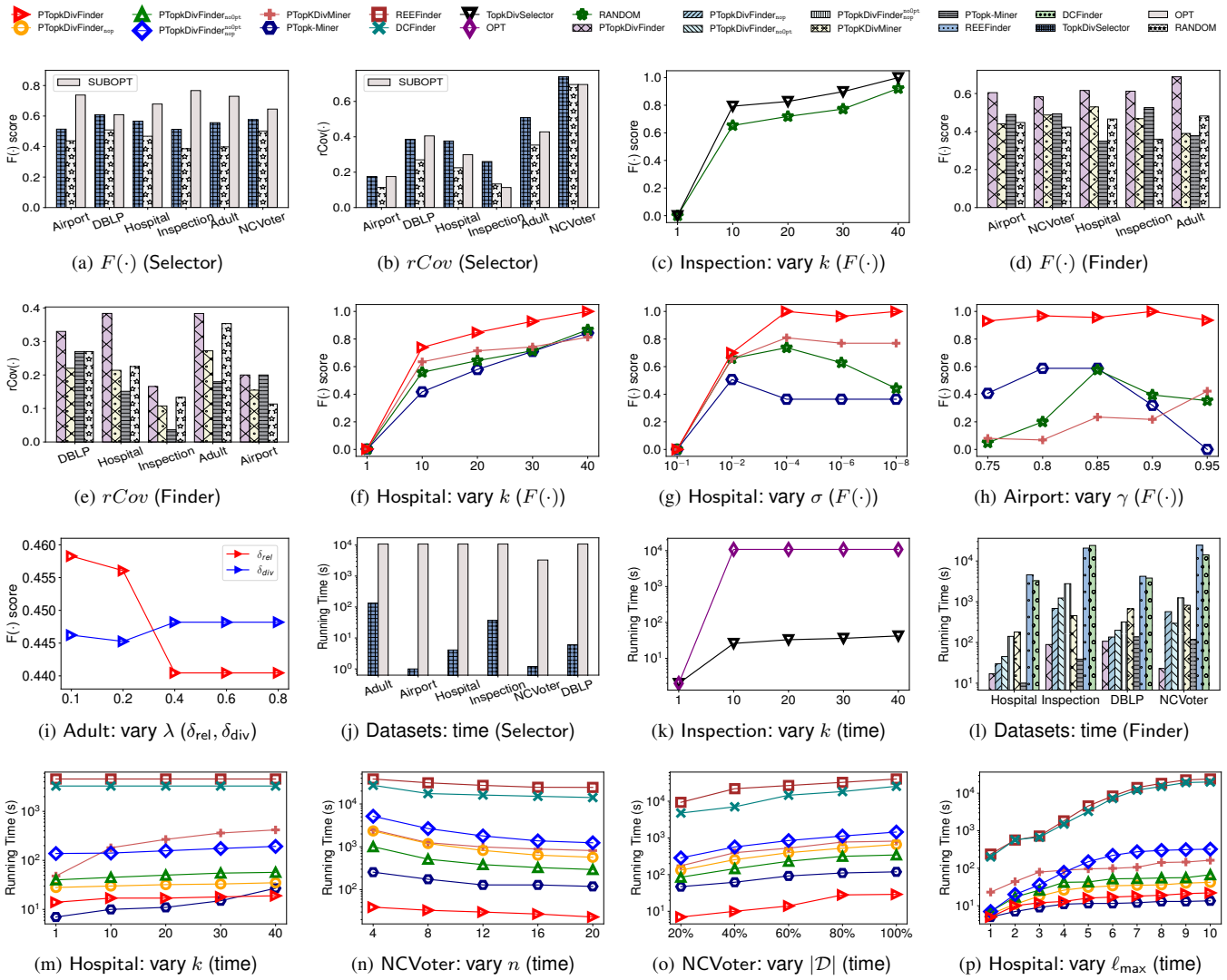
Datasets	Train $\mathcal{M}_{\mathsf{corr}}$	Train $\mathcal{M}_{\mathsf{valid}}$	Process $\mathcal{P}_c$	Discover Σ_s
Adult	71	2.78	0.47	442
Airport	83	1.43	2.60	38
Hospital	74	1.56	0.70	62
Inspection	86	2.63	2.16	400
NCVoter	90	3.76	0.62	1648
DBLP	87	1.85	2.99	58

Accuracy. Table IV reports the accuracy of $\mathcal{M}_{\mathsf{corr}}$ and $\mathcal{M}_{\mathsf{valid}}$ across all datasets; they achieve average accuracies of 0.928 and 0.904, respectively. This indicates that $\mathcal{M}_{\mathsf{corr}}$ provides reliable guidance for embedding learning, while $\mathcal{M}_{\mathsf{valid}}$ enables effective model-driven pruning in rule discovery.

Baselines. We implemented algorithms TopkDivSelector and PTopkDivFinder (*i.e.,*TopkDivFinder equipped with three optimizations) in Java. Since no existing methods directly solve TopkDivSP, we compared TopkDivSelector with (1) OPT, the exact solution that selects the best set of k rules following the criterion in Section III; and (2) RANDOM, an approach that randomly selects a set of k rules from a given rule set.

We compared PTopkDivFinder with the RANDOM; (3) PTopk-Miner [10], a top-k rule discovery method that trains a relevance model without considering the diversity among rules; (4) PTopKDivMiner [15], a diversified top-k rule discovery method equipped with an error detection-based relevance model and four diversity measures; (5) REEFinder [14], an exhaustive method that discovers the entire set Σ_{all} of ($> k$) possible REEs; and (6) DCFinder [12], the state-of-the-art method that mines all DCs. In particular, we parallelized DCFinder for a fair comparison. We also evaluated three variants of PTopkDivFinder: (7) $\mathsf{PTopkDivFinder}_{\mathsf{nop}}$, a variant that does not prune rules via $\mathcal{M}_{\mathsf{valid}}$; (8) $\mathsf{PTopkDivFinder}_{\mathsf{noOpt}}$, a variant that considers constant predicates when expanding $\mathcal{P}_{\mathsf{sel}}$ in rule discovery; and (9) $\mathsf{PTopkDivFinder}_{\mathsf{nop}}^{\mathsf{noOpt}}$, a variant that does not use $\mathcal{M}_{\mathsf{valid}}$ for pruning rules and meanwhile considers constant predicates in the expansion of $\mathcal{P}_{\mathsf{sel}}$. We compared PTopkDivFinder with (5)-(9) to verify the efficiency of the proposed optimization strategies.

Metrics. We evaluated the effectiveness of TopkDivSelector and PTopkDivFinder with two commonly used metrics: the proposed objective function score in Section III-B and the rule coverage [38]. We evaluated the efficiency of TopkDivSelector (resp. PTopkDivFinder) by its execution time on a single machine (resp. a cluster of machines).



(a) $F(\cdot)$ (Selector) (b) $rCov$ (Selector) (c) Inspection: vary k ($F(\cdot)$) (d) $F(\cdot)$ (Finder)

(e) $rCov$ (Finder) (f) Hospital: vary k ($F(\cdot)$) (g) Hospital: vary σ ($F(\cdot)$) (h) Airport: vary γ ($F(\cdot)$)

(i) Adult: vary λ ($\delta_{\text{rel}}, \delta_{\text{div}}$) (j) Datasets: time (Selector) (k) Inspection: vary k (time) (l) Datasets: time (Finder)

(m) Hospital: vary k (time) (n) NCVoter: vary n (time) (o) NCVoter: vary $|\mathcal{D}|$ (time) (p) Hospital: vary ℓ_{max} (time)

Fig. 2. Performance evaluation of TopkDivSelector and PTopkDivFinder: Effectiveness and efficiency analysis.

Consider Σ (resp. Σ_{all}) a set of k (resp. all) rules discovered from dataset $\mathcal{D}$. Following [38], we define the *rule coverage* of a rule set Σ as:

$$rCov(\Sigma) = \frac{\sum_{\varphi_j \in \Sigma_{\text{all}}} \mathbb{1}[\exists \varphi_i \in \Sigma \ s.t. \ sim(\varphi_i, \varphi_j) > \tau]}{|\Sigma_{\text{all}}|}, \quad (9)$$

where $sim(\cdot)$ is a similarity function and τ is a predefined threshold; we used Jaccard Similarity for $sim(\cdot)$ and set $\tau = 0.5$. A large $rCov(\Sigma)$ indicates that despite Σ only contains k rules, it covers a wide range of diversified rules in Σ_{all}.

Configuration. We conducted experiments on a cluster of 21 machines (1 coordinator and 20 workers), each powered by 64GB RAM and 16 processors with 2.20 GHz. We ran the experiments 3 times, and report the average. Unless stated explicitly, we set the number of machines $n = 20$, the support threshold $\sigma = 10^{-6} \cdot |\mathcal{D}|^2$, the confidence threshold $\gamma = 0.75$, the parameter $\lambda = 0.5$, the prediction confidence threshold 0.9 for the $\mathcal{M}_{\text{valid}}$, the maximum discovery level $\ell_{\text{max}} = 5$ and $k = 10$. Note we do not include the time of loading data and constructing auxiliary structure, *e.g.,* PLI, for all algorithms.

B. Experimental results

Exp-1: Effectiveness of TopkDivSelector**.** We first evaluated the effectiveness of TopkDivSelector and baselines under the default setting, then studied the impact of parameter k.

Accuracy vs. baselines. Since computing OPT is NP-hard and infeasible within the time budget, we followed [15] and compared TopkDivSelector against a sub-optimal comparator SUBOPT obtained by swapping-based local search. Specifically, we initialized 100 random k-subsets $\Sigma_s \subset \Sigma_{\text{all}}$ and, for each Σ_s, repeatedly replace one rule in Σ_s with another from Σ_{all} that yields the largest increase in $F(\Sigma_s)$, until no improving single swap existed. The best local optimum across the 100 runs was reported as the suboptimal result. As shown in Figs. 2(a) and 2(b), TopkDivSelector consistently outperforms Random *w.r.t.* both normalized $F(\cdot)$ and $rCov(\cdot)$, *e.g.,* it is on average 24.29% and 51.6% higher than Random, and up to 39.9% and 95.1%, respectively. Although SUBOPT yields a higher $F(\cdot)$ than TopkDivSelector (by 26.01% on average), TopkDivSelector achieves 29.25% higher $rCov(\cdot)$ on average, highlighting a relevance–diversity trade-off.

Varying k. We varied the number k of discovered rules from 1 to 40 in Fig. 2(c). From the results, we can see that when k increases, the $F(\cdot)$ increases, *e.g.,* the $F(\cdot)$ of TopkDivSelector increases by 0.11 on Inspection when k increases from 10 to 30. This is because a larger k enables the algorithm to discover and return a larger set of rules with more complex correlations. Nevertheless, TopkDivSelector performs the best, *e.g.,* its $F(\cdot)$ on Inspection is 0.793 when $k = 10$, 21.37% higher than that of Random, which again verifies the effectiveness of TopkDivSelector.

Exp-2: Effectiveness of PTopkDivFinder**.** It first evaluated the effectiveness of PTopkDivFinder and baselines, followed by the impact of different parameters on the performance.

Accuracy vs. baselines. As shown in Figs. 2(d) and 2(e), PTopkDivFinder performs the best *w.r.t.* both relative function score $F(\cdot)$ and rule coverage $rCov(\cdot)$, *e.g.,* its $F(\cdot)$ is 0.702 on average, 36.09%, 43.31% and 43.57% higher than PTopKDivMiner, PTopk-Miner and RANDOM, respectively. This is because, rather than considering the relevance of rules in Σ alone as PTopk-Miner, PTopkDivFinder considers both the relevance and diversity of rules in Σ. We also compared PTopkDivFinder with its variants, and obtained similar results. All above verifies the effectiveness of PTopkDivFinder.

Varying k. We varied the number k of discovered rules from 1 to 40 in Fig. 2(f). Similar as the trend of TopkDivSelector, when k increases, the $F(\cdot)$ increases, *e.g.,* the $F(\cdot)$ of PTopkDivFinder on Hospital is 0.738 with $k = 10$, as opposed to 0.847 with $k = 20$. Moreover, PTopkDivFinder beats PTopKDivMiner, PTopk-Miner and RANDOM by 0.12, 0.19 and 0.15 on average, up to 0.19, 0.32 and 0.21, respectively. The reasons behind are very similar to that of TopkDivSelector on Inspection, and thus we omit the discussion here.

Varying σ. Varying the support threshold σ from 10^{-1} to 10^{-8} in Fig. 2(g), we found that when σ decreases, the $F(\cdot)$ of PTopkDivFinder first increases and then converges, *e.g.,* its $F(\cdot)$ on Hospital stabilizes when $\sigma = 10^{-4}$. This occurred because a smaller σ allows PTopkDivFinder to mine more rules, enlarging the search space; however, the number of rules is limited and will be fixed when σ reaches a value (*e.g.,* 10^{-4}). Nevertheless, PTopkDivFinder outperforms PTopKDivMiner, PTopk-Miner and RANDOM by 0.132, 0.413 and 0.24 on average, up to 0.23, 0.636 and 0.557, respectively.

Varying γ. We varied the confidence threshold γ from 0.75 to 0.95 in Fig. 2(h). We can see that PTopkDivFinder performs stably with a large γ (*e.g.,* ≥ 0.75). It beats PTopKDivMiner, PTopk-Miner and RANDOM, *e.g.,* its $F(\cdot)$ on Airport is 0.95 on average, as opposed to 0.2, 0.38 and 0.31 of the three. This confirms the validity of PTopkDivFinder.

Varying λ. We tested the impact of the parameter λ on the performance of PTopkDivFinder in Fig. 2(i), by varying λ from 0.1 to 0.8. To illustrate better, we report the score of relevance and diversity metrics instead of $F(\cdot)$. From the figure, when λ increases, the diversity (resp. relevance) score increases (resp. decreases), since a larger λ indicates that we consider more (resp. less) about the diversity (resp. relevance) of rules in Σ. This confirms that λ provides an interpretable way to trade off relevance and diversity in practice.

Exp-3: Efficiency of TopkDivSelector**.** We first compared the efficiency of TopkDivSelector and OPT in the default setting, followed by the impact of the parameter k on the efficiency.

Comparison vs. baselines. As illustrated in Fig. 2(j), TopkDivSelector is much faster than OPT, *e.g.,* it takes 30s to execute the algorithm on average, at least 3060× faster than OPT. Note that, we report the time of OPT as 3 hours if it cannot terminate in this period.

Varying k. We report the runtime of TopkDivSelector and OPT as k varied from 1 to 40 (Fig. 2(k)). While both become slower with larger k, TopkDivSelector consistently beats OPT, *e.g.,* it needs 28s on average, 260× faster than OPT.

Exp-4: Efficiency of PTopkDivFinder**.** We report the time of PTopkDivFinder and baselines in default settings, as well as the time under different parameters.

Comparison vs. baselines. As illustrated in Fig. 2(l), PTopkDivFinder is efficient on all datasets. It beats all baselines, *e.g.,* it is 14.4×, 1.87× and >100× faster than PTopKDivMiner, PTopk-Miner and REEFinder on average, respectively. Moreover, PTopkDivFinder outperforms all its variants, *e.g.,* it is on average 8.9×, 7.8× and 24.1× faster than $\text{PTopkDivFinder}_{\text{nop}}$, $\text{PTopkDivFinder}_{\text{noOpt}}$ and $\text{PTopkDivFinder}_{\text{nop}}^{\text{noOpt}}$, up to 24.8×, 13.9× and 54×, respectively. These verify the efficiency of PTopkDivFinder equipped with effective optimizations.

Varying k. We varied the number k of discovered rules from 1 to 40 in Fig. 2(m). From the results, REEFinder and DCFinder are insensitive to k, since they mine all rules regardless of k. For other approaches, they need more time when k is larger, since more rules need to be discovered. Nevertheless, PTopkDivFinder is the most efficient approach, *e.g.,* it is 14.4×, 9.45× and >100× faster than PTopKDivMiner, $\text{PTopkDivFinder}_{\text{nop}}^{\text{noOpt}}$ and REEFinder, on average.

Varying n. Varying the number n of machines from 4 to 20 in Fig. 2(n), all methods need less time with more machines, as expected. Nevertheless, PTopkDivFinder still beats all baselines and variants. This indicates that PTopkDivFinder is scalable in a large dataset when multiple machines are available.

Varying $|\mathcal{D}|$. We varied the scaling factor of $\mathcal{D}$ from 20% to 100% in Fig. 2(o), *i.e.,* we changed the number of tuples from $20\%|\mathcal{D}|$ to $100\%|\mathcal{D}|$. As expected, all approaches take longer when $\mathcal{D}$ contains more tuples. Nevertheless, PTopkDivFinder beats all baselines and variants, *e.g.,* it is on average 49.7× faster than $\text{PTopkDivFinder}_{\text{nop}}^{\text{noOpt}}$.

Varying ℓ_{max}. We varied the maximum level ℓ_{max} in rule discovery. As shown in Fig. 2(p), it takes more time for all approaches when ℓ_{max} increases. This is because a larger ℓ_{max} enables to discover more complex rules in a larger search space. PTopkDivFinder takes the least time, *e.g.,* it only needs dozen of seconds on Hospital even when $\ell_{\text{max}} = 10$. This verifies the effectiveness of our pruning and optimization strategies.

We also tested other parameters (*e.g.,* σ and γ), where PTopkDivFinder consistently runs faster than all baselines.

Per-level overhead. We instrumented PTopkDivFinder on Inspection with $\ell_{\text{max}} = 10$ and reported per-level costs in

TABLE VI
PER-LEVEL OVERHEAD OF PTopkDivFinder ON Inspection ($\ell_{\mathsf{max}} = 10$)

Level	Stage	$\lvert Q\rvert$ (or $\lvert Q_{\mathsf{next}}\rvert$)	#WorkUnits	Broadcast (MB)	Aggregate (MB)	Coordinator time (ms)	Worker wait (avg/max, ms)	Worker compute (avg/max, ms)
1	Rule validation	588	70	0.0015	0.218	total: 370,581	2,614.3 / 21,114.4	45,970.5 / 352,632.5
	Generate Q_{next}	11,412	–	–	0.238	261	–	–
2	Rule validation	925	128	458.101	0.360	total: 332,663	1,661.6 / 6,318.6	48,037.3 / 285,269.3
	Generate Q_{next}	27,109	–	–	0.590	193	–	–
3	Rule validation	480	162	465.710	0.328	total: 250,156	811.3 / 5,505.1	23,418.7 / 238,156.6
	Generate Q_{next}	50,125	–	474.286	0.886	b/a: 13,645 / 24,258	318.7 / 3,890.6	313.0 / 14,317.5
4	Rule validation	153	97	465.713	0.172	total: 98,428	1,256.5 / 10,045.8	11,528.4 / 92,321.0
	Generate Q_{next}	69,722	–	474.306	1.395	b/a: 13,155 / 15,532	211.0 / 5,229.8	22.7 / 381.2
5	Rule validation	26	26	465.714	0.043	total: 43,873	816.9 / 3,834.4	5,304.4 / 31,174.1
	Generate Q_{next}	73,855	–	474.315	1.865	b/a: 13,360 / 30,225	340.1 / 24,310.6	41.8 / 699.0

Notes. (1) For the *Rule validation* stage, *Coordinator time* is the end-to-end wall-clock time of the corresponding Spark stage (including scheduling, task execution, and shuffle aggregation). (2) For the *Generate Q_{next}* stage, *Coordinator time* is reported as the coordinator-side breakdown *b/a* for *broadcast/aggregate* when executed in parallel; when executed serially on the coordinator (Levels 1–2), it reports the local generation time, and *Aggregate (MB)* reports the size of Q_{next}, as parallelization overhead dominates at small $|Q|$.

Table VI. Experiments ran on 17 machines (1 coordinator and 16 workers), each with 64GB RAM and 24 processors @2.50GHz. The table summarizes (i) broadcast payload, (ii) aggregate payload, (iii) coordinator time per stage (rule validation wall-clock; broadcast/aggregate for generating Q_{next}), and (iv) worker wait/compute statistics. Although $\ell_{\mathsf{max}} = 10$, the search often terminates early due to effective pruning; hence, we reported only the levels that are actually executed.

Across executed levels, coordinator aggregation is not a bottleneck. For generating Q_{next}, the aggregated payload is small (0.238–1.865 MB), and the broadcast+aggregation time is 110.6s. In contrast, rule validation takes 1095.7s in total wall-clock time (including scheduling, worker execution, and shuffle aggregation), indicating that the end-to-end runtime is dominated by validation rather than coordinator aggregation in Q_{next} generation. Moreover, worker computation dominates in the validation stage; the average compute time is 29.9s, compared with 1.39s average wait ($\approx 21.5\times$). The maxima (352.6s compute *vs.* 21.1s wait) reflect occasional stragglers.

Notably, although $|Q_{\mathsf{next}}|$ can grow rapidly during candidate generation, the number of candidates that proceed to the next-level rule-validation stage (*i.e.,* $|Q|$) is much smaller due to pruning/filtering before validation, substantially reducing the effective search space. We further reduce scheduling overhead by grouping candidate rules with the same precondition X into one work unit, yielding fewer work units than $|Q|$.

Summary. We found the following. (1) Based on the correlation model $\mathcal{M}_{\mathsf{corr}}$, TopkDivSelector and PTopkDivFinder are able to obtain a set of diversified top-k rules, consistently outperforming the baselines on normalized score $F(\cdot)$ and rule coverage $rCov(\cdot)$. For TopkDivSelector, $F(\cdot)$ and $rCov(\cdot)$ exceed Random by 24.29% and 51.6% on average. For PTopkDivFinder, $F(\cdot)$ exceeds PTopKDivMiner, PTopk-Miner, and Random by 36.09%, 43.31%, and 43.57%, respectively, with consistently higher $rCov(\cdot)$. (2) TopkDivSelector is efficient, it is at least 3060$\times$ faster than OPT with default setting over all datasets, on average. (3) PTopkDivFinder is efficient, *e.g.,* it beats REEFinder by $>$100$\times$; with optimization strategies, it achieves comparable speed with PTopk-Miner that do not consider diversity in discovery; with a user-guided embedding-based objective function, it speeds up PTopKDivMiner by 14.4$\times$ on average, up to 35.57$\times$. (4) The proposed pruning strategies are effective, *e.g.,* PTopkDivFinder is 24.1$\times$ faster than $\mathsf{PTopkDivFinder}^{\mathsf{noOpt}}_{\mathsf{nop}}$ on average, up to 54$\times$.

VII. RELATED WORK

Rule discovery. Rule discovery has been extensively studied for many years. Various algorithms are proposed for mining rules such as FDs [4], CFDs [5], MDs [6], DCs [7] and REEs [8], [9]. Based on the discovery manner, these works can be further classified as: (1) levelwise search for (a) FDs, *e.g.,* TANE [39], FUN [40], FD_mine [41], HyFD [42], DynFD [43], Depmine [44] and SMFD [45]; and (b) MDs, *e.g.,* CTANE [46], tableau generation [47] and similarity thresholds [48]; (2) depth-first search (DFS) for (a) FDs, *e.g.,* DFD [49] utilizes random walk to mine FDs and FastFDs [50] uses heuristic DFS to mine FDs; (b) CFDs, *e.g.,* FastCFDs [46] extends FastFDs for discovering CFDs; and (c) DCs, *e.g.,* FastDC [12], Hydra [11], DCFinder [29] and ADCMiner [13]; (3) hybrid approaches (*e.g.,* HyMD [51] and MDedup [52]) that combine the idea of both levelwise and DFS strategies; and (4) ML-based approaches, *e.g.,* [53]–[59].

Top-k Rule discovery. Top-k rule discovery has been studied in some works, *e.g.,* [60]–[62] aim to mine top-k association rule; [63] discovers top-k covering rules for gene expression; [64] mines top-k sequential rules from sequence databases; [65] discovers specialized top-k preference rules from user-rated reviews; and [10] trains a subjective model to learn user preferences and return top-k relevant rules, by asking users to label some rule pairs. Moreover, [66], [67] summarizes the works for mining association rules and sequential patterns.

Diversified Top-k algorithms. There are plenty of works on diversified top-k algorithms. The diversity maximization problem is initially studied in subset selection [68]. There are some works on diversity metrics [28] and more general objectives [69]–[71]. Diversified top-k algorithms have been studied in various fields, *e.g.,* clique search [72], subgraph query [73], shortest path [74], graph pattern mining [75], information retrieval [76], and rule discovery [15]. There are also works on top-k results diversifying by post-processing [77] and the complexity of diversification [23]–[25].

This work differs from existing works in the following aspects. (1) We develop a semi-automatic user-friendly interaction scheme to capture user preferences at low cost. Using

preference templates, we learn user-guided embeddings for predicates and rules, and formulate an embedding-based objective function. (2) We study the problem of selecting diversified top-k rules from a given set of rules, providing a plugin of any rule discovery algorithm. (3) We further study diversified top-k rule discovery from scratch, and develop effective optimization strategies to accelerate the proposed algorithm.

VIII. Conclusion

In this work, we study the problem of selecting and discovering diversified top-k rules, respectively. Its novelty consists of the following: (1) a user-friendly interaction scheme that effectively captures user preferences; (2) a novel user-guided embedding-based objective function that jointly optimizes the relevance and diversity of rules; (3) a diversified rule selection algorithm that can be applied in any existing rule discovery algorithm for selecting diversified top-k rules; and (4) a diversified rule discovery algorithm with effective optimizations. We verified the effectiveness and efficiency of the proposed approaches via extensive evaluation on real-world datasets.

A key limitation is that embedding quality and rule ranking depend on the preference templates used to train $\mathcal{M}_{\mathsf{corr}}$. In our target scenarios, the built-in templates together with a small number of user-defined templates are typically informative and aligned well with user preferences, as shown in our experiments. However, in extreme settings (*e.g.,* highly complex attribute relationships or strongly conflicting user templates), a small template set may fail to capture domain semantics, causing $\mathcal{M}_{\mathsf{corr}}$ to learn only a coarse approximation and degrade ranking quality. The system can still operate using built-in templates and statistical properties of rules, but may not reflect fine-grained, domain-specific semantics. Future work includes diagnosing and reconciling inconsistent preferences, enriching domain-aware template libraries, and supporting incremental diversified top-k discovery in dynamic settings (e.g., streams).

IX. Acknowledgements

This work was supported by NSFC 62472289, National Key R&D program of China 2022YFC3800602, Shenzhen Natural Science Foundation 20220810142731001 and Guangdong Province Key Laboratory of Popular High Performance Computers 2017B030314073.

References

[1] X. Zhu, X. Ao, Z. Qin, Y. Chang, Y. Liu, Q. He, and J. Li, "Intelligent financial fraud detection practices in post-pandemic era," *The Innovation*, vol. 2, no. 4, 2021.

[2] C. Li, F. Hao, L. Zhao, L. Song, and X. Dong, "Analysis of medical and healthcare data based on positive and negative association rules," in *ICNC-FSKD*, 2017, pp. 1559–1564.

[3] W. Fan, X. Wang, Y. Wu, and J. Xu, "Association rules with graph patterns," *PVLDB*, vol. 8, no. 12, pp. 1502–1513, 2015.

[4] E. F. Codd, "Relational completeness of data base sublanguages," *IBM Corporation*, 1972.

[5] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for capturing data inconsistencies," *TODS*, vol. 33, no. 2, pp. 6:1–6:48, 2008.

[6] W. Fan, H. Gao, X. Jia, J. Li, and S. Ma, "Dynamic constraints for record matching," *VLDB J.*, vol. 20, no. 4, pp. 495–520, 2011.

[7] M. Arenas, L. Bertossi, and J. Chomicki, "Consistent query answers in inconsistent databases," in *PODS*, 1999.

[8] W. Fan, C. Tian, Y. Wang, and Q. Yin, "Parallel discrepancy detection and incremental detection," *PVLDB*, vol. 14, no. 8, pp. 1351–1364, 2021.

[9] W. Fan, P. Lu, and C. Tian, "Unifying logic rules and machine learning for entity enhancing," *Science China Information Sciences*, 2020.

[10] W. Fan, Z. Han, Y. Wang, and M. Xie, "Discovering top-k rules using subjective and objective criteria," *SIGMOD*, vol. 1, no. 1, pp. 1–29, 2023.

[11] T. Bleifuß, S. Kruse, and F. Naumann, "Efficient denial constraint discovery with hydra," *PVLDB*, vol. 11, no. 3, pp. 311–323, 2017.

[12] X. Chu, I. F. Ilyas, and P. Papotti, "Discovering denial constraints," *PVLDB*, vol. 6, no. 13, pp. 1498–1509, 2013.

[13] E. Livshits, A. Heidari, I. F. Ilyas, and B. Kimelfeld, "Approximate denial constraints," *PVLDB*, vol. 13, no. 10, pp. 1682–1695, 2020.

[14] W. Fan, Z. Han, Y. Wang, and M. Xie, "Parallel rule discovery from large datasets by sampling," in *SIGMOD*, 2022, pp. 384–398.

[15] W. Fan, Z. Han, M. Xie, and G. Zhang, "Discovering top-k relevant and diversified rules," *SIGMOD*, vol. 2, no. 4, pp. 1–28, 2024.

[16] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," *Advances in neural information processing systems*, vol. 6, 1993.

[17] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *CVPR*, vol. 2, 2006, pp. 1735–1742.

[18] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Re, "Snorkel: Rapid training data creation with weak supervision," *PVLDB*, vol. 11, no. 3, p. 269, 2017.

[19] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *EMNLP-IJCNLP*, 2019, pp. 3980–3990.

[20] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[21] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan, "Embdi: generating embeddings for relational data integration," in *SEDB*, 2021.

[22] A. Silberschatz and A. Tuzhilin, "What makes patterns interesting in knowledge discovery systems," *TKDE*, pp. 970–974, 1996.

[23] S. Gollapudi and A. Sharma, "An axiomatic approach for result diversification," in *WWW*, 2009.

[24] M. R. Vieira, H. L. Razente, M. C. Barioni, M. Hadjieleftheriou, D. Srivastava, C. Traina, and V. J. Tsotras, "On query result diversification," in *ICDE*, 2011, pp. 1163–1174.

[25] T. Deng and W. Fan, "On the complexity of query result diversification," *ACM Trans. Database Syst.*, vol. 39, no. 2, pp. 15:1–15:46, 2014.

[26] R. Hassin, S. Rubinstein, and A. Tamir, "Approximation algorithms for maximum dispersion," *Operations Research Letters*, vol. 21, no. 3, pp. 133–137, 1997.

[27] B. Birnbaum and K. J. Goldman, "An improved analysis for a greedy remote-clique algorithm using factor-revealing lps," *Algorithmica*, vol. 55, no. 1, pp. 42–59, 2009.

[28] B. Chandra and M. M. Halldórsson, "Approximation algorithms for dispersion problems," *Journal of algorithms*, vol. 38, no. 2, pp. 438–465, 2001.

[29] E. H. M. Pena, E. C. de Almeida, and F. Naumann, "Discovery of approximate (and exact) denial constraints," *PVLDB*, vol. 13, no. 3, pp. 266–278, 2019.

[30] S. Kruse and F. Naumann, "Efficient discovery of approximate dependencies," *PVLDB*, vol. 11, no. 7, pp. 759–772, 2018.

[31] E. H. M. Pena, E. C. de Almeida, and F. Naumann, "Discovery of approximate (and exact) denial constraints," *PVLDB*, vol. 13, no. 3, pp. 266–278, 2019.

[32] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *VLDB*, 1994, p. 487–499.

[33] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré, "HoloClean: Holistic data repairs with probabilistic inference," *PVLDB*, 2017.

[34] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: Extraction and mining of academic social networks," in *SIGKDD*, 2008, pp. 990–998.

[35] Z. Han, "Source code and datasets," https://github.com/philo-vanguard/PTopkDivFinder, 2025.

[36] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *SIGKDD*, 2016, pp. 785–794.

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

[38] X. Ge and P. K. Chrysanthis, "Efficient prefdiv algorithms for effective top-k result diversification," in *EDBT*, 2020, pp. 335–346.

[39] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen, "TANE: an efficient algorithm for discovering functional and approximate dependencies," *Comput. J.*, 1999.

[40] N. Novelli and R. Cicchetti, "Fun: An efficient algorithm for mining functional and embedded dependencies," in *ICDT*, 2001, pp. 189–203.

[41] H. Yao, H. Hamilton, and C. Butz, "Fd_mine: discovering functional dependencies in a database using equivalences, canada," in *ICDM*, 2002, pp. 1–15.

[42] T. Papenbrock and F. Naumann, "A hybrid approach to functional dependency discovery," in *SIGMOD*, 2016.

[43] P. Schirmer, T. Papenbrock, S. Kruse, F. Naumann, D. Hempfing, T. Mayer, and D. Neuschäfer-Rube, "Dynfd: Functional dependency discovery in dynamic datasets," in *EDBT*, 2019.

[44] S. Lopes, J.-M. Petit, and L. Lakhal, "Efficient discovery of functional dependencies and Armstrong relations," in *EDBT*, 2000, pp. 350–364.

[45] C. Ge, I. F. Ilyas, and F. Kerschbaum, "Secure multi-party functional dependency discovery," *PVLDB*, vol. 13, no. 2, pp. 184–196, 2019.

[46] W. Fan, F. Geerts, J. Li, and M. Xiong, "Discovering conditional functional dependencies," *TKDE*, vol. 23, no. 5, pp. 683–698, 2011.

[47] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu, "On generating near-optimal tableaux for conditional functional dependencies," *PVLDB*, 2008.

[48] S. Song and L. Chen, "Discovering matching dependencies," in *CIKM*, 2009, pp. 1421–1424.

[49] Z. Abedjan, P. Schulze, and F. Naumann, "Dfd: Efficient functional dependency discovery," in *CIKM*, 2014, pp. 949–958.

[50] C. M. Wyss, C. Giannella, and E. L. Robertson, "FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances - extended abstract," in *DaWak*, 2001.

[51] P. Schirmer, T. Papenbrock, I. Koumarelas, and F. Naumann, "Efficient discovery of matching dependencies," *RODS*, vol. 45, no. 3, pp. 1–33, 2020.

[52] l. Koumarelas, T. Papenbrock, and F. Naumann, "Mdedup: Duplicate detection with matching dependencies," *PVLDB*, vol. 13, no. 5, pp. 712–725, 2020.

[53] P. A. Flach and I. Savnik, "Database dependency discovery: A machine learning approach," *AI communications*, vol. 12, no. 3, pp. 139–160, 1999.

[54] P. Mandros, M. Boley, and J. Vreeken, "Discovering reliable approximate functional dependencies," in *SIGKDD*, 2017, pp. 355–363.

[55] Y. Zhang, Z. Guo, and T. Rekatsinas, "A statistical perspective on discovering functional dependencies in noisy data," in *SIGMOD*, 2020, pp. 861–876.

[56] K. Cheng, J. Liu, W. Wang, and Y. Sun, "Rlogic: Recursive logical rule learning from knowledge graphs," in *SIGKDD*, 2022, pp. 179–189.

[57] Y. Zhang, Z. Guo, and T. Rekatsinas, "A statistical perspective on discovering functional dependencies in noisy data," in *SIGMOD*, 2020, pp. 861–876.

[58] R. Singh, V. V. Meduri, A. K. Elmagarmid, S. Madden, P. Papotti, J. Quiané-Ruiz, A. Solar-Lezama, and N. Tang, "Synthesizing entity matching rules by examples," *PVLDB*, vol. 11, no. 2, pp. 189–202, 2017.

[59] H. Köpcke, A. Thor, and E. Rahm, "Evaluation of entity resolution approaches on real-world match problems," *PVLDB*, vol. 3, no. 1-2, pp. 484–493, 2010.

[60] G. I. Webb and S. Zhang, "k-optimal rule discovery," *Data Mining and Knowledge Discovery*, vol. 10, no. 1, pp. 39–79, 2005.

[61] P. Fournier-Viger and V. S. Tseng, "Mining top-k non-redundant association rules," in *ISMIS*, 2012, pp. 31–40.

[62] X. Liu, X. Niu, and P. Fournier-Viger, "Fast top-k association rule mining using rule generation property pruning," *Applied Intelligence*, vol. 51, pp. 2077–2093, 2021.

[63] G. Cong, K.-L. Tan, A. K. Tung, and X. Xu, "Mining top-k covering rule groups for gene expression data," in *SIGMOD*, 2005, pp. 670–681.

[64] P. Fournier-Viger and V. S. Tseng, "Mining top-k sequential rules," in *ADMA*, 2011, pp. 180–194.

[65] Z. Tan, H. Yu, W. Wei, and J. Liu, "Top-k interesting preference rules mining based on maxclique," *Expert Systems with Applications*, vol. 143, p. 113043, 2020.

[66] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, and P. S. Yu, "A survey of parallel sequential pattern mining," *TKDD*, vol. 13, no. 3, pp. 1–34, 2019.

[67] S. K. Solanki and J. T. Patel, "A survey on association rule mining," in *ACCT*, 2015, pp. 212–216.

[68] S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi, "Heuristic and special case algorithms for dispersion problems," *Operations research*, vol. 42, no. 2, pp. 299–310, 1994.

[69] A. Dasgupta, R. Kumar, and S. Ravi, "Summarization through submodularity and dispersion," in *ACL*, 2013, pp. 1014–1022.

[70] A. Borodin, A. Jain, H. C. Lee, and Y. Ye, "Max-sum diversification, monotone submodular functions, and dynamic updates," *TALG*, vol. 13, no. 3, pp. 1–25, 2017.

[71] C. Qian, D.-X. Liu, and Z.-H. Zhou, "Result diversification by multi-objective evolutionary algorithms with theoretical guarantees," *Artificial Intelligence*, vol. 309, p. 103737, 2022.

[72] B. Lyu, L. Qin, X. Lin, Y. Zhang, Z. Qian, and J. Zhou, "Maximum and top-k diversified biclique search at scale," *VLDB J.*, vol. 31, no. 6, pp. 1365–1389, 2022.

[73] H. Ma, S. Guan, C. Toomey, and Y. Wu, "Diversified subgraph query generation with group fairness," in *WSDM*, 2022, pp. 686–694.

[74] Z. Luo, L. Li, M. Zhang, W. Hua, Y. Xu, and X. Zhou, "Diversified top-k route planning in road network," *PVLDB*, vol. 15, no. 11, pp. 3199–3212, 2022.

[75] W. Fan, X. Wang, and Y. Wu, "Diversified top-k graph pattern matching," *PVLDB*, vol. 6, no. 13, pp. 1510–1521, 2013.

[76] Z. Jiang, J.-R. Wen, Z. Dou, W. X. Zhao, J.-Y. Nie, and M. Yue, "Learning to diversify search results via subtopic attention," in *SIGIR*, 2017, pp. 545–554.

[77] L. Qin, J. X. Yu, and L. Chang, "Diversifying top-k results," *PVLDB*, vol. 5, no. 11, 2012.

Ziyan Han received the B.E. degree in Computer Science and Technology from Xidian University, Xi'an, China, in 2018, and the Ph.D. degree in Computer Software and Theory from Beihang University, Beijing, China, in 2025. She has published papers in top-tier database conferences, including SIGMOD and ICDE. Her research interests include data management, data mining, rule discovery, data cleaning, and machine learning.

Wanjia Chen received the B.S. degree in Information and Computing Science from Shenzhen University, Shenzhen, China, in 2019. He worked as a Research Assistant at The Hong Kong University of Science and Technology (Guangzhou) in 2022. His research interests include data management, data mining, and machine learning.

Yunpeng Han received the B.E. degree in Mechanical Engineering from Shijiazhuang Tiedao University, Hebei, China, in 2016, and the M.S. degree in Mechanical Engineering from Florida Institute of Technology, Melbourne, USA, in 2018. He is currently pursuing the Ph.D. degree in Electrical Engineering at Louisiana State University, Baton Rouge, USA. His research interests include data mining, robotics, and intelligent transportation systems.

Rui Mao received the B.S. degree (1997) and the M.S. degree (2000) in Computer Science from the University of Science and Technology of China, and the M.S. degree (2006) in Statistics and the Ph.D. degree (2007) in Computer Science from the University of Texas at Austin. He is currently a Distinguished Professor in the College of Computer Science and Software Engineering at Shenzhen University, and the Executive Director of Shenzhen Institute of Computing Sciences. His research interests mainly focus on metric-space data processing.

Jianbin Qin received the B.E. degree from Northeastern University, China, in 2007, and the Ph.D. degree from the University of New South Wales, Australia, in 2013. He is currently a Distinguished Professor in the College of Computer Science and Software Engineering at Shenzhen University, and a Research Scientist at Shenzhen Institute of Computing Sciences. His research interests include database theory, database systems, similarity query, information retrieval, and data management.