The VLDB Journal (2021) 30:515–536
https://doi.org/10.1007/s00778-021-00653-w

REGULAR PAPER

Mis-categorized entities detection

Shuang Hao[1,2] · Nan Tang[3] · Guoliang Li[2] · Jianhua Feng[2] · Ning Wang[1]

Received: 11 December 2019 / Revised: 30 May 2020 / Accepted: 14 January 2021 / Published online: 6 March 2021

Abstract
Entity categorization, the process of categorizing entities into groups, is an important problem with many applications. However, in practice, many entities are mis-categorized, such as Google Scholar and Amazon products. In this paper, we study the problem of *discovering mis-categorized entities* from a given group of categorized entities. This problem is inherently hard: All entities within the same group have been "well" categorized by the *state-of-the-art* solutions. Apparently, it is nontrivial to differentiate them. We propose a novel *rule-based framework* to solve this problem. It first uses positive rules to compute disjoint partitions of entities, where the partition with the largest size is taken as the correctly categorized partition, namely the *pivot partition*. It then uses negative rules to identify mis-categorized entities in other partitions that are *dissimilar* to the entities in the pivot partition. We describe optimizations on applying these rules and discuss how to generate positive/negative rules. In addition, we propose novel strategies to resolve inconsistent rules. Extensive experimental results on real-world datasets show the effectiveness of our solution.

Keywords Mis-categorized entity · Rule-based framework · Signature · Rule generation · Rule inconsistency

1 Introduction

Categorizing entities into sensible groups is a fundamental mode for many applications. Nevertheless, mis-categorized entities are pervasive, *e.g.,* there exist others' publications in many researchers' Google Scholar pages. A practical problem is *how to (automatically) discover them?*

Example 1 *(Mis-Categorized Google Scholar Papers.)* Consider *Nan Tang*'s sample Google Scholar entities in Fig. 1. Each entity has three attributes: `Title`, `Authors` and `Venue`. Two entities, e_4 and e_6, are mis-categorized—they do not belong to the *Nan Tang* at QCRI. The e_4[`Authors`] does not contain a valid name, and the *Nan Tang* in e_6[`Authors`] is a different person working in chemistry.
[Mis-Categorized Amazon Products.] Consider sample entities in Amazon's product category "*Router*," as shown in Fig. 2. Each entity has four attributes: `Asin` is the ID of product, `Title` is its name, `Also_viewed` contains a list of products viewed together with it, and `Description` describes its features. The entity o_3 is mis-categorized, since it should be in category "*Adapter*" instead of "*Router*."

Entity categorization (EC) is closely related to, but different from, the problem of entity matching (EM). EM is to determine whether two entities are the same, which is often solved by comparing aligned attributes symbolically (or syntactically). EC groups entities more conceptually (or semantically), *e.g.,* e_1 and e_2, in Fig. 1 are not symbolically similar in either `Title`, `Authors`, or `Venue`.

Challenge All entities in the same group already use all attributes of the entities, how can we use the same set of attributes (or signals) to differentiate them?

✉ Guoliang Li
liguoliang@tsinghua.edu.cn

Shuang Hao
haoshuang@bjtu.edu.cn

Nan Tang
ntang@hbku.edu.qa

Jianhua Feng
fengjh@tsinghua.edu.cn

Ning Wang
nwang@bjtu.edu.cn

1 School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China

2 Department of Computer Science and Technology, Tsinghua University, Beijing, China

3 Qatar Computing Research Institute, HBKU, Ar-Rayyan, Qatar

e_1: Title: KATARA: A data cleaning system powered by knowledge bases and crowdsourcing
Authors: Xu Chu, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, **Nan Tang**
Venue: SIGMOD 2015

e_2: Title: Hierarchical indexing approach to support xpath queries
Authors: **Nan Tang**, Jeffrey Xu Yu, M. Tamer Özsu, Kam-Fai Wong
Venue: ICDE 2008

e_3: Title: NADEEF: A generalized data cleaning system
Authors: Amr Ebaid, Ahmed Elmagarmid, Ihab F. Ilyas, **Nan Tang**
Venue: VLDB 2013

e_4: Title: Partition based hierarchical index for text retrieval
Authors: Yunqing Xia, **NJ Tang**, Amir Hussain, Erik Cambria
Venue: SIGIR 2005

e_5: Title: Win: an efficient data placement strategy for parallel xml databases
Authors: **Nan Tang**, Guoren Wang, Jeffrey Xu Yu
Venue: ICPADS 2005

e_6: Title: Extractive and oxidative desulfurization of model oil in polyethylene glycol
Authors: Jianlong Wang, Rijie Zhao, Baixin Han, **Nan Tang**, Kaixi Li
Venue: RSC Advances 1905

Fig. 1 Sample categorized Google Scholar entities

o_1: Asin: B000BTL0OA
Title: **Linksys WRT54GL Wi-Fi Wireless-G Broadband Router**
Also-viewed: "B00004SYNW", "B00004SYLI", "B00004SB92", "B00006I5XC"
Description: Shares a single Internet connection with 4 Ethernet wired

o_2: Asin: B00004SYNW
Title: **D-Link DI-701 Ishare Cable/DSL Internet Sharing Router**
Also-viewed: "B000BTL0OA", "B00004SYLI", "B00004SB92", "B00006I5XC"
Description: Provides 32-user Internet access and file sharing

o_3: Asin: B00004TF4X
Title: **StarTech.com USB to Ethernet LAN Adapter**
Also-viewed: "B00007LTB6", "B0001PFO3C", "B00007KDVK", "B000063XJ7"
Description: Compatible with Gigabit Ethernet networks and powered via USB

o_4: Asin: B00004SYLI
Title: **Netgear RT311 DSL/Cable Internet Gateway Router**
Also-viewed: "B000BTL0OA", "B00004SYNW", "B00004SB92", "B00006I5XC"
Description: Ethernet router allows 32 users to share a single Internet connection

Fig. 2 Sample entities in Amazon's router category

Our Methodology We propose a rule-based framework using positive and negative rules to tackle this problem. Positive rules are used to conservatively find disjoint partitions, such that the entities within the same partition should be categorized together. The partition with the largest size is called the *pivot partition*, which is treated as correctly categorized partition under the practical assumption that the largest one is correct compared with others with much smaller sizes; that is, existing algorithms that produce these groups are not that bad. Negative rules are then used to compare other partitions with the pivot partition to discover dissimilar entities as mis-categorized entities, since the violation of positive rules is not enough to specify these entities should not be categorized together.

Example 2 Consider the entities in Fig. 1 and the following positive rules (φ^+) and negative rules (ϕ^-). We use the terms "similar"/"dissimilar" to indicate that two entities should/should not be in the same category.

φ_1^+: *Two entities are similar, if they have* ≥ 2 *common* `Authors`.
φ_2^+: *Two entities are similar, if they have overlapping in* `Authors` *and their* `Venue` *are in the same field.*
ϕ_1^-: *Two entities are dissimilar, if they have no common* `Author`.
ϕ_2^-: *Two entities are dissimilar, if they have* ≤ 1 *common author in* `Authors` *and their* `Venue` *are in different fields.*

(1) *Positive rules* are used in a disjunctive fashion as $\varphi_1^+ \vee \varphi_2^+$: Two entities belong to the same category if either φ_1^+ or φ_2^+ is *true*. Also, we assume the *transitivity* of categorization: Entities (e_i, e_j) and (e_j, e_k) are categorized together implies that (e_i, e_k) are also categorized together. We can find three partitions P_1 : $\{e_1, e_2, e_3, e_5\}$, P_2 : $\{e_4\}$, P_3 : $\{e_6\}$. For example, e_1 and e_2 are categorized together based on φ_2^+: They have an overlapping author *Nan Tang*, and their venues *SIGMOD* in e_1 and *ICDE* in e_2 are in the same field (see ontology-based similarity in Sect. 3).
(2) The partition P_1 is treated as the pivot partition, because it has the largest size.
(3) *Negative rules* are used either individually, *e.g.,* ϕ_1^-, or in a disjunction, *e.g.,* $\phi_1^- \vee \phi_2^-$: Two entities belong to different categories if either ϕ_1^- or ϕ_2^- is *true*. ϕ_1^- discovers e_4 as mis-categorized because e_4 does not have overlapping in `Authors` with any entity in P_1. $\phi_1^- \vee \phi_2^-$ further discovers e_6, since e_6 only has one common author with the entities in P_1 and its `Venue` is in the field of *Chemical Sciences*, which is different from the field *Computer Science* in P_1. Negative rules are used in sequence: We first use ϕ_1^-, then $\phi_1^- \vee \phi_2^-$, and so on, if more negative rules are available. We provide a scrollbar for the user to confirm the mis-categorized entities from either ϕ_1^- or $\phi_1^- \vee \phi_2^-$, as shown in Fig. 3.

We can see from Example 2 that (i) positive rules need to be conservative, so as not to include mis-categorized entities in the pivot partition; (ii) we apply multiple negative rules by following the "*one does not fit all*" philosophy—no single negative rule can perfectly discover mis-categorized entities for every given group; and (iii) the purpose to apply a sequence of negative rules, which ensures that their outputs are in a monotonic fashion, is user friendly, which allows the user to easily choose the true ones and early stop the execution when there is no new satisfied mis-categorized entities to be discovered.

Applicability First-party tools, developed by the companies, do their business of improving the quality of their products, such as Google Scholar papers and Amazon shopping items, while our techniques actually belong to third-party tools, which can be treated as add-ons to make first-party tools more useful. We have released a Chrome extension called GSCleaner [36] for cleaning Google Scholar pages. Our techniques make it cheaper for users to confirm our suggested mis-categorized entities than selecting them manually from the entire group.

Contributions We summarize our major contributions as follows:

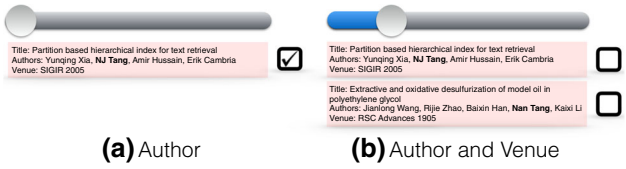


Fig. 3 Tuning discovered mis-categorized entities using a scrollbar

(1) We define a problem of *discovering mis-categorized entities* (Sect. 3).
(2) We present the solution overview of a novel *rule-based framework* to tackle the studied problem (Sect. 4).
(3) We devise signature-based algorithms to improve the performance of applying positive/negative rules (Sect. 5).
(4) We describe effective algorithms to generate positive and negative rules from examples (Sect. 6).
(5) We propose two novel strategies to resolve inconsistencies of positive and negative rules (Sect. 7).
(6) We conducted extensive experiments on real-world datasets to show that our framework can efficiently discover mis-categorized entities with high accuracy (Sect. 8).

2 Related work

This work extends our conference version [35] by including (1) more detailed explanations of signature-based fast solution (Sect. 5), providing more insight to the readers; (2) detailed proofs of all the lemmas and theorems, and some of the proofs are nontrivial and are interesting in their own right; (3) formalized the definitions of soft rules and two strategies to resolve inconsistencies of positive and negative rules (Sect. 7); and (4) a new empirical evaluation of the inconsistency solutions (Sect. 8).

Entity Matching (EM) finds pairs of entities that refer to the same real-world object. EM rules have been widely used [26], and there are solutions to learn EM rules by examples [53,62]. Generic entity matching that provides abstract functions to the users is proposed [7]. Collective entity resolution [8] uses additional information to match entities by an agglomerative clustering algorithm. Optimizing EM performance has also been studied [17,43]. Moreover, there are machine learning-based EM methods [28,54], and different generic methods for automatically selecting training data have been proposed [6,44]. Crowdsourcing has also been applied to EM [13,21,60].

As mentioned earlier, entity categorization (EC) differs from EM in that entities within the same category typically refer to different real-world objects. Consequently, the traditional wisdom of inferring whether entities are the same cannot be directly applied to our studied problem.

Named Entity Disambiguation (NED) is the task of disambiguating named entities mentioned in text of data and link them to their corresponding entries. Wikipedia or DBpedia is widely used as an auxiliary source of information for NED [11,18,30]. Some methods use the graph model to evaluate the relatedness among named entities [3,24,31,34] also introduce crowdsourcing techniques into NED tasks. More recently, neural approaches for NED are proposed, which employ the convolutional neural network model [29,56], recurrent neural network model [27] and the neural embedding techniques [40,64] to disambiguate named entities.

Note that NED aims to distinguish entities that have the same name but refer to different real-world entities. In addition to disambiguation, we also identify entities that have different names (or references) but should be in the same category.

Classification Algorithms Another related topic is classification, where many kinds of classification algorithms have been published, including widely used decision tree [49], random forest [45], native Bayesian classifier [51], SVM [14], neural network classifier [46] and kNN [19]. Clearly, finding a "perfect" classification algorithm that computes two partitions for a group, one for correctly categorized entities and the other for mis-categorized entities, is likely to fail, as will be empirically verified in Sect. 8.

Outlier Detection is the identification of rare objects which are significantly different from the majority of the data. Unsupervised outlier detection techniques [2,12,50,65] usually identify the objects that have few neighbors in an unlabeled dataset as outliers, such as clustering-based outlier detection [38] and density-based techniques [10], while supervised methods [1,48,55,59,63] train a binary classification model with labeled outliers.

In reality, the mis-categorized entity is not equivalent to outlier. What we want to capture is the relationship between the entity and its group. Thus, the mis-categorized entity may be regarded as inlier which is similar to most entities in some attributes, and the correctly categorized entity may be regarded as outlier.

Correlation Clustering is the problem of clustering the data into the optimal number of clusters based on the similarity between the data points, which is to minimize disagreements or maximize agreements between clusters. [5] discuss the NP-completeness proof and present a constant factor approximation algorithm to find the clusters in this setting. The work on minimizing disagreement is extended in [23] where an $\mathcal{O}(\log n)-$approximation algorithm is presented for general graph using the linear programming and region-growing techniques. [15] propose a $4-$approximation algorithm for minimizing disagreements on complete graphs, and [57] give a $0.7666-$approximation algorithm for maximizing agreements using semidefinite programming. A polynomial-time approximation scheme is proved to be exist

on complete graphs and fixed number of clusters [42]. The best polynomial-time approximation algorithm achieves a 2.06−approximation on complete and complete k-partite graphs by linear programming [16]. Correlation clustering is the key technique in a scenario where the relationships between the data points are known.

3 Problem and notation

Problem Given a group **G** of entities that have been categorized together by existing algorithms, the problem of *discovering mis-categorized entities* is to find the proper subset $\mathbf{G}' \subset \mathbf{G}$ that should not belong to this group.

Next, we define the notations that are used in this paper.

Entities An *entity* e is defined over a multi-valued relation $R(A_1, \ldots, A_m)$ of m attributes (*a.k.a.* fields or features). Each attribute A_i of an entity can take a list of values. We write $e[A_i]$ the attribute value of e on attribute A_i.

Groups A *group* **G** is a set of entities $\{e_1, \ldots, e_n\}$.

Example 3 (*Entities.*) Consider entities in Fig. 1, which are defined over schema (`Title`, `Authors`, `Venue`). Entity e_1 has three attributes, and e_1[`Authors`] has multi-values. [Groups.] Nan's group is shown in Fig. 1.

Positive Rules A *positive rule* $\varphi^+(e, e')$ is a *conjunction* of predicates: $\varphi^+(e, e') = \bigwedge_{A_i \in R} f_i(A_i) \geq \theta_i$, where $f_i(A_i)$ is a similarity function and θ_i is a threshold. $\varphi^+(e, e')$ is evaluated to be *true* if all predicates $f_i(A_i) \geq \theta_i$ return *true*, **indicating the two entities e and e' are "similar" and should be categorized together**. Otherwise, $\varphi^+(e, e')$ returns *false*, if any of the predicates returns *false*, indicating we *do not know* whether they should be in the same category.

Negative Rules A *negative rule* $\phi^-(e, e')$ is defined similarly: $\phi^-(e, e') = \bigwedge_{A_i \in R} f_i(A_i) \leq \sigma_i$. $\phi^-(e, e')$ is evaluated to be *true* if all predicates $f_i(A_i) \leq \sigma_i$ return *true*, **indicating e and e' are "dissimilar" and should not be categorized together**. Otherwise, $\phi^-(e, e')$ returns *false*, if any of the predicates returns *false*, indicating we *do not know* whether they should be in different categories.

We will simply write φ^+ (resp. ϕ^-) as a positive (resp. a negative) rule, when (e, e') is clear from the context.

Similarity Functions We consider three types of similarity functions $f(\cdot)$ to quantify the similarity between two values.
(i) Set-based It first splits each value into a set of tokens and then utilizes the set-based similarity to quantify the similarity, such as overlap and Jaccard similarity.
(ii) Character-based It measures the similarity of two values based on character transformations, *e.g.,* edit distance.

The above similarity functions have a common limitation that they mainly use the symbolic (or textual) information, while ignoring the semantics, which is important to the entity categorization problem. To this purpose, we propose to use ontology to capture the semantic similarity. Many enterprises are building their own ontologies, such as Google and Amazon. We can use them directly, or build an ontology using existing knowledge-base construction techniques [37].
(iii) Ontology-based Ontology is usually modeled by a tree structure; for example, the ontology for venues of publications provided by Google Scholar Metric[1] is shown in Fig. 4.

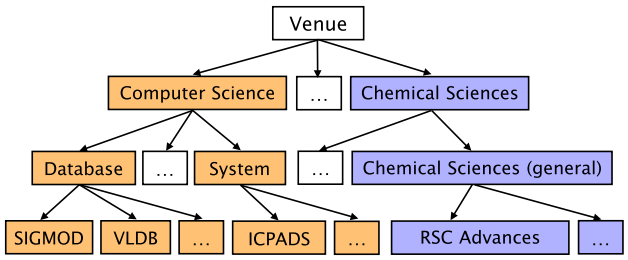


Fig. 4 Google Scholar metrics

Given two entities, we first map them to tree nodes[2]. The ontology similarity is computed based on their lowest common ancestor (`LCA`), which is formally defined as follows.

Ontology Similarity Given two entities e and e', let their mapping nodes on the ontology tree be n and n', respectively. Their *ontology similarity* is defined as: $\frac{2|\mathrm{LCA}(n,n')|}{|n|+|n'|}$, where $\mathrm{LCA}(n, n')$ is the lowest common ancestor of n and n', and $|n|$ is the depth of node n in the tree (the depth of the root is 1). Two entities are *similar* if their similarity is larger than a threshold τ.

Example 4 (*Ontology Similarity.*) Consider the tree structure of venues in Fig. 4 and two nodes *SIGMOD* and *VLDB*. They have rather low string similarity. However, they have a high ontology similarity $\frac{3}{4}$, because their depths are both 4 and their `LCA` is *Database* with depth 3.

The rules in Example 2 can be formulated as follows:

$$\varphi_1^+ : f_{ov}(\texttt{Authors}) \geq 2$$
$$\varphi_2^+ : f_{ov}(\texttt{Authors}) \geq 1 \wedge f_{on}(\texttt{Venue}) \geq 0.75$$
$$\phi_1^- : f_{ov}(\texttt{Authors}) = 0$$
$$\phi_2^- : f_{ov}(\texttt{Authors}) \leq 1 \wedge f_{on}(\texttt{Venue}) \leq 0.25$$

where f_{ov} denotes overlap similarity (common authors in the above rules) and f_{on} indicates ontology similarity.

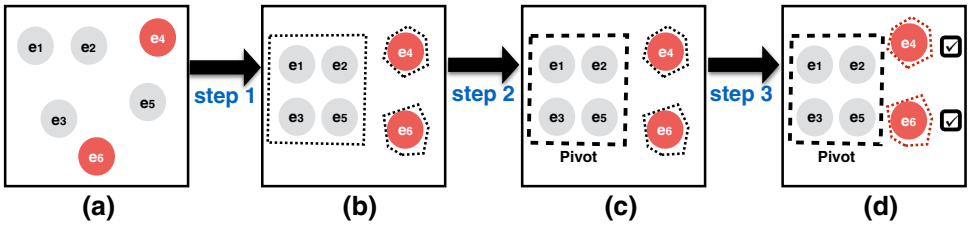


Fig. 5 Solution overview: a rule-based framework

Algorithm 1: DIME: Rule-based Framework

Input: a group $\mathbf{G} = \{e_1, ..., e_n\}$, a set of positive rules $\{\varphi_1^+, \ldots \varphi_x^+\}$, a set of negative rules $\{\phi_1^-, \ldots \phi_y^-\}$
Output: mis-categorized entities $\mathbf{G}^-$
// Step 1: Computing Disjoint Partition Set $\mathbf{P}$
1 Construct a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathbf{G}$, $\mathcal{E} = \emptyset$;
2 **for each** *entity pair* $(e, e') \in \mathbf{G} \times \mathbf{G}$ **do**
3 **if** $\exists \varphi_i^+$ *such that* $\varphi_i^+(e, e')$ *returns* true **then**
4 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(e, e')\}$;
5 Compute the connected components of $\mathcal{G}$;
6 Let $\mathbf{P}$ denote the set of connected components in $\mathcal{G}$;
// Step 2: Identifying The Pivot Partition P^*
7 Let $P^* \in \mathbf{P}$ (the one with the largest size) be pivot partition;
// Step 3: Discovering Mis-Categorized Entities $\mathbf{G}^-$
8 **for each** *partition* $P \in \mathbf{P} \setminus \{P^*\}$ **do**
9 **if** $\exists (e \in P, e^* \in P^*, \phi^- \in \Sigma^-)$, $\phi^-(e, e^*)$ *returns* true **then**
10 $\mathbf{G}^- \leftarrow \mathbf{G}^- \cup P$;
11 **return** $\mathbf{G}^-$

4 A rule-based framework

Our framework DIME is described in Algorithm 1 and depicted in Fig. 5. In a nutshell, it first uses a set of positive rules $\Sigma^+ = \{\varphi_1^+, \cdots, \varphi_x^+\}$ as a disjunction (*i.e.,* $\varphi_1^+ \vee \cdots \vee \varphi_x^+$) on a group of entities $\mathbf{G} = \{e_1, \ldots, e_n\}$ to compute disjoint partitions (step 1; lines 2-6). The partition with the largest size is the *pivot partition* (line 7). Given a set of negative rules $\Sigma^- = \{\phi_1^-, \cdots, \phi_y^-\}$, we either apply the first rule ϕ_1^-, or jointly use ϕ_1^- with the other negative rules in sequence as $\phi_1^- \vee \phi_2^-$, $\phi_1^- \vee \phi_2^- \vee \phi_3^-$, and so on, to discover mis-categorized entities (lines 8-11).

Step 1: Computing Disjoint Partitions Positive rules are used to group entities into partitions. Two cases are considered to put entities e and e' in the same partition. (i) e and e' satisfy a positive rule; or (ii) e and e' satisfy transitivity—there exists another entity e'' such that both (e, e'') and (e', e'') match.

For case (i), we enumerate every entity pair (e, e') and every rule φ^+, and check whether $\varphi^+(e, e')$ returns true. For case (ii), we first construct a graph, where the vertices are entities and edges are entity pairs that satisfy a positive rule (computed from case (i)), and then compute its connected components. Clearly, the entities in the same connected component satisfy the transitivity and form a partition.

The complexity of checking whether an entity pair satisfies a positive rule depends on the similarity functions used in the rule. For overlap, the complexity of computing the similarity is $\mathcal{O}(|e| + |e'|)$, where $|e|$ is the size of e. For edit distance, the complexity of computing the similarity is $\mathcal{O}(\theta \min(|e|, |e'|)$, where θ is the similarity threshold and $|e|$ is the length of e. For ontology, the complexity of computing the similarity is $\mathcal{O}(|d_e| + |d_{e'}|)$, where $|d_e|$ is the depth of e's corresponding node in the tree structure. For ease of presentation, suppose the complexity of checking whether an entity pair satisfies a positive rule is $\mathcal{O}(\upsilon)$. The time complexity of checking every entity pair and every positive rule is $\mathcal{O}(n^2 \upsilon |\Sigma^+|)$, where n is the number of entities and $|\Sigma^+|$ is the number of positive rules. The time complexity of computing connected components (*e.g.,* by a depth-first traversal) is the number of vertices and edges in the graph. Thus, the overall complexity is $\mathcal{O}(n^2 \upsilon |\Sigma^+|)$.

Step 2: Identifying the Pivot Partition The pivot partition $P^* \in \mathbf{P}$ is the one with the largest size, which is treated as the correctly categorized partition.

Step 3: Discovering Mis-Categorized Entities Given the set $\mathbf{P}$ of partitions and the pivot partition P^*, we use the negative rules to mark whether another partition $P \in \mathbf{P} \setminus \{P^*\}$ is a wrongly categorized partition, such that if P is, all entities in P are reported as mis-categorized. To discover the mis-categorized partitions, we enumerate every entity pair $(e^* \in P^*, e \in P)$, and every negative rule $\phi^- \in \Sigma^-$, if $\phi^-(e, e^*)$ returns true, we mark P as a wrongly categorized partition. The complexity of this algorithm is $\mathcal{O}(n^2 \upsilon |\Sigma^-|)$.

The GUI for Scrolling with Multiple Negative Rules Depending on user's capacity, we provide user with a GUI to check manually the outputs of multiple combinations of negative rules, which works pretty well for Google Scholar [36]. The benefits are as follows: (1) The positive/negative rules are provided—the user does not need to know how they are generated (see Sect. 6 for more details). (2) It is cheaper to manually check our suggested mis-categorized entities than checking the entire group, since the number of mis-categorized entities is typically much smaller than the total number of entities in a group; that is, $|\mathbf{G}^-| \ll |\mathbf{G}|$.

[1] https://scholar.google.com/citations?view_op=top_venues.

[2] Here, we use exact string matching for example. We can also use approximate matching based on similarity functions.

Example 5 (*Rule-based Framework.*) Consider entities in Example 1, which are shown in Fig. 5a.

(1) Consider the positive rules φ_1^+ and φ_2^+ in Example 2. Using $\varphi_1^+ \vee \varphi_2^+$ we can find three partitions P_1 : $\{e_1, e_2, e_3, e_5\}$, P_2 : $\{e_4\}$ and P_3 : $\{e_6\}$, as shown in Fig. 5b.
(2) The pivot partition $P^* = P_1$ is surrounded by the dashed rectangle in Fig. 5c.
(3) Using negative rules $\phi_1^- \vee \phi_2^-$ in Example 2, we can discover mis-categorized entities e_4 and e_6, as shown in Fig. 5d (see Example 2 for more explanations about how negative rules are used and see Fig. 3 for the usage of a scrollbar that applies either ϕ_1^- or $\phi_1^- \vee \phi_2^-$).

Also, it is practical to make the following assumptions.

ASSUMPTIONS. (1) The *pivot partition*—the partition with the largest size—only contains truly categorized entities. The rationality is that compared with other partitions with much smaller sizes, the pivot partition is more likely to be correct; that is, existing algorithms that produce these groups are not that bad. (2) We assume the *transitivity*: If e and e' should be categorized together, and e' and e'' should be categorized together, then e and e'' should also be categorized together. The "transitivity" is commonly used as a soft rule [8,58,61], since this assumption can significantly improve the computational efficiency without sacrificing much accuracy in practice. (3) The positive and negative rules should be consistent. More concretely, any two entities in the same partition will not satisfy the negative rules, and any two entities in different partitions will not satisfy the positive rules. How to resolve an inconsistent rule set will be discussed in Sect. 7.

5 Signature-based fast solution

The naïve method of applying positive/negative rules is by enumerating all pairs of entities—evidently an expensive solution. To cope with this issue, we propose a fast signature-based framework DIME$^+$ (Sect. 5.1). We also study how to generate signatures (Sect. 5.2). Finally, we present algorithms for processing positive rules (Sect. 5.3) and negative rules (Sect. 5.4).

5.1 A signature-based framework

We propose a *filter-verification* framework to efficiently find entity pairs satisfying given positive or negative rules.

Signatures of Entities Signatures are substrings (or ancestor nodes in the ontology hierarchy) of entities that can be used to prune pairs of entities that cannot satisfy a positive/negative rule. Intuitively, two entities match, only if they share enough common signatures. For example, consider the three venues *VLDB*, *PVLDB*, *SIGMOD*, and one predicate requires that their edit distance is no greater than 1. Then, their 2-length substrings can be taken as signatures, i.e., {*VL, LD, DB*}, {*PV, VL, LD, DB*}, and {*SI, IG, GM, MO, OD*}. *VLDB* and *PVLDB* are possibly similar since they share 3 signatures; *PVLDB* and *SIGMOD* cannot be similar since they have no common signature and there is no need to compute their real similarity.

Efficient Algorithms for Positive Rules The "*filter*" step: We generate signatures for each entity *w.r.t.* each positive rule: If two entities satisfy a positive rule, they must share a common signature. Thus, we can take the pairs of entities that share common signatures as candidate pairs (and other pairs that do not share common signatures can be safely pruned). To find such candidate pairs, we build a signature-based inverted index for each positive rule, which keeps a mapping from a signature to an inverted list of entities that contain the signature. Then, the entity pair on each inverted list is taken as a candidate pair. We will describe how to generate signatures in Sect. 5.2.

The "*verification*" step: We verify the candidate pairs by computing their real similarities. We use the transitivity to avoid computing the similarities of unnecessary pairs: If (e, e') and (e', e'') are categorized together, we can infer that (e, e'') should also be categorized together without further verifying (e, e''), which will be elaborated in Sect. 5.3.

Efficient Algorithms for Negative Rules A negative rule is defined on top of partitions. A partition P is dissimilar with the pivot partition P^* if there exists a pair of entities $(e \in P, e^* \in P^*)$ and a negative rule ϕ^- such that $\phi^-(e, e^*)$ returns true. To effectively check whether two partitions are dissimilar, we also utilize the signatures in the "*filter*" step: If two entities share common signatures, they may be similar; if two entities do not share any signature, they must be dissimilar. So, we generate the signature of a partition which is the union of signatures of entities in the partition. If two partitions have no common signatures, they satisfy the negative rule; otherwise, we verify the pairs of entities across these two partitions.

The "*verification*" step: We first verify the entity pair with large probability to be dissimilar, because once we find a pair satisfying the negative rule, we do not need to verify the other pairs, which will be discussed in Sect. 5.4.

A Signature-based Algorithm We present DIME$^+$ in Algorithm 2. It first utilizes the signatures to compute disjoint partitions (lines 1–11 for step 1). It enumerates every entity e in $\mathbf{G}$, computes its signature g, and builds inverted list $L_{\varphi_i^+}(g)$ for positive rule φ_i^+, which keeps the list of entities that contain g (lines 1–2). It then computes the entity pairs in each inverted list as candidate pairs (lines 3–4). Next it

Algorithm 2: DIME$^+$: Signature-based Algorithm

Input: a group $\mathbf{G} = \{e_1, ..., e_n\}$, a set of positive rules $\{\varphi_1^+, \ldots \varphi_x^+\}$, a set of negative rules $\{\phi_1^-, \ldots \phi_y^-\}$
Output: mis-categorized entities $\mathbf{G}^-$
// Step 1: Computing Disjoint Partitions $\mathbf{P}$
1 **for each** *signature g of each* $e \in \mathbf{G}$ *wrt each* φ_i^+ **do**
2 $L_{\varphi_i^+}(g) \leftarrow L_{\varphi_i^+}(g) \cup \{e\}$;
3 **for each** *entity pair* $e, e' \in L_{\varphi_i^+}(g)$ **do**
4 $\mathcal{C} \leftarrow \mathcal{C} \cup \{(e, e')\}$;
5 sort the candidate set $\mathcal{C}$;
6 build a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathbf{G}, \mathcal{E} = \emptyset$;
7 **for each** *candidate* $(e, e') \in \mathcal{C}$ **do**
8 **if** *e,e′ belong to different connect components* **then**
9 **if** $\varphi^+(e, e')$ *returns* true **then**
10 add an edge (e, e') to $\mathcal{G}$;
11 $\mathbf{P} \leftarrow$ the connect components of $\mathcal{G}$;
// Step 2: Identifying The Pivot Partition P^*
12 Let $P^* \in \mathbf{P}$ be the pivot partition; $\mathcal{C} \leftarrow \emptyset$;
// Step 3: Discovering mis-categorized Entities $\mathbf{G}^-$
13 **for each** *signature g of each* $e \in P^*$ *wrt each* ϕ_i^- **do**
14 $L_{\phi_i^-}(P^*) \leftarrow L_{\phi_i^-}(P^*) \cup \{g\}$;
15 **for each** *partition* $P \in \mathbf{P} \setminus \{P^*\}$ **do**
16 **for each** *signature g of* $e \in P$ *wrt* ϕ_i^- **do**
17 $L_{\phi_i^-}(P) \leftarrow L_{\phi_i^-}(P) \cup \{g\}$;
18 **if** $L_{\phi_i^-}(P) \cap L_{\phi_i^-}(P^*) = \emptyset$ **then**
19 $\mathbf{G}^- \leftarrow \mathbf{G}^- \cup P$;
20 **else** add each pair (e, e^*) that $e \in P, e^* \in P^*$ into $\mathcal{C}$
21 sort the candidate set $\mathcal{C}$;
22 **for each** *candidate* $(e, e^*) \in \mathcal{C}$ *that* $e \in P, e^* \in P^*$ **do**
23 **if** $e \notin \mathbf{G}^-$ *and* $\phi_i^-(e, e^*)$ *returns* true **then**
24 $\mathbf{G}^- \leftarrow \mathbf{G}^- \cup P$;
25 **return** $\mathbf{G}^-$

sorts the candidate pairs and builds a graph (line 5–6). If e and e' belong to different connected components and satisfy a positive rule, we add the edge into the graph (lines 7–10). Then, it computes the connected components (line 11).

It then picks the pivot partition (line 12 for step 2).

Finally, it uses the signatures to discover mis-categorized entities that are not in the pivot partition (lines 13–24 for step 3). It generates the signature set of pivot partition P^* by computing the union of signatures of entities in P^* for each negative rule ϕ_i^-, denoted by $L_{\phi_i^-}(P^*)$ (lines 13–14). For each partition P, it computes its signature set $L_{\phi_i^-}(P)$ (lines 15–17). If the signature sets of P and P^* have no overlap, P is a mis-categorized partition (lines 18–19); otherwise, (e, e^*) that $e \in P, e^* \in P^*$ is a candidate (line 20). Then, it sorts candidates and checks whether each candidate satisfies a negative rule (lines 22–24). The mis-categorized entities are returned (line 25).

5.2 Signature generation

Signatures for Positive Rules Given a positive rule $\varphi^+ = \bigwedge_{A_i \in R} f_i(A_i) \geq \theta_i$, for each predicate $f_i(A_i) \geq \theta_i$, we generate a signature set $\mathtt{Sig}^i_{\varphi^+}(e)$ for each entity e such that if two entities, e and e', can satisfy this predicate, they must share some common signatures, *i.e.,* $\mathtt{Sig}^i_{\varphi^+}(e) \cap \mathtt{Sig}^i_{\varphi^+}(e') \neq \emptyset$. Conversely, if two entities do not share any common signature, they must be dissimilar. Note that a positive rule φ^+ is a conjunction of predicates $\bigwedge_{A_i \in R} f_i(A_i) \geq \theta_i$, we pick one signature *w.r.t.* each predicate as the signature *w.r.t.* φ^+ for an entity e, so $\mathtt{Sig}_{\varphi^+}(e) = \{(g^1, g^2, ..., g^{|\varphi^+|}) \mid g^1 \in \mathtt{Sig}^1_{\varphi^+}(e), g^2 \in \mathtt{Sig}^2_{\varphi^+}(e), \cdots, g^{|\varphi^+|} \in \mathtt{Sig}^{|\varphi^+|}_{\varphi^+}(e)\}$, where $\mathtt{Sig}^i_{\varphi^+}(e)$ is the signature *w.r.t.* the ith predicate.

Next we discuss how to generate the signatures $\mathtt{Sig}^i_{\varphi^+}(e)$. For set-based and character-based functions, we use existing techniques to generate signatures [41].

(i) Set-based We first set a global ordering on all the tokens (*e.g.,* document frequency). Then, for each value v in an attribute, we sort its tokens based on the ordering. For overlap similarity, given the overlap threshold θ, we select the first $|v| - \theta + 1$ tokens in v as the signatures. Obviously, if two values are similar (with at least θ common tokens), they must share a common signature. This can be extended to other metrics [41]. For example, consider e_1 in Fig. 1, and let the overlap threshold on `Authors` be 2. If we sort the authors based on their frequency, e_1[`Authors`] = *{Xu Chu, John Morcos, Mourad Ouzzani, Paolo Papotti, Ihab F. Ilyas, Nan Tang}*, and its signatures are the first $6 - 2 + 1 = 5$ authors.

(ii) Character-based The gram-based method [33] is widely used to support character-based similarity functions. Take edit distance as an example. We generate the q-grams (*i.e.,* substring with length q) of the value, sort the grams based on a global ordering, and select the first $q\theta + 1$ grams as its signatures. Evidently, if two values are similar (within edit distance of θ), they must have common signatures [33]. For example, given a value *Nan Tang* and the length of gram $q = 3$. We generate a set of 3-grams as *{Nan, an_, n_T, _Ta, Tan, ang}* where "_" denotes the white space. Suppose that the distance threshold $\theta = 1$, then we take the first $3 \times 1 + 1 = 4$ grams *{Nan, an_, n_T, _Ta}* as its signatures.

(iii) Ontology-based For ease of presentation, we use entity e and its mapping tree node n interchangeably. Given a node n, if another node n' is similar to n, we have $\frac{2|\mathtt{LCA}(n,n')|}{|n|+|\mathtt{LCA}(n,n')|} \geq \frac{2|\mathtt{LCA}(n,n')|}{|n|+|n'|} \geq \theta$ and $|\mathtt{LCA}(n, n')| \geq \frac{\theta|n|}{2-\theta}$. Let $\tau_n = \lceil \frac{\theta|n|}{2-\theta} \rceil$ and denote $\mathcal{A}_{\tau_n}$ by the ancestor of n at depth τ_n. We can take $\mathcal{A}_{\tau_n}$ as a signature of n. Similarly we can take $\mathcal{A}_{\tau_{n'}}$ as a signature of n'. If n and n' are similar, both $\mathcal{A}_{\tau_n}$ and $\mathcal{A}_{\tau_{n'}}$

are their LCA, and thus, we have $\mathcal{A}_{\tau_n} = \mathcal{A}_{\tau_{n'}}$ or one is an ancestor of the other, as proved in the following Lemma.

Lemma 1 *Given two nodes n and n′, if n and n′ are similar, $\mathcal{A}_{\tau_n} = \mathcal{A}_{\tau_{n'}}$, or $\mathcal{A}_{\tau_n}$ is an ancestor or a descendent of $\mathcal{A}_{\tau_{n'}}$.*

Proof First, if $|n| = |n'|$, $\tau_n = \tau_{n'} = \lceil \frac{\theta|n|}{2-\theta} \rceil$. We can prove that $\mathcal{A}_{\tau_n} = \mathcal{A}_{\tau_{n'}}$ by a contradiction. Suppose $\mathcal{A}_{\tau_n} \neq \mathcal{A}_{\tau_{n'}}$. The depth of their LCA must be smaller than τ_n. Thus, their similarity $\frac{2|\mathrm{LCA}(n,n')|}{|n|+|n'|} \leq \frac{2(\lceil \frac{\theta|n|}{2-\theta} \rceil - 1)}{2|n|} < \frac{\theta|n|}{(2-\theta)|n|} \leq \theta$. This is contradicted to the assumption that they are similar.

Second, if $|n| < |n'|$, we can prove that $\mathcal{A}_{\tau_n}$ is an ancestor of $\mathcal{A}_{\tau_{n'}}$ by a contradiction. Suppose $\mathcal{A}_{\tau_n}$ is not an ancestor of $\mathcal{A}_{\tau_{n'}}$. The depth of their LCA must be smaller than τ_n. Thus, their similarity is smaller than θ. This is also contradicted to the assumption that they are similar.

Third, if $|n| > |n'|$, we prove $\mathcal{A}_{\tau_n}$ is a descendent of $\mathcal{A}_{\tau_{n'}}$ by a contradiction. Suppose $\mathcal{A}_{\tau_n}$ is not a descendant of $\mathcal{A}_{\tau_{n'}}$. The depth of their LCA must be smaller than $\tau_{n'}$. Thus, their similarity is smaller than θ. So there is a contradiction. □

It is very efficient to check whether $\mathcal{A}_{\tau_n} = \mathcal{A}_{\tau_{n'}}$, but it is not easy to check the ancestor-descendant relationship. We propose to use node signatures to address this issue.

Node Signature Let $\tau_{n,n'} = \min(\tau_n, \tau_{n'})$ denote the smaller depth of nodes $\mathcal{A}_{\tau_n}, \mathcal{A}_{\tau_{n'}}$. If we take the nodes at depth $\tau_{n,n'}$ as a signature of n and n', they must be the same node when n is similar to n'. To generate the signature for all nodes, we set τ_{min} as the minimum depth of their signatures. Then for each node n, we select $\mathcal{A}_{\tau_{min}}$ as its node signature and use this signature to find similar entities.

Example 6 (*Node Signature.*) Consider three nodes *Computer Science*, *Database*, and *VLDB* in Fig. 4. Suppose $\theta = 0.75$. Their τ_n are $\lceil \frac{0.75*2}{2-0.75} \rceil = 2$, $\lceil \frac{0.75*3}{2-0.75} \rceil = 2$, and $\lceil \frac{4*0.75}{2-0.75} \rceil = 3$. Thus, their signatures are *Computer Science*, *Computer Science*, and *Database*, respectively. Their node signatures are all *Computer Science*.

Lemma 2 *Given two nodes n and n′, if n and n′ are similar, their node signatures must be the same node.*

Proof Suppose their node signatures are not the same node. The depth of their LCA must be smaller than $\tau_{n,n'}$. Thus, their similarity is smaller than $\frac{2\theta\tau_{n,n'}}{(2-\theta)2\tau_{n,n'}} \leq \theta$. □

Example 7 (*Signatures for Positive Rules.*) Consider e_1 in Fig. 1 and the positive rules φ_1^+, φ_2^+ below Example 4. φ_1^+ has only one predicate $f_{ov}(\mathtt{Authors}) \geq 2$, and the signatures of e_1 *w.r.t.* φ_1^+ are $\mathtt{Sig}_{\varphi_1^+}(e_1) = \{$*Xu Chu, John Morcos, Mourad Ouzzani, Paolo Papotti, Ihab F. Ilyas*$\}$. φ_2^+ has two predicates $p_1 : f_{ov}(\mathtt{Authors}) \geq 1$ and $p_2 : f_{on}(\mathtt{Venue}) \geq 0.75$. As for p_1, the signatures are all authors. As for p_2, the node signature of *SIGMOD* is *Database*. Thus, the signature set of e_1 *w.r.t.* φ_2^+ is $\mathtt{Sig}_{\varphi_2^+}(e_1) = \{$(*Xu Chu, Database*), (*John Morcos, Database*), (*Mourad Ouzzani, Database*), (*Paolo Papotti, Database*), (*Ihab F. Ilyas, Database*), (*Nan Tang, Database*)$\}$.

Signatures for Negative Rules We generate the signature set for an entity *w.r.t.* a negative rule similar to the positive rule. The difference is that the thresholds, θ_i and σ_i, are different. If two entities e and e' have no common signature in every predicate, they must satisfy the negative rule ϕ^-.

5.3 Filter and verification for positive rules

Filter Strategy for Positive Rules For each positive rule φ_i^+, we build an inverted index of the signatures of all entities, where each inverted list maintains a mapping from a signature to a list of entities that contain the signature—$L_{\varphi_i^+}(sig)$ contains the set of entities with sig as a signature. Then, every entity pair (e, e') on $L_{\varphi_i^+}(sig)$ is a candidate pair. Two entities that are not on the same inverted list cannot be similar—they do not share any common signature.

Verification Acceleration for Positive Rules It is expensive to verify whether an entity pair satisfies a positive rule (see Sect. 4). To address this issue, we can use transitivity to avoid verifying unnecessary pairs. Given a candidate pair (e, e'), we check whether they are in the same connected component with a constant time complexity[3]. If so, we do not need to verify the pair.

Verification Order of Candidate Pairs The order of verifying the candidate pairs will affect the performance. If we know that (e, e') match and (e', e'') match, we can infer that (e, e'') match. Naturally, we can infer the answer of (e, e'') based on those of (e, e') and (e', e''). Wang et al. [61] proved that it is optimal to ask the candidate pairs sorted by the similar probabilities in descending order, and they used the similarity to approximate the probability. The similarity, however, is expensive to compute. Thus, we propose a new method to sort the entity pairs taking both computational cost and similar probability into consideration.

Benefit Order We compute the *benefit* of verifying a candidate pair. Suppose that each candidate pair (e, e') has a verification cost $\mathcal{C}(e, e')$ and similar probability $\mathcal{P}(e, e')$. We compute the benefit $\mathcal{B}(e, e') = \frac{\mathcal{P}(e,e')}{\mathcal{C}(e,e')}$. Obviously, the greater the benefit of a candidate pair is, the better it is to verify with low cost. Thus, we compute the benefit for each pair and verify the candidates sorted by the benefits in a descending order.

[3] We assign each entity a partition ID and utilize a union-find data structure. If e and e' are verified that they satisfy a positive rule, we update their partition ID to the same ID. Assume the partition ID of e is i and that of e' is j, and $i < j$, we change the partition ID of e' to i.

Next we discuss how to compute the verification cost and the similar probability.

Verification Cost The cost of verifying whether (e, e') satisfies a positive rule is the sum of the cost of verifying whether (e, e') can satisfy each predicate. The well-known verification algorithm for edit distance (character-based similarity) is the dynamic-programming algorithm with the time cost of $\mathcal{O}(\theta \min(|e|, |e'|))$, where θ is the similarity threshold. The verification algorithm for Jaccard (set-based similarity) is $\mathcal{O}(|e|+|e'|)$. The verification cost for ontology-based similarity is $O(|d_e| + |d_{e'}|)$ where d_e is the depth of the node in the tree that entity e matches.

Similar Probability The probability of whether (e, e') can satisfy a positive rule is hard to compute. To address this issue, we can use their shared signatures to approximate the probability, which is the ratio of the number of shared signatures to their average signature number.

Example 8 (*Efficient Checking for Positive Rules.*) Given entities in Fig. 1 and positive rules φ_1^+, φ_2^+ below Example 4. We generate two candidates $\{(e_1, e_3), (e_2, e_5)\}$ for φ_1^+ and three candidates $\{(e_1, e_2), (e_1, e_3), (e_2, e_3)\}$ for φ_2^+.

Then, we decide the order of verification. The cost of verifying (e_1, e_3) *w.r.t.* φ_1^+ is $\mathcal{C}(e_1, e_3) = 10$, and the similarity probability is $\mathcal{P}(e_1, e_3) = \frac{1}{4} = 0.25$. Thus, the benefit of verifying (e_1, e_3) is 0.025. We compute the benefit for other candidates, and sort them as $\{(e_2, e_5)_{\varphi_1^+}, (e_1, e_3)_{\varphi_1^+}, (e_1, e_3)_{\varphi_2^+}, (e_2, e_3)_{\varphi_2^+}, (e_1, e_2)_{\varphi_2^+}\}$. In this order, we verify (e_2, e_5), (e_1, e_3) for φ_1^+ and (e_2, e_3) for φ_2^+; then, e_1, e_2, e_3, e_5 are in the same partition.

5.4 Filter and verification for negative rules

Filter Strategy for Negative Rules After specifying the pivot partition P^*, we utilize the signatures to detect mis-categorized entities from another partition $P \in \mathbf{P} \setminus \{P^*\}$. For each negative rule ϕ_i^-, it generates the signature set $L_{\phi_i^-}(P^*)$ of P^*, and the signature set $L_{\phi_i^-}(P)$ of P. If $L_{\phi_i^-}(P^*) \cap L_{\phi_i^-}(P) = \emptyset$, P is a mis-categorized partition; otherwise, we add $(e \in P, e^* \in P^*)$ into the candidate set.

Verification Acceleration for Negative Rules It is also inefficient to verify every candidate (e, e^*). Note that once we find a dissimilar pair, P is a mis-categorized partition. In other words, if $\phi^-(e, e^*)$ returns true, there is no need to verify other entities from P. Thus, we want to first verify the pair with the smallest probability in order to prune a partition. In addition, the pair (e, e^*) with smaller verification cost $\mathcal{C}(e, e^*)$ should be verified first. So, the benefit should be defined as $\frac{1}{\mathcal{C}(e,e^*)\mathcal{P}(e,e^*)}$. Then, we compute the benefit of each entity pair and verify the pairs sorted by the benefit in a descending order.

Example 9 (*Efficient Checking for Negative Rules.*) Following the above example, partition $P_1 : \{e_1, e_2, e_3, e_5\}$ is regarded as the pivot partition P^*. Consider the negative rules ϕ_1^-, ϕ_2^- below Example 4 and other two partitions $P_2 : \{e_4\}$ and $P_3 : \{e_6\}$. We can conclude that P_2 is mis-categorized partition based on ϕ_1^-, because $L_{\phi_1^-}(P_2) = \{$*Yunqing Xia, NJ Tang, Amir Hussain, Erik Cambria*$\}$ which cannot match any signature of the entities in P^*. In the same way, we can detect that P_3 is also a mis-categorized partition based on ϕ_2^- since $L_{\phi_2^-}(P^*) \cap L_{\phi_2^-}(P_3) = \emptyset$.

6 Rule generation

In practice, it is hard for human to fine tune the parameters of all rules, such as similarity functions and their associated thresholds. Hence, we discuss how to generate positive/negative rules from examples. A positive/negative example is a pair of entities that are/are not in the same category. Note that finding "good examples" is a known challenging problem in training EM models [52], where the good examples normally refer to the non-matched pairs that are likely to be confused as matches. Fortunately, in our case, it is much easier to find good examples, since we only find examples within groups and mis-categorized entities can be paired with any other correctly categorized entities as good examples. Another challenge, which is not well addressed before, is how to derive rules only from examples.

6.1 Positive rule generation

Positive Rule Generation Given a multi-valued relation $R(A_1, ..., A_m)$, a set of positive examples $\mathbf{S}^+$, a set of negative examples $\mathbf{S}^-$, and a library of similarity functions F, the problem is to find a set Σ^+ of positive rules to maximize a pre-defined objective function $\mathbb{F}(\Sigma^+, \mathbf{S}^+, \mathbf{S}^-)$ and $|\Sigma^+|$ is minimum among all rule sets that maximize $\mathbb{F}$.

Objective Function Consider a set of positive rules Σ^+. Given a rule $\varphi^+ \in \Sigma^+$, let $\mathcal{E}_{\varphi^+}$ be the set of entity pairs satisfying φ^+ and $\mathcal{E}_{\Sigma^+} = \bigcup_{\varphi^+ \in \Sigma^+} \mathcal{E}_{\varphi^+}$. To evaluate the quality of Σ^+, we focus on a general case of objective function $\mathbb{F}(\Sigma^+, \mathbf{S}^+, \mathbf{S}^-)$: The larger $|\mathcal{E}_{\Sigma^+} \cap \mathbf{S}^+|$, the larger $\mathbb{F}(\Sigma^+, \mathbf{S}^+, \mathbf{S}^-)$, and the smaller $|\mathcal{E}_{\Sigma^+} \cap \mathbf{S}^-|$, the larger $\mathbb{F}(\Sigma^+, \mathbf{S}^+, \mathbf{S}^-)$. Many functions belong to this general case, *e.g.,* $\mathbb{F}(\Sigma^+, \mathbf{S}^+, \mathbf{S}^-) = |\mathcal{E}_{\Sigma^+} \cap \mathbf{S}^+| \setminus |\mathcal{E}_{\Sigma^+} \cap \mathbf{S}^-|$.

From Infinite to Finite A naïve method of rule generation is to enumerate all possible rules and then select some of them to maximize the objective function. Then, a problem arises: *Whether the number of all possible positive rules is finite?* The answer is yes. Each rule contains at most m predicates, where m is the number of attributes. For each predicate, there are $|F|$ similarity functions that can be chosen. Although it

seems to have infinite similarity thresholds, we can pick finite thresholds which can also maximize the objective function. Next we use an example to illustrate the idea. Consider a rule φ^+ with a single predicate $f(A) \geq \theta_1$ and another rule φ'^+ with a single predicate $f(A) \geq \theta_2$ by relaxing θ_1 to θ_2 ($\theta_2 < \theta_1$). Obviously, $\mathcal{E}_{\varphi^+} \subseteq \mathcal{E}_{\varphi'^+}$. If there is no positive example in $\mathcal{E}_{\varphi'^+} \setminus \mathcal{E}_{\varphi^+}$, then $\mathbb{F}(\varphi^+, \mathbf{S}^+, \mathbf{S}^-) \geq \mathbb{F}(\varphi'^+, \mathbf{S}^+, \mathbf{S}^-)$, and we only need to keep θ_1. Then, a question is *which thresholds affect the number of satisfied positive examples?* We compute the similarity for all positive examples on attribute A, and only these similarities can affect the number of satisfied positive examples. Thus, the number of possible thresholds for an attribute is only $|F||\mathbf{S}^+|$.

Theorem 1 *Consider a set of positive examples $\mathbf{S}^+$, a set of negative examples $\mathbf{S}^-$, an objective function $\mathbb{F}(\Sigma^+, \mathbf{S}^+, \mathbf{S}^-)$, and two sets of positive rule Σ_1^+, Σ_2^+. Suppose Σ_2^+ is transformed from Σ_1^+ that only replacing $f(A) \geq \theta_1$ in Σ_1^+ by $f(A) \geq \theta_2$. If $\theta_2 < \theta_1$ and there is no positive example in $\mathcal{E}_{\Sigma_2^+} \setminus \mathcal{E}_{\Sigma_1^+}$, $\mathbb{F}(\Sigma_2^+, P, N) \leq \mathbb{F}(\Sigma_1^+, P, N)$.*

The correctness of theorem has been proved in [62]. Based on this idea, we can generate a finite set of predicates for each attribute. Let $\mathcal{C}_p^+(A)$ denote all candidate predicates of attribute A. For each similarity function $f \in F$ and each pair of positive examples (e, e') from $\mathbf{S}^+$, we compute θ based on $f(A)$ for e and e', and add $f(A) \geq \theta$ into $\mathcal{C}_p^+(A)$. For all other thresholds, we can safely prune them since they cannot provide a higher objective value.

Example 10 (*Candidate Predicates.*) Consider the entities in Fig. 1; we have positive/negative examples as follows: $\mathbf{S}^+ = \{(e_1, e_2), (e_1, e_3), (e_1, e_5), (e_2, e_3), (e_2, e_5), (e_3, e_5)\}$; $\mathbf{S}^- = \{(e_1, e_4), (e_1, e_6), (e_2, e_4), (e_2, e_6), (e_3, e_4), (e_3, e_6), (e_4, e_5), (e_5, e_6)\}$. Based on Theorem 1, we have the following predicates. For ease of presentation, we use specific similarity function for each attribute: $\mathcal{C}_p^+(\texttt{Title}) = \{f_j(\texttt{Title}) \geq 0.3,\ f_j(\texttt{Title}) \geq 0.07,\ f_j(\texttt{Title}) \geq 0.05\}$; $\mathcal{C}_p^+(\texttt{Authors}) = \{f_{ov}(\texttt{Authors}) \geq 2,\ f_{ov}(\texttt{Authors}) \geq 1\}$; $\mathcal{C}_p^+(\texttt{Venue}) = \{f_{on}(\texttt{Venue}) \geq 0.75,\ f_{on}(\texttt{Venue}) \geq 0.5\}$;

Based on the finite set of possible predicates on each attribute A_i, we can generate all possible rules by selecting 0-1 predicate from each set $\mathcal{C}_p^+(A_i)$ for $1 \leq i \leq m$. Let Σ_a^+ denote the set of all possible rules. Then from Σ_a^+, we aim to find a subset to maximize the objective function. We prove that this problem is NP-hard by a reduction from the maximum coverage problem.

Theorem 2 *The rule generation problem is NP-hard.*

Proof We prove the rule generation problem is NP-hard by a reduction from the maximum coverage problem [39]. Given a universal set $U = \{o_1, o_2, ..., o_n\}$ of n elements, a collection $C = \{S_1, S_2, ..., S_p\}$ where $S_i \subseteq U (i \in [1, p])$, and an integer k, the maximum coverage problem is to select k sets from C such that their union has the maximum cardinality. Then, we reduce the maximum coverage problem to the rule generation problem.

- Construct a set of positive examples $\mathbf{S}^+ = U = \{o_1, o_2, ..., o_n\}$ and a set of negative examples $\mathbf{S}^- = \{o_{n+1}, o_{n+2}, ..., o_{2n}\}$ whose elements are not from U.
- Given the multi-valued relation $R(A_1, ..., A_m)$ and a library of similarity functions F, generate a finite set of predicates $\mathcal{C}_p^+(A_i)$ for each attribute A_i.
- Generate all possible rules Σ_a^+ by selecting 0-1 predicate from each set $\mathcal{C}_p^+(A_i)$ for $1 \leq i \leq m$.
- Define the objective function as $\mathbb{F}(\Sigma^+, \mathbf{S}^+, \mathbf{S}^-) = |\mathcal{E}_{\Sigma^+} \cap \mathbf{S}^+|\ s.t. |\mathcal{E}_{\Sigma^+} \cap \mathbf{S}^-| = 0$. This objective function means that we need maximize the positive examples Σ^+ can recognize and ensure that no false negative is exist. Therefore, Σ_c^+ contains all rules from Σ_a^+ which satisfy $\mathcal{E}_{\varphi^+} \cap \mathbf{S}^+ = \mathcal{E}_{\varphi^+}$.

Then, our rule generation problem can be stated as a maximum coverage problem. The universal set U contains all positive examples from $\mathbf{S}^+$, and the collection $C = \{\mathcal{E}_{\varphi_1^+}, \mathcal{E}_{\varphi_2^+}, ..., \mathcal{E}_{\varphi_{|\Sigma_c^+|}^+}\}$ where $\mathcal{E}_{\varphi_i^+} = \mathcal{E}_{\varphi_i^+} \cap \mathbf{S}^+ \subseteq U$. As the size of final rule set should be minimum among all rule sets that maximize $\mathbb{F}$, we change k from one to $|\Sigma_c^+|$ and select k sets from C in turn such that their union has the maximum cardinality until $\mathbb{F}$ can not be improved any more. For certain $k \in [1, |\Sigma_c^+|]$, the problem is NP-hard. Thus, the rule generation problem is NP-hard. □

6.2 An enumeration-based algorithm

Any subset Σ_c^+ of Σ_a^+ is a candidate set of positive rules. We enumerate each subset Σ_c^+ in the order of increasing size, compute its objective-function value, and select the one with the maximal value. Obviously, there are large numbers of candidates, *i.e.,* $2^{|\Sigma_a^+|} = \mathcal{O}(2^{|F|^m |\mathbf{S}^+|^m})$, where m is the number of attributes, $|F|$ is the number of similarity functions, and $|\mathbf{S}^+|$ is the number of positive examples. Clearly, this enumeration-based method is rather expensive.

Example 11 (*Enumeration-based Algorithm.*) We enumerate 35 positive rules based on $\mathcal{C}_p^+(\texttt{Title})$, $\mathcal{C}_p^+(\texttt{Authors})$ and $\mathcal{C}_p^+(\texttt{Venue})$ such as:

φ^+: $f_j(\texttt{Title}) \geq 0.31 \wedge f_{ov}(\texttt{Authors}) \geq 2 \wedge f_{on}(\texttt{Venue}) \geq 0.75$
$\varphi^{+\prime}$: $f_j(\texttt{Title}) \geq 0.31 \wedge f_{ov}(\texttt{Authors}) \geq 2 \wedge f_{on}(\texttt{Venue}) \geq 0.5$
$\varphi^{+\prime\prime}$: $f_j(\texttt{Title}) \geq 0.31 \wedge f_{ov}(\texttt{Authors}) \geq 1$ ...

Then, we enumerate all combinations of these positive rules to construct candidate sets of rules. Suppose that Σ_c^+ contains the above three rules and the objective function is $\mathbb{F}(\Sigma^+, \mathbf{S}^+, \mathbf{S}^-) = |\mathcal{E}_{\Sigma^+} \cap \mathbf{S}^+| \setminus |\mathcal{E}_{\Sigma^+} \cap \mathbf{S}^-|$. As $\mathcal{E}_{\Sigma_c^+} = \{(e_1, e_3)\}$, $\mathbb{F}(\Sigma_c^+, \mathbf{S}^+, \mathbf{S}^-) = 1$. For other candidates, we also compute their objective-function value and then select the one with the maximal value.

6.3 A greedy algorithm

Due to the high computational complexity of the enumeration algorithm, we propose a greedy algorithm to find a near-optimal set of rules $\hat{\Sigma}^+$. Initially $\hat{\Sigma}^+ = \emptyset$, and we greedily choose the currently best rule until the objective-function value cannot be improved. To generate the best rule φ^+, we greedily select the best predicate as follows.

Given a set of positive examples $\mathbf{S}^+$, a set of negative examples $\mathbf{S}^-$, and sets of candidate predicate $\mathcal{C}_p^+(A_i)$ for each attribute A_i, we first choose the best predicate to initialize the rule $\varphi^+ = f(A) \geq \theta$ which has the maximal objective-function value $\mathbb{F}(\varphi^+, \mathbf{S}^+, \mathbf{S}^-)$. Meanwhile, we update the example set $\mathbf{S}^+, \mathbf{S}^-$ to $\mathbf{S}'^+, \mathbf{S}'^-$ by removing the examples that cannot satisfy φ^+.

Afterward, we pick another predicate p' to join with φ^+ which can prune the negative examples that have satisfied φ^+ as many as possible (smaller $|\mathcal{E}_{p'} \cap \mathbf{S}'^-|$) and keep the positive examples as many as possible (larger $|\mathcal{E}_{p'} \cap \mathbf{S}'^+|$). Thus, we select the one with the maximal objective-function value $\mathbb{F}(p', \mathbf{S}'^+, \mathbf{S}'^-)$ and update the positive rule as $\varphi^+ = f(A) \geq \theta \wedge f'(A') \geq \theta'$. Another predicate will be chosen until $\mathbb{F}(\varphi^+, \mathbf{S}^+, \mathbf{S}^-)$ cannot be improved and we get φ^+.

We generate a set of rules $\hat{\Sigma}^+$ on the basis of the steps stated above. After φ^+ is added into $\hat{\Sigma}^+$, we should update the example set $\mathbf{S}^+, \mathbf{S}^-$ to $\mathbf{S}''^+, \mathbf{S}''^-$ by removing the examples that satisfy φ^+ and make use of $\mathbf{S}''^+, \mathbf{S}''^-$ to measure the performance of other positive rules. The algorithm terminates when $\mathbb{F}(\Sigma^+, \mathbf{S}^+, \mathbf{S}^-)$ cannot be improved.

Example 12 (*Greedy Algorithm.*) Consider the entities in Fig. 1. The positive examples, negative examples, all predicates and the objective function are the same with the above example.

At first, we compute the score of the objective function for all predicates. Let p_1 denote the predicate $f_{ov}(\texttt{Authors}) \geq 2$. As $\mathcal{E}_{p_1} = \{(e_1, e_3), (e_2, e_5)\}$, $\mathbb{F}(p_1, \mathbf{S}^+, \mathbf{S}^-) = 2 - 0 = 2$. Actually, it is the predicate whose objective value is maximum. So we first initialize the rule $\varphi_1^+ = p_1$ and update the example set as $\mathbf{S}'^+ = \{(e_1, e_3), (e_2, e_5)\}$ and $\mathbf{S}'^- = \emptyset$. Since the set of negative examples is empty, we generate one positive rule φ_1^+: $f_{ov}(\texttt{Authors}) \geq 2$.

Next, we update the example set by removing the examples that satisfy φ_1^+. Then, $\mathbf{S}''^+ = \{(e_1, e_2), (e_1, e_5), (e_2, e_3), (e_3, e_5)\}$ and $\mathbf{S}''^- = \{(e_1, e_4), (e_1, e_6), (e_2, e_4), (e_2, e_6), (e_3, e_4), (e_3, e_6), (e_4, e_5), (e_5, e_6)\}$. We re-evaluate the values of objective function for all predicates, and generate another positive rule φ_2^+: $f_{ov}(\texttt{Authors}) \geq 1 \bigwedge f_{on}(\texttt{Venue}) \geq 0.75$.

6.4 Negative rule generation

The goal of negative rules is to identify mis-categorized partitions. We want to select the negative rules Σ^- to cover more negative examples (larger $|\mathcal{E}_{\Sigma^-} \cap \mathbf{S}^-|$) and less positive examples (smaller $|\mathcal{E}_{\Sigma^-} \cap \mathbf{S}^+|$). The objective function should be $\mathbb{F}(\Sigma^-, \mathbf{S}^+, \mathbf{S}^-) = |\mathcal{E}_{\Sigma^-} \cap \mathbf{S}^-| \setminus |\mathcal{E}_{\Sigma^-} \cap \mathbf{S}^+|$, or other functions that can measure the target. Theorem 1 can be utilized to generate a finite set of predicates for each attribute. Let $\mathcal{C}_p^-(A)$ denote all candidate predicates of attribute A. For each similarity function $f \in F$ and each pair of negative examples (e, e') from $\mathbf{S}^-$, we compute $\sigma = f(e[A], e'[A])$ and add $f(A) \leq \sigma$ into $\mathcal{C}_p^-(A)$. Recall that in each step of our greedy algorithm, we generate a negative rule ϕ^- with the maximal score based on $\mathbb{F}(\phi^-, \mathbf{S}^+, \mathbf{S}^-)$. Thus, these rules can be applied in the order of generation.

7 Resolving inconsistent rules

Although not desired, conflict may exist between positive and negative rules. In this section, we first define the *consistency* of positive and negative rules based on the *transitivity* assumption for positive rules and discuss how to check the consistency of a set of rules (Sect. 7.1). However, the transitivity assumption is sometimes violated by real-world cases. Hence, we propose soft positive/negative rules that weaken the transitivity assumption, which can better capture real-world cases (Sect. 7.2). Finally, we present two novel strategies to discover mis-categorized entities by using a set of soft rules (Sect. 7.3).

7.1 Checking consistency

Rule Consistency Given a set Σ^+ of positive rules, a set Σ^- of negative rules, and a group $\mathbf{G}$ of entities, we say that Σ^+ and Σ^- are *inconsistent w.r.t.* $\mathbf{G}$, *if* there exist two entities e and e' from $\mathbf{G}$, such that

1. there have a positive rule $\varphi^+ \in \Sigma^+$ and a negative rule $\phi^- \in \Sigma^-$, where both $\varphi^+(e, e')$ and $\phi^-(e, e')$ return true; or
2. $\varphi^+(e, e')$ returns false for any $\varphi^+(e, e') \in \Sigma^+$ and there exists a negative rule $\phi^-(e, e')$ that returns true, but e and e' satisfy transitivity: There exists another entity e'' such that both (e, e'') and (e', e'') match.

7.1.1 A brute-force solution

We can first compute disjoint partitions $\mathbf{P}$ of $\mathbf{G}$ by using positive rules Σ^+ and applying transitivity (see more details in Sect. 4, Step 1). Given a partition $P \in \mathbf{P}$, we enumerate every entity pair (e, e') in P, and check whether $\phi^-(e, e')$ returns true for each negative rule $\phi^- \in \Sigma^-$. If any $\phi^-(e, e')$ returns true, Σ^+ and Σ^- are inconsistent, by definition.

Example 13 (*Checking Consistency.*) Consider the positive and negative rules in Example 2 and suppose that there is another positive rule:

$\varphi_3^+ : f_{on}(\texttt{Title}) \geq 0.5 \wedge f_{on}(\texttt{Venue}) \geq 0.75$

which says that two entities are similar, if they have similar topic and their Venue are in the same field. We then check whether $\Sigma^+ = \{\varphi_1^+, \varphi_2^+, \varphi_3^+\}$ and $\Sigma^- = \{\phi_1^-, \phi_2^-\}$ are consistent within the entities e_1 to e_6 in Example 1.
The sorted candidate pairs which may satisfy a positive rule are $\mathcal{C} = \{(e_2, e_5)_{\varphi_1^+}, (e_1, e_3)_{\varphi_1^+}, (e_2, e_4)_{\varphi_3^+}, (e_1, e_3)_{\varphi_3^+}, (e_1, e_3)_{\varphi_2^+}, (e_2, e_3)_{\varphi_2^+}, (e_1, e_2)_{\varphi_2^+}\}$. After we verify the first two candidates, there are three connected components: $\{e_1, e_3\}$, $\{e_4\}$ and $\{e_2, e_5\}$. Since the third candidate (e_2, e_4) can satisfy φ_3^+, before merging these two connected components $\{e_4\}$ and $\{e_2, e_5\}$, we should check whether (e_2, e_4) and (e_4, e_5) will satisfy a negative rule. We first test (e_4, e_5) which has lower verification cost, and $\phi_1^-(e_4, e_5)$ returns true. Thus, Σ^+ is inconsistent with Σ^-.

7.1.2 A signature-based fast solution

A smarter and more efficient approach is to *push the consistency check down* to the computation of disjoint partitions $\mathbf{P}$, by using signatures introduced in Sect. 5. Recall that in the "*filter*" step, we generate signatures for each entity *w.r.t.* each positive rule and take the pairs of entities that share common signatures as candidate pairs $\mathcal{C}$ (lines 1–4 of Algorithm 2). Then in the "*verification*" step, if a candidate pair $(e, e') \in \mathcal{C}$ is evaluated to be true and these two entities belong to different connected components (lines 7–9 of Algorithm 2), before putting e and e' into the same partition, we first enumerate every entity pair across these two connected components, and verify whether they can satisfy a negative rule. The pair with the lower similar probability and verification cost will be checked first. Once we find a dissimilar pair, the rules are inconsistent.

A potential performance bottleneck is that we can *only* draw the conclusion for rule consistency when all entity pairs which are grouped together cannot satisfy negative rules. Apparently, this is time consuming when there is a large amount of entities. To cope with this, one approximate yet effective strategy is to leverage the positive and negative examples that are used for rule generation. Alternatively, after computing disjoint partitions $\mathbf{P}$, we only check whether the entity pairs with lower similar probability would satisfy the negative rules. If the rules are consistent, we can utilize the framework in Sect. 4 to discover mis-categorized entities. Otherwise, we will describe how to handle harder cases in the following of this section.

7.2 Soft positive and negative rules

As mentioned earlier, the transitivity assumption may be violated by real-world cases. In other words, this assumption puts a large burden of rule generation algorithms to precisely capture real-world scenarios. Hence, we propose soft positive rules and soft negative rules to deal with this case.

Soft Positive Rules A *soft positive rule* $\Psi^+(e, e')$ is formalized as $[\varphi^+(e, e'), w_{\varphi^+}]$, where $\varphi^+(e, e')$ is a positive rule and w_{φ^+} is a rule weight in $[0, 1]$. $\Psi^+(e, e')$ is evaluated to be *true* if $\varphi^+(e, e')$ returns *true*, **indicating the two entities e and e' are "similar" and are likely to be categorized together**. Otherwise, $\Psi^+(e, e')$ returns *false*, if $\varphi^+(e, e')$ returns *false*, indicating we *do not know* whether they should be in the same category.

Soft Negative Rules A *soft negative rule* $\Phi^-(e, e')$ is defined similarly: $\Phi^-(e, e') = [\phi^-(e, e'), w_{\phi^-}]$, where $\phi^-(e, e')$ is a negative rule and w_{ϕ^-} is a rule weight in $[-1, 0]$. $\Phi^-(e, e')$ is evaluated to be *true* if $\phi^-(e, e')$ returns *true*, **indicating e and e' are "dissimilar" and possibly should not be categorized together**. Otherwise, $\Phi^-(e, e')$ returns *false*, if $\phi^-(e, e')$ returns *false*, indicating we *do not know* whether they should be in different categories.

Rule Weights The weight of a rule measures how accurate the rule is to be used to group entities. It can be obtained with the positive and negative examples for rule generation. We first take the positive rule as an example here. Concretely, given a set of positive example $\mathbf{S}^+$, a set of negative examples $\mathbf{S}^-$ and a positive rule φ^+ learned from $\mathbf{S}^+$ and $\mathbf{S}^-$, let $\mathcal{E}_{\varphi^+}$ be the set of entity pairs satisfying φ^+. Then, the precision of rule φ^+ is $\frac{\mathcal{E}_{\varphi^+} \cap \mathbf{S}^+}{\mathcal{E}_{\varphi^+}}$, and the recall is $\frac{\mathcal{E}_{\varphi^+} \cap \mathbf{S}^+}{\mathbf{S}^+}$. F-measure is the harmonic mean of precision and recall, which can be regarded as the weight of corresponding rule. For negative rule, it only needs to take the opposite value of F-measure as the rule weight.

Example 14 (*Soft Rules.*) Here, we have three soft positive rules and two soft negative rules.

$\Psi_1^+ : [f_{ov}(\texttt{Authors}) \geq 2, 0.5]$
$\Psi_2^+ : [f_{ov}(\texttt{Authors}) \geq 1 \wedge f_{on}(\texttt{Venue}) \geq 0.75, 0.67]$
$\Psi_3^+ : [f_{on}(\texttt{Title}) \geq 0.5 \wedge f_{on}(\texttt{Venue}) \geq 0.75, 0.25]$
$\Phi_1^- : [f_{ov}(\texttt{Authors}) = 0, -0.2]$
$\Phi_2^- : [f_{ov}(\texttt{Authors}) \leq 1 \wedge f_{on}(\texttt{Venue}) \leq 0.25, -0.67]$

Given the positive examples $\mathbf{S}^+$ and the negative examples $\mathbf{S}^-$ in Example 10, the rule weight in Ψ_1^+ will be 0.5. This is because $\mathcal{E}_{\varphi_1^+} = \{(e_1, e_3), (e_2, e_5)\}$, the precision of φ_1^+ is 1 and the recall is $\frac{1}{3}$. Thus, the F-measure of φ_1^+ is 0.5 which can also be assigned to the weight of φ_1^+.

7.3 Discovering mis-categorized entities by soft rules

This section discusses how to utilize soft rules to identify pivot partition and discover mis-categorized entities. Note that the previous framework cannot work here, because mis-categorized entities may be grouped into the pivot partition if we only consider soft positive rules when computing disjoint partitions, *e.g.,* entity e_4 will be grouped into the pivot partition because of φ_3^+. Thus, we propose two novel strategies to discover mis-categorized entities by soft rules: the correlation method (Sect. 7.3.1) and the partition split method (Sect. 7.3.2).

7.3.1 Correlation partition

As we state above, only considering soft positive rules is not enough to group entities. Here, we utilize the correlation clustering [5] to compute disjoint partitions. That is, we apply the soft positive and negative rules at the same time, and find the partitions which can make more soft positive/negative rules be satisfied.

Given a group of entities $\mathbf{G} = \{e_1, ..., e_n\}$, a set of soft positive rules Σ_s^+ and a set of soft negative rules Σ_s^-, we first build a weighted undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ for all entities in $\mathbf{G}$. Each vertex $v \in \mathcal{V}$ is an entity, and if two entities satisfy a positive rule in Σ_s^+ or a negative rule in Σ_s^-, there is an edge between them. Each edge is associated with a weight, which is the sum of weights of the soft rules these two entities can satisfy. Specifically, if two entities either satisfy positive rules only or the total weight of positive rules is larger than the absolute value of the total weight of negative rules they can satisfy, then edge (u, v) has a positive weight w_{uv}^+, indicating the two entities are likely to be grouped together. Otherwise, edge (u, v) has a negative weight w_{uv}^-, indicating the two entities possibly should not be grouped together. Then, the goal of correlation clustering is to find a clustering either maximizes agreements (the sum of positive weights inside clusters plus the absolute value of the sum of negative weights between clusters) or minimizes disagreements (the absolute value of the sum of negative weights inside clusters plus the sum of positive weights between clusters).

It has been proved that the problem of minimizing disagreements, or equivalently, maximizing agreements, is NP-hard by a reduction from the multiway cut problem [5,22]. The majority of approximate algorithms can only apply to the complete graph. Thus, here we adopt [22] which is an $\mathcal{O}(\log n)-$approximation algorithm of correlation clustering in general weighted graph. The algorithm first solves an integer program and the results are interpreted as distances between vertices: $d_{uv} = 0$ if u and v are in the same cluster, and $d_{uv} = 1$ if u and v are in different clusters. Here, d_{uv} denotes the distance between u and v, and they satisfy the triangle inequality constraints. Concretely, let $\mathcal{E}^+$ and $\mathcal{E}^-$ be the set of edges with positive and negative weights, respectively, then we can relax the integrality constraints and obtain the following linear program:

$$\begin{aligned} &\min \sum_{(u,v)\in\mathcal{E}^+} w_{uv}^+ d_{uv} + \sum_{(u,v)\in\mathcal{E}^-} |w_{uv}^-|(1 - d_{uv}) \\ &s.t.\ \ d_{uv} \in [0, 1], d_{uv} + d_{vw} \geq d_{uw} \end{aligned}$$

After finding the valid assignment of d_{uv}, the algorithm then uses region-growing techniques to group close vertices and thus round the fractional variables. Before introducing the techniques, we first give some related concepts. A region $R(u, r)$ consists of all vertices around u such that their distances are no larger than r. The cut of region $R(u, r)$, denoted by $cut(R)$, is the sum of weights of positive edges with exactly one endpoint in R, and the volume of region $R(u, r)$, denoted by $vol(R)$, is the weighted distance of the positive edges in R plus the fractional weighted distance of positive edges leaving R. Now, we will introduce the steps of region-growing. The algorithm randomly picks a vertex $u \in \mathcal{V}$ and initializes r to zero. Each region has a initial volume, which is the volume of the entire graph divided by the number of vertices. Then, it grows r so that $R(u, r)$ includes at least another new vertex, and repeat this step until $cut(R) \leq \lambda ln(n+1) \times vol(R)$[4]. $R(u, r)$ is returned as one of the clusters in $\mathcal{G}$. Meanwhile, all vertices in $R(u, r)$ and the corresponding edges are removed from $\mathcal{G}$. The algorithm picks another vertex next and grow its region. This process will be repeated until $\mathcal{G}$ is empty.

Lastly, the largest cluster of $\mathcal{G}$ is regarded as pivot partition. All entities in the clusters which connect to the pivot partition with negative edges are reported as mis-categorized entities.

Example 15 (*Correlation Partition.*) Given the entities in Fig. 1 and the soft positive and negative rules in Example 14, we first build a weighted undirected graph for all the entities in Fig. 1, which is shown in Fig. 6a-1. The black edge has a positive weight, indicating the two entities are likely to be grouped together. The red edge has a negative weight, indicating the two entities possibly should not be grouped together. The weight of edge (e_1, e_3) is 1.42. This is because e_1 and e_3 can satisfy all three positive rules and the sum of rule weights is $0.5 + 0.67 + 0.25 = 1.42$. The weight of edge (e_2, e_4) is

[4] λ is a constant that determines the approximation ratio. It has been proved in [22] that λ must be greater than 2.

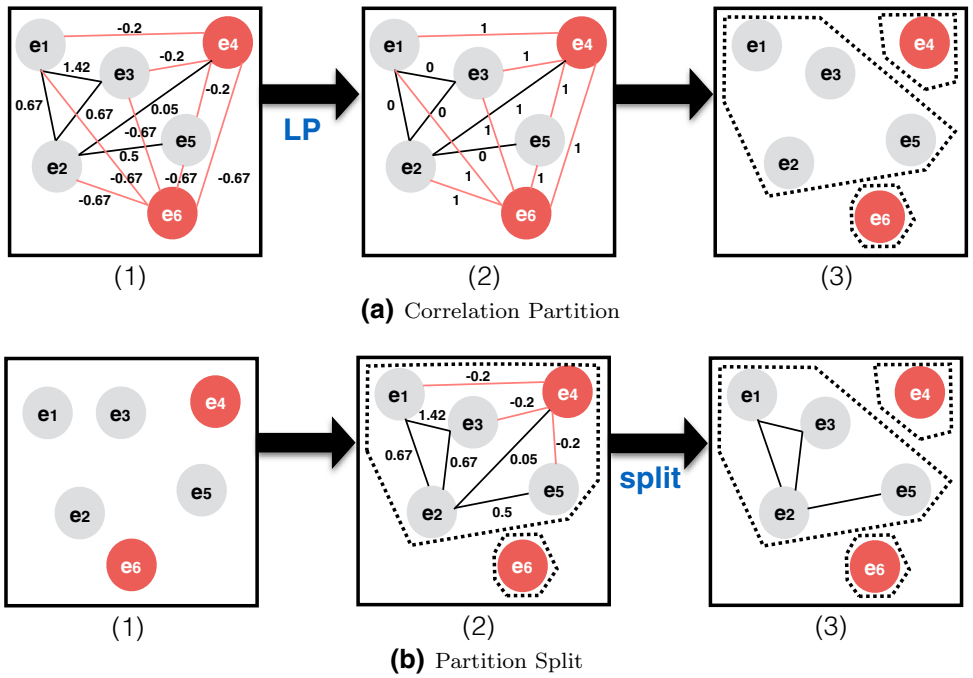


Fig. 6 Soft rule application

0.05, because both $\Psi_3^+(e_2, e_4)$ and $\Phi_1^-(e_2, e_4)$ return true and the sum of rule weights is $0.25 + (-0.2) = 0.05$.

After substituting all edge weights into the linear program, we can get $d_{12} = d_{13} = d_{23} = d_{25} = 0$ and $d_{14} = d_{16} = d_{24} = d_{26} = d_{34} = d_{36} = d_{45} = d_{46} = d_{56} = 1$, which is shown in Fig. 6a-2. Next, we utilize region-growing techniques to group vertices. If we first choose e_1 and initialize r to zero, then we have $R_1(e_1, 0) = \{e_1, e_2, e_3, e_5\}$. At this time, $cut(R_1) = 0.05$, $vol(R_1) = 0.008$, if $\lambda = 4$, $cut(R_1) \leq \lambda ln(n+1) \times vol(R_1)$ is satisfied. Thus, region $R_1(e_1, 0)$ will be returned as a cluster. After removing R_1, only e_4, e_6 and the edge (e_4, e_6) are left. $R_2(e_4, 0) = \{e_4\}$ and $cut(R_2) = 0$. $cut(R_2) \leq \lambda ln(n+1) \times vol(R_2)$ is satisfied and R_2 will be returned as another cluster. As a result, the entities are grouped into $\{e_1, e_2, e_3, e_5\}$, $\{e_4\}$ and $\{e_6\}$ as shown in Fig. 6a-3.

7.3.2 Partition split

The correlation partition method is time-consuming: When the amount of entities increases, the rapid growth of the number of variables and constraints makes its performance drop rapidly. Besides, the value of λ need to be carefully designed. If we set $\lambda = 3$, $cut(R_1) \leq \lambda ln(n+1) \times vol(R_1)$ is just hold when R_1 is grown into $R_1(e_1, 1) = \{e_1, e_2, e_3, e_4, e_5, e_6\}$ and all entities are grouped into the same cluster. In the following, we propose a new method which first computes disjoint partitions and then checks whether each partition should be split.

Soft positive rules are first used to group entities into partitions. Due to the inconsistency of rules, there must exist two entities (e, e') that belong to the same partition but also satisfy a soft negative rule. In this situation, we need to consider whether these two entities should be separated into different partitions.

Give a partition P, a set of soft positive rules Σ_s^+ and a set of soft negative rules Σ_s^-, we first build a weighted undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ for P. Then, given $(e \in P, e' \in P)$ that $\phi(e, e')$ returns true, if the following conditions are satisfied, we separate the two entities into different partitions.

- The weight of edge between e and e' is negative. That is to say, if both $\varphi^+(e, e')$ and $\phi(e, e')$ return true, $w_{\phi^-} > w_{\varphi^-}$ must hold.
- For each positive path (every edge in the path has a positive weight) between e and e', the absolute value of the weight of edge (e, e') is greater than the smallest edge weight in the path.

If e and e' are separated, we generate two new partitions P' and P'', and put these two entities into different partitions. For another entity $e'' \in P$, let $prob(e'', P')$ denote the probability of entity e'' belonging to partition P', which is the largest value of the minimum weight in each positive path between e and e''. Then if $prob(e'', P') > prob(e'', P'')$, entity e'' will be added into partition P'. Otherwise, e'' will be added into P''.

After checking all negative edges, we get the final partitions. The largest one is regarded as the pivot partition, which can be used to discover mis-categorized entities.

Example 16 (*Partition Split.*) Given the entities in Fig. 1 and the soft positive and negative rules in Example 14, we first use soft positive rules to group entities into two partitions $P_1 : \{e_1, e_2, e_3, e_4, e_5\}$, $P_2 : \{e_6\}$, as shown in Fig. 6b-2. Then, we build a weighted undirected graph for P_1.

After building the graph model, we check each negative edge in turn. Given the edge (e_4, e_5), we find that $|w_{e_4e_5}|$ is greater than the smallest edge weight in the positive path $e_4 - e_2 - e_5$. Thus, e_4 and e_5 should be separated into different partitions. Suppose that $P_3 = \{e_4\}$ and $P_4 = \{e_5\}$, considering entity e_2, since $prob(e_2, P_4) = w_{e_2e_5} > w_{e_2e_4} = prob(e_2, P_3)$, e_2 should be added into P_4. As for e_1 and e_3, the weights of (e_1, e_4) and (e_3, e_4) are negative. On the contrary, $prob(e_1, P_4) = w_{e_2e_5} = 0.5$ and $prob(e_3, P_4) = w_{e_2e_5} = 0.5$. Thus, $P_4 = P_4 \cup \{e_1, e_3\} = \{e_1, e_2, e_3, e_5\}$ and no negative edge exist. P_1 is split into P_3 and P_4. The final result is shown in Fig. 6b-3.

8 Experiments

We conduct experiments to answer the following questions for discovering mis-categorized entities: (Exp-1) How good is our approach compared with entity matching solutions? (Exp-2) How good is our approach compared with machine learning approaches? (Exp-3) How good is our approach compared with outlier detection methods? (Exp-4) What is the effect of multiple negative rules? (Exp-5) What is the effect of positive rules? (Exp-6) How efficient is our signature-based algorithm? (Exp-7) How good are the generated rules? (Exp-8) How good is our method to resolve inconsistent rules?

8.1 Experimental setup

Datasets (1) `Google Scholar`. We crawled 200 Google Scholar pages (*i.e.,* groups), with the average number of 340 entities in a group, all from SIGMOD/VLDB/ICDE PC members [5]. Each entity consists of eight attributes: `Title`, `Authors`, `Date`, `Venue`, `Volume`, `Issue`, `Pages` and `Publisher`. (2) `Amazon Product`. We downloaded the data from http://jmcauley.ucsd.edu/data/amazon/links.html [47]. We used 4286 product categories and each product has 8 attributes: `Asin`, `Title`, `Brand`, `Also_bought`, `Also_viewed`, `Bought_together`, `Buy_after_viewing` and `Description`.

Mis-categorized Entities We did not inject mis-categorized entities to `Google Scholar` because they were dirty originally. Their ground truth was manually verified. For `Amazon Product`, we injected some products from similar categories to a group as mis-categorized entities for better testing the performance *w.r.t.* different error rates under a more controlled evaluation. The errors were produced with a rate $e\%$, *i.e.,* the percentage of the number of mis-categorized entities over all entities.

Ontologies Ontologies are not available for all attributes. Fortunately, some are publicly available, *e.g.,* Google Scholar Metrics for `Venue` (see Sect. 3). For the attributes that do not have ontologies in presence, we can build them. For instance, for product description, we utilized Latent Dirichlet Allocation (LDA) [20] to learn a hierarchy structure as follows. Given a set S of product descriptions, we first learned k topics from S using LDA, and categorized each document to the most likely topic. The set S was divided into $\{S_1, ..., S_k\}$. Then, for each subset S_i, we continued to learn m topics from S_i and divided it into $\{S_{i_1}, ..., S_{i_m}\}$. Then, we had three layers: The root was a dummy node; the second layer had k nodes corresponding to the k topics learned from S, and the third layer had $k \times m$ nodes learned from each S_i. If there were still many documents in a topic, we would further split it.

Positive/Negative Rules The positive/negative rules used in Exp-1 to Exp-6 were generated as described in Sect. 6 which are consistent. For `Google Scholar`, we used two positive rules and three negative rules, learned from 229 positive examples and 201 negative examples.

$$\varphi_1^+ : f_{ov}(\texttt{Authors}) \geq 2$$
$$\varphi_2^+ : f_{ov}(\texttt{Authors}) \geq 1 \wedge f_{on}(\texttt{Venue}) \geq 0.75$$
$$\phi_1^- : f_{ov}(\texttt{Authors}) = 0$$
$$\phi_2^- : f_{ov}(\texttt{Authors}) \leq 1 \wedge f_{on}(\texttt{Venue}) \leq 0.25$$
$$\phi_3^- : f_{ov}(\texttt{Authors}) \leq 1 \wedge f_{on}(\texttt{Title}) \leq 0.25$$

Three positive rules and two negative rules were applied in `Amazon Product`, learned using 247 positive examples and 245 negative examples. The entities associated with these training examples were removed for testing data.

$$\varphi_3^+ : f_{ov}(\texttt{Also\_bought}) \geq 2 \wedge f_{ov}(\texttt{Also\_viewed}) \geq 2$$
$$\varphi_4^+ : f_{ov}(\texttt{Bought\_together}) \geq 1 \wedge f_{on}(\texttt{Description}) \geq 0.75$$
$$\varphi_5^+ : f_{ov}(\texttt{Buy\_after\_viewing}) \geq 1 \wedge f_{on}(\texttt{Description}) \geq 0.75$$
$$\phi_4^- : f_{ov}(\texttt{Also\_bought}) = 0 \wedge f_{on}(\texttt{Description}) \leq 0.5$$
$$\phi_5^- : f_{ov}(\texttt{Also\_viewed}) = 0 \wedge f_{on}(\texttt{Description}) \leq 0.5$$

Algorithms We have implemented the following algorithms. (1) `DIME`: the basic algorithm in Sect. 4; (2) `DIME`$^+$: the signature-based fast algorithm in Sect. 5; (3) `Corr`: the correlation method in Sect. 7.3.1; and (4) `Split`: the partition split method in Sect. 7.3.2.

[5] We gathered the Google Scholar data at the end of the year 2016, which may be dirtier than the current Google Scholar pages. However, mis-categorized entities still exist in the current version due to the paper assignment method applied in Google Scholar.

For comparison, we have implemented (5) CR [8], which is a collective relational entity linkage solution based on the hierarchical clustering; (6) Dedupalog [4], which is a constraint-based collective deduplication method; (7) SVM [9], a machine learning (ML) approach for classifying entity, and (8) LOF [10], one of the most popular unsupervised outlier detection methods. For rule generation, we compared with (9) DecisionTree [32], which is a ML-based rule generation method; and (10) SIFI [62], a heuristic-based approach that searches optimal similarity functions and thresholds.

Algorithm Parameters For CR, we picked three thresholds 0.5, 0.6, 0.7 and show the best results. That is, when the minimum distance between two groups was larger than the thresholds, CR stopped merging groups and terminated. We used SVM with linear kernel and balanced class weights to optimize F-measure. As for LOF, if the LOF score of an entity is larger than 1, it was regarded as mis-categorized entity. Since the score of each entity is determined by the average local density of its k nearest neighbors, we tried different k in [5,10] and report the best results. DecisionTree was run with maximum depth 4, and for SIFI, we asked an expert to formulate the structure of rules.

Effectiveness Metrics We used precision, recall, and F-measure to measure the effectiveness. The precision is the ratio of mis-categorized entities we correctly identified to the number of all discovered mis-categorized entities. The recall is the ratio of mis-categorized entities correctly identified to the number of actual mis-categorized entities. And the F-measure (or F_1 score) is the harmonic mean of precision and recall.

Experimental Environment We implemented DIME, DIME$^+$, Corr, Split, CR, Dedupalog and LOF in Java, and obtained the Java implementation of SVM from Bilenko et al. [9]. DecisionTree and SIFI were acquired from the authors. All tests were conducted on a PC with a 2.80GHz Intel CPU and 16GB RAM.

8.2 Experimental results

Exp-1: Comparison with EM Approach We first compared with CR, which is a hierarchical clustering-based EM algorithm that group entities based on their distances. In our experiments, we first ran CR to group entities and then regarded the entities which did not belong to the maximal group as mis-categorized entities. Figure 7 reports the results of comparison about the accuracy. Since Google Scholar was dirty originally, we only varied the error rate of Amazon from 10% to 40%. We used three negative rules for Google Scholar and two negative rules for Amazon. In this part, we report the best precision and recall, when the user dragged the scrollbar. For CR, we tried three thresholds and reported the best.

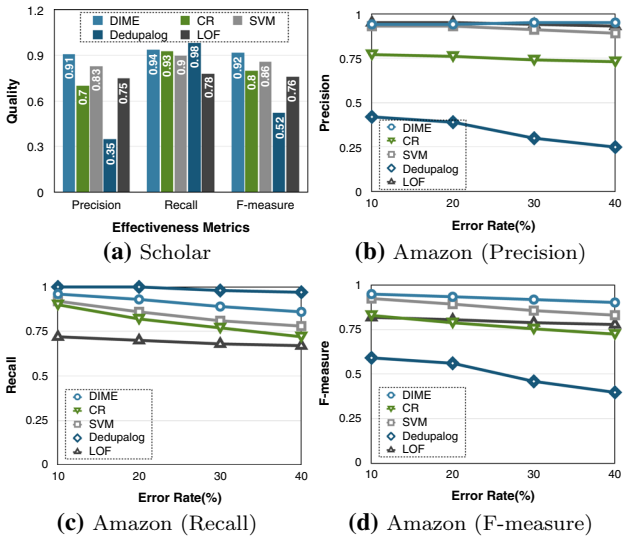


Fig. 7 Comparison with state-of-the-art algorithms

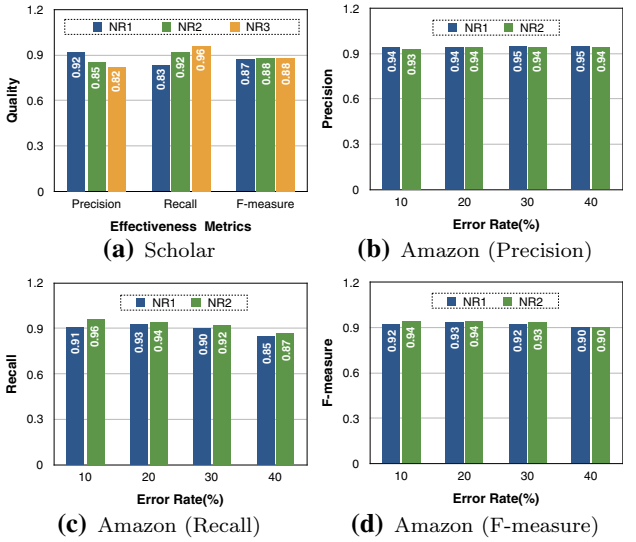


Fig. 8 Effectiveness of scrollbar

Figure 7 shows that our method achieved higher precision and recall than CR and certainly higher F-measure. The reason is that (1) some correct entities appeared in small partitions. Without the help of negative rules, these entities were regarded as mis-categorized entities in CR. That is why EM approaches cannot be directly applied to solve our problem. (2) We used ontology similarity to measure the similarity in attribute Venue and Description rather than traditional string similarity, which resulted in higher accuracy. For example, given two entities with venue "International Conference on Very Large Data Bases" and "ACM Transactions on Database Systems," respectively, CR may put them into different groups because of low string similarity. On the

γ_1 : Paper * (ID,ID') ⇐ Refs(ID,−,Authors,−,−,−,−,−,−), Refs(ID',−,Authors',−,−,−,−,−,−), TwoCommonAuthor(Authors,Authors')
γ_2 : Paper * (ID,ID') ↔ Refs(ID,−,Authors,−,Venue,−,−,−,−), Refs(ID',−,Authors',−,Venue',−,−,−,−), OneCommonAuthor(Authors, Authors'), VenueSimilar(Venue,Venue')
γ_3 : ¬Paper * (ID,ID') ⇐ Refs(ID,−,Authors,−,−,−,−,−,−), Refs(ID',−,Authors',−,−,−,−,−,−), NoCommonAuthor(Authors,Authors')
γ_4 : ¬Paper * (ID,ID') ⇐ Refs(ID,−,Authors,−,Venue,−,−,−,−), Refs(ID',−,Authors',−,Venue',−,−,−,−), LessThanTwoCommonAuthor (Authors,Authors'), VenueDissimilar(Venue,Venue')
γ_5 : ¬Paper * (ID,ID') ⇐ Refs(ID,Title,Authors,−,−,−,−,−,−), Refs(ID',Title',Authors',−,−,−,−,−,−), LessThanTwoCommonAuthor (Authors,Authors'), TitleDissimilar(Title,Title')
η_1 : Product * (Asin,Asin') ⇐ Refs(Asin,−,−,Also_bought,Also_viewed,−,−,−), Refs(Asin',−,−,Also_bought',Also_viewed',−,−,−), TwoCommonAlsoBought(Also_bought,Also_bought'), TwoCommonAlsoViewed(Also_viewed,Also_viewed')
η_2 : Product * (Asin,Asin') ↔ Refs(Asin,−,−,−,−,Bought_together,−,Description), Refs(Asin',−,−,−,−,Bought_together',−, Description'), OneCommonBoughtTogether(Bought_together,Bought_together'), DescriptionSimilar (Description,Description')
η_3 : Product * (Asin,Asin') ← Refs(Asin,−,−,−,−,−,Buy_after_viewing,Description), Refs(Asin',−,−,−,−,−,Buy_after_viewing', Description'), OneCommonBuyAfter(Buy_after_viewing,Buy_after_viewing'), DescriptionSimilar (Description,Description')
η_4 : ¬Product * (Asin,Asin') ⇐ Refs(Asin,−,−,Also_bought,−,−,−,Description), Refs(Asin',−,−,Also_bought',−,−,−,Description'), NoCommonAlsoBought(Also_bought,Also_bought'), DescriptionDissimilar(Description,Description')
η_5 : ¬Product * (Asin,Asin') ⇐ Refs(Asin,−,−,−,Also_viewed,−,−,Description), Refs(Asin',−,−,−,Also_viewed,−,−,Description'), NoCommonAlsoViewed(Also_viewed,Also_viewed'), DescriptionDissimilar(Description,Description')

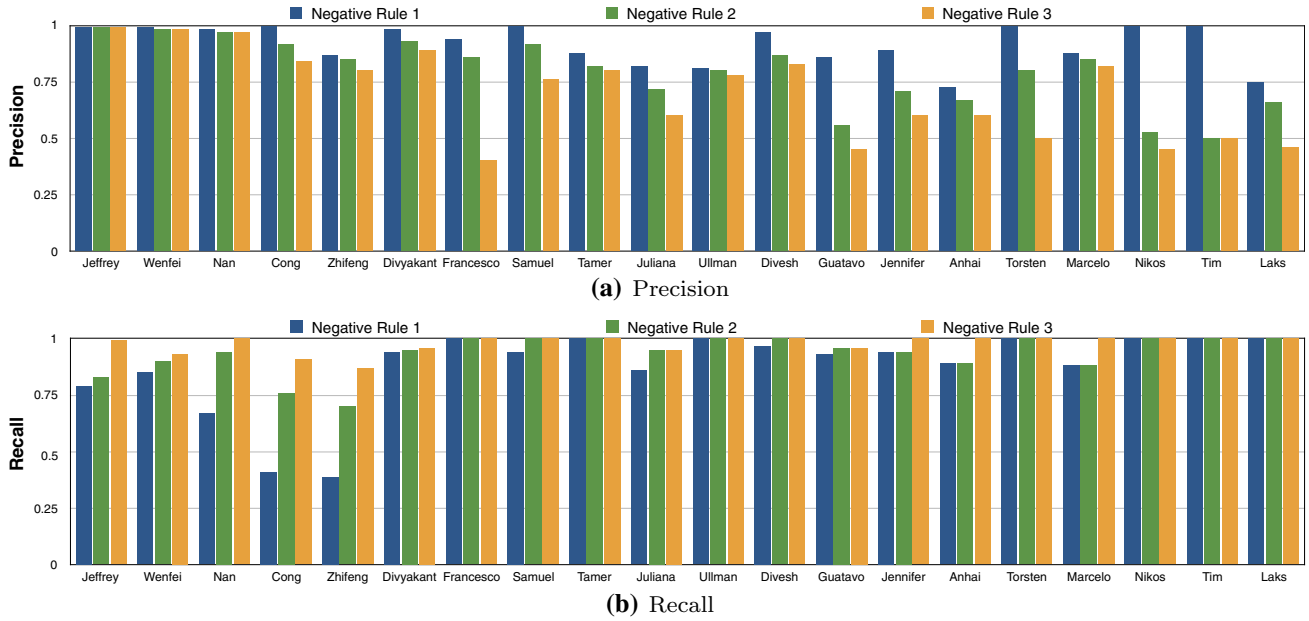
Fig. 9 Constraints used in Dedupalog

other hand, we can judge that these two venues belong to the same field. (3) CR started with the resolutions that only considered attribute string similarity. As an iterative method, one incorrect decision in CR lead to more errors in the following iterations.

We then compared with Dedupalog, a constraint-based EM method which aims to generate a set of partitions of the input records that minimizes the number of soft constraints that are violated, while ensuring that no hard constraint is violated. We designed similar constraints for Dedupalog which are shown in Fig. 9 ($\gamma_1 - \gamma_5$ for Google Scholar and $\eta_1 - \eta_5$ for Amazon Product). Since the type of rules must be defined and at least one soft-complete rule is required in Dedupalog, we specified that the first positive rule was a hard rule that must be satisfied in any legal clustering and then the second one was a soft-complete rule. The next positive rule was a soft-incomplete rule and all negative rules were hard rules. Here, two entities being clustered together by Dedupalog did not mean that they were deduplicated but should be categorized together following our setting.

From Fig. 7, we can see that our method achieved much higher precision and relatively lower recall which led to higher F − measure. It is because Dedupalog would assign each pair of entities with one of the labels {[=]: should be clustered together, [≠]: should not be clustered together} after hardening, and generated smaller partitions than DIME. When we also regarded the largest partition as correctly categorized partition, every entity that did not belong to this partition were recognized as mis-categorized entity due to the "[≠]" edge connected to the largest partition. That is why the recall of Dedupalog was almost 1 with the similar positive rules. As a tradeoff, the precision of Dedupalog was very low since lots of correctly categorized entities were not in the largest partition which would be mistakenly regarded as mis-categorized entities. On the contrary, DIME can carefully distinguish whether entities not in the largest partition were mis-categorized entities with the help of negative rules.

Exp-2: Comparison with ML Approach We also compared with a ML-based method SVM, which has been shown to outperform other machine learning techniques for classification. We trained two SVM models. The first extracted features from entities and converted each entity to a vector for classification. The positive examples were taken as the entities that should be in the category and the negative examples were mis-categorized entities. In the second one, the features in positive/negative examples were the similarities between two entities that should/should not in the same category. Then, it grouped entities and regarded the entities not in the maximal group as mis-categorized entities. Since the similarities between examples were rather important, the latter model was better. Thus, we used the latter in SVM. Figure 7 shows that our method had higher precision and recall than SVM in Google Scholar. In Amazon dataset, we achieved a little higher precision than SVM but had much higher recall, which is often favored in such applications of discovering mis-categorized entities. When putting together, our method achieved much higher F-measure. It is noteworthy that when error rate $e\%$ increased, the precision of our method moderately increased. This was because when more mis-categorized entities existed in the group, we found that LDA worked better to differentiate the product description of correct entities from mis-categorized entities. Thus, less false positive would exist. As $e\%$ increased, the recall of all methods decreased. The reason is that more mis-categorized entities were injected into the dataset as noise which had sim-



(a) Precision

(b) Recall

Fig. 10 Effectiveness of scrollbar (Google Scholar details)

ilar buying behavior and product description. Thus, it became harder to detect them.

Exp-3: Comparison with Outlier Detection Method Here, we compared our framework DIME with one of the most popular unsupervised outlier detection method LOF, which is adequate for the general case when clusters of different densities exist. The degree of an entity being an outlier depend on how isolated it is with respect to the surrounding neighborhood. We followed the paper [25] to generate the vector representation of each entity, and regarded the entities whose LOF scores were larger than a given threshold as mis-categorized entities. The best results are reported in Fig. 7 after trying different parameter values.

Figure 7 shows that DIME had higher precision and recall, and certainly higher F-measure than LOF. This confirms that the mis-categorized entity is not equivalent to outlier. Concretely, some correctly categorized entities would be regarded as outliers by LOF especially those scattered in small partitions, which led to lower precision. Besides, the mis-categorized entity might be regarded as inlier by LOF which was similar to most entities in many attributes, *e.g.*, e_4 in Fig. 1. Thus, outlier detection methods cannot be directly applied to our problem.

Exp-4: Effectiveness of Tuning Negative Rules In this set of experiments, we studied the effect of tuning negative rules. Recall that we used three negative rules for Google Scholar and two negative rules for Amazon. By default, we showed the user the discovered mis-categorized entities by the first negative rule, and the user can then drag the scrollbar to see the results using other negative rules.

The average results of applying three negative rules for Google Scholar are shown in Fig. 8a, and the results of applying two negative rules for Amazon in different error rates are presented in Fig. 8b–d. The recall value increased when applying more negative rules, which was expected because more mis-categorized entities could be captured. As a trade-off, the precision value decreased, since some correct entities which were not so similar with others were regarded as mis-categorized (Fig. 9).

We present the specific results of 20 Google Scholar pages in Fig. 10, as different groups have different performances when tuning negative rules. For precision, the first negative rule was the best, which verified that our choice using only author names as the default discriminative attribute in the first negative rule was valid. A further observation is that in most cases, using the default negative rule can get the best precision and close to the best recall. Thus in most cases, the user did not need to touch the scroll bar at all. However, there were several cases, such as Nan Tang, Cong Yu, that needed to use more negative rules to find more mis-categorized entities. It deserves to notice that our tool only suggests mis-categorized entities. It was up to the user to decide which ones to remove from their own group. Hence, the high recall and good precision together made our proposal an ideal tool for cleaning mis-categorized entities.

Exp-5: Effectiveness of Positive Rules We tested the goodness of our positive rules of grouping entities, which can better understand the reported results in the above experiments. We omitted the result for Amazon as the result was equally good. Table 1 shows statistics of 20 Google Scholar pages after applying our positive rule to form initial dis-

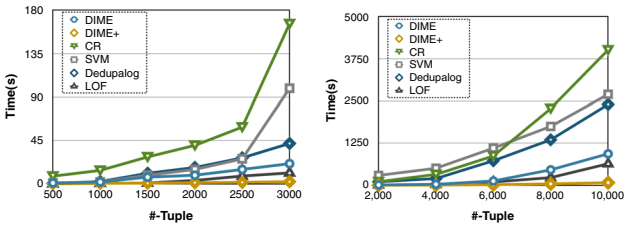
Table 1 Effect of positive rules

Size	[1, 10)			[10, 100)			[100, 1000)		
	#-partitions	#-entities	#-errors	#-partitions	#-entities	#-errors	#-partitions	#-entities	#-errors
Divyakant	104	147	104	2	35	21	2	480	0
Jeffrey	1608	2334	2234	8	158	148	2	508	199
Wenfei	224	348	307	3	138	0	0	0	0
Nan	51	65	55	4	52	11	0	0	0
Cong	50	78	46	2	67	0	0	0	0
Zhifeng	30	48	23	1	50	0	0	0	0
Francesco	60	74	29	2	45	0	1	111	0
Samuel	64	93	18	3	136	0	0	0	0
Tamer	123	173	28	4	75	0	1	109	0
Juliana	64	93	21	3	47	0	1	137	0
Ullman	177	220	40	5	97	0	1	223	0
Anhai	36	45	9	2	102	0	0	0	0
Divesh	67	95	29	2	50	0	1	369	0
Gustavo	116	161	27	8	268	0	0	0	0
Jennifer	71	88	18	2	73	0	1	162	0
Torsten	20	30	4	1	50	0	0	0	0
Marcelo	62	107	8	3	95	0	0	0	0
Nikos	41	76	9	1	55	0	0	0	0
Tim	10	10	3	1	75	0	0	0	0
Laks	64	99	6	4	58	0	1	96	0

joint partitions in step 1 (refer to Fig. 5 for more details). The table reads as follows, taking Divyakant for instance. Given 652 publications in Divy's Google Scholar page, we computed 104 partitions whose sizes were < 10, the number of total entities in these 104 partitions was 147 that contained 104 mis-categorized entities. Also, there were 2 partitions whose sizes were in [10, 100) that contained 35 entities where 21 were mis-categorized entities. Moreover, there were 2 partitions whose sizes were in [100, 1000) that contained 480 entities without any mis-categorized entities. All numbers about mis-categorized entities were highlighted in red. Table 1 first tells us that most mis-categorized entities appeared in small partitions, which verified the effectiveness of our conservative positive rule that successfully isolated them. This also helps to explain why our negative rules can discover mis-categorized entities in the above experiments.

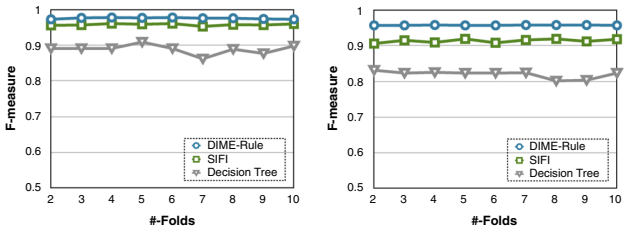
Exp-6: Efficiency Study We compared the efficiency of our algorithms with the baselines `CR`, `Dedupalog`, `SVM` and `LOF`. We sampled six Google Scholar pages by varying the number of tuples from 500 to 3000. For `Amazon Product`, we picked five categories with error rate 40% and varied the number of entities from 2000 to 10,000. Figure 11 shows the running time. Our algorithms were faster than `CR`, `Dedupalog` and `SVM`, because in each iteration, `CR` re-evaluated the attribute distance and reference distance between two groups and selected the closest pair to merge. It was time-consuming; especially, the size of dataset was large. Our efficiency was higher than `Dedupalog` mainly because every pair of entities must be measured with each constraint in `Dedupalog`. As for `SVM`, we used part of the dataset as training data and the remaining dataset as testing data. It took lots of iterations for many entities. `LOF` ran faster than `DIME` with much simpler algorithmic logic, where each entity only needed to be compared with its neighbors.

Impact of Indices With the signature-based framework, the efficiency was significantly improved. As shown in Fig. 11, `DIME`$^+$ was 2–10x faster than `DIME`. For `Google Scholar`, when there were 3000 entities, the runtime of `DIME` was 11 s, but 2.3 s for `DIME`$^+$. For `Amazon`, when there were 10,000 entities, the runtime of `DIME` was 936 s, but 77 s for `DIME`$^+$. The results showed the superiority of our pruning techniques. In reality, the size of group is unlikely to be so large after a former cluster algorithm. But we still utilized a data generator `DBGen` (http://www.cs.utexas.edu/users/ml/riddle/data) to generate some large groups with the number of entities from 20 to 100 k to test the efficiency of our algorithms. We used two positive rules and two negative rules and the results are shown in the table below. Our signature-based algorithm `DIME`$^+$ can run for 100 k entities in 175 s, which was 15x faster than `DIME`.



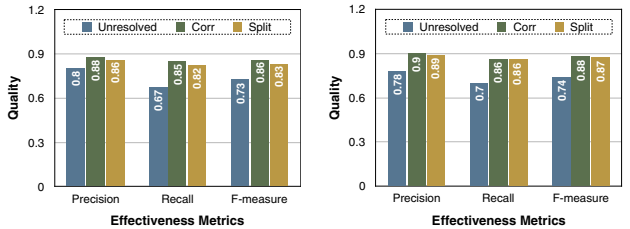
(a) Time (Scholar) **(b)** Time (Amazon)

Fig. 11 Efficiency



(a) F-measure (Scholar) **(b)** F-measure (Amazon)

Fig. 12 Effectiveness of rule generation



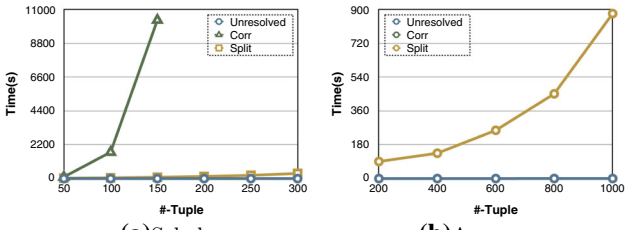
(a) Scholar **(b)** Amazon

Fig. 13 Effectiveness of inconsistency solution

	Gen (20k)	Gen (40k)	Gen (60k)	Gen (80k)	Gen (100k)
DIME	430	619	891	1723	2610
DIME$^+$	39	52	69	133	175

Exp-7: Comparison with Rule Generation Methods We compared our greedy rule generation algorithm DIME-Rule with existing ML methods DecisionTree and heuristic-based approach SIFI. We made cross-validation on the training set, and Fig. 12 reports F-measure values by varying the number of folds. Note that for SIFI, we asked an expert to formulate the structure of rules who was familiar with similarity functions and our datasets. Figure 12 shows that our method can get higher F-measure than DecisionTree and SIFI. DecisionTree failed to find the optimal similarity functions with considerable depth when there were a lot of options, and in SIFI, it was hard for expert to always provide the optimal structure of rules and suitable similarity functions.

Exp-8: Experiments with Inconsistent Rule Set The soft positive/negative rules used in this set of experiments were



(a) Scholar **(b)**Amazon

Fig. 14 Efficiency of inconsistency solution

modified from the consistent rule set above, and the rule weights were calculated based on the positive and negative examples. For Google Scholar, we used one soft positive rule and three soft negative rules.

$$\Psi_1^+ : [f_{ov}(\texttt{Authors}) \geq 1, 0.83]$$
$$\Phi_1^- : [f_{ov}(\texttt{Authors}) = 0, -0.82]$$
$$\Phi_2^- : [f_{ov}(\texttt{Authors}) \leq 1 \wedge f_{on}(\texttt{Venue}) \leq 0.25, -0.79]$$
$$\Phi_3^- : [f_{ov}(\texttt{Authors}) \leq 1 \wedge f_{on}(\texttt{Title}) \leq 0.25, -0.69]$$

Three soft positive rules and two soft negative rules were applied in Amazon Product.

$$\Psi_2^+ : [f_{ov}(\texttt{Also\_bought}) \geq 2 \wedge f_{ov}(\texttt{Also\_viewed}) \geq 2, 0.77]$$
$$\Psi_3^+ : [f_{ov}(\texttt{Bought\_together}) \geq 1 \wedge f_{on}(\texttt{Description}) \geq 0.75, 0.63]$$
$$\Psi_4^+ : [f_{ov}(\texttt{Buy\_after\_viewing}) \geq 1, 0.58]$$
$$\Phi_4^- : [f_{ov}(\texttt{Also\_bought}) = 0 \wedge f_{on}(\texttt{Description}) \leq 0.5, -0.81]$$
$$\Phi_5^- : [f_{ov}(\texttt{Also\_viewed}) = 0 \wedge f_{on}(\texttt{Description}) \leq 0.5, -0.77]$$

We picked smaller groups from Google Scholar and Amazon Product because of the time-out which may be caused by the correlation method on large datasets, and the experimental results are reported in Figs. 13, 14. Here, Unresolved denotes the framework in Algorithm 1. Figure 13 shows that Corr and Split have higher precision and recall than the method without considering the inconsistency between positive and negative rules. The reason is that a negative pair (a pair of entities that should not be in the same category) would also satisfy the positive rules. Corr can recognize this negative semantics, such that when splitting these two entities it can make more rules to be satisfied. Split would put them into different partitions when the weight of negative rule they can satisfy is larger than the possibility of putting them into the same partition. However, in Unresolved, mis-categorized entities would be assigned to the pivot partition in Steps 1–2 of Algorithm 1. Even worse, the entities that are dissimilar to real mis-categorized entities in the pivot partition may be wrongly reported as mis-categorized entities in Step 3 of Algorithm 1. We also tested the performance of our algo-

rithms in the case where a positive pair (a pair of entities that should be in the same category) would satisfy the negative rules. The effectiveness of `Unresolved` was not affected a lot because positive rules were applied first. `Corr` and `Split` can also recognize this positive semantics in most situations. `Corr` had a little higher `precision` and `recall` than `Split`. This is because `Corr` resolved the inconsistency from a global perspective. However, from Fig. 14, we can see that `Corr` was much slower than `Split` even when there were a few entities in a category. Note that we do not present the running time of `Corr` in Fig. 14 when the number of tuples exceeded 150 since the runtime had reached to 10,309 s when #-Tuple = 150.

9 Conclusion

In this paper, we have proposed a new problem that discovers mis-categorized entities. We have presented a general rule-based framework to solve this problems in different applications. We have also proposed a signature-based algorithm to efficiently apply the rules. We have formulated the rule-generation problem, proved that it is NP-hard and proposed exact and approximate solutions. We have discussed how to resolve the inconsistent positive and negative rules. Finally, we have demonstrated both the effectiveness and efficiency of our approach on real datasets.

Acknowledgements This work was supported by NSF of China (Grant Nos. 61902017, 61925205, 61632016, 61521002, 61661166012), Huawei, TAL Education Group, China Postdoctoral Science Foundation (2019M650468) and China Scholarship Council. Note that Ning Wang's partial work was supported by National Key R&D Program of China (2018YFC0809800) National Basic Research Program of China (973 Program) (Grant No. 2015CB358700), Fundamental Research Funds for the Central Universities (Grant No. 2019RC015).

References

1. Abe, N., Zadrozny, B., Langford, J.: Outlier detection by active learning. In: SIGKDD (2006)
2. Aggarwal, C.C.: Outlier ensembles: position paper. ACM SIGKDD Explor. Newslett. **14**(2), 49–58 (2013)
3. Alhelbawy, A., Gaizauskas, R.: Graph ranking for collective named entity disambiguation. In: Annual Meeting of the Association for Computational Linguistics (2014)
4. Arasu, A., Ré, C., Suciu, D.: Large-scale deduplication with constraints using dedupalog. In: ICDE (2009)
5. Bansal, N., Blum, A., Chawla, S.: Correlation clustering. Mach. Learn. **56**(1–3), 89–113 (2004)
6. Bellare, K., Iyengar, S., Parameswaran, A.G., Rastogi, V.: Active sampling for entity matching. In: SIGKDD (2012)
7. Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S.E., Widom, J.: Swoosh: a generic approach to entity resolution. VLDB J. **18**(1), 255–276 (2009)
8. Bhattacharya, I., Getoor, L.: Collective entity resolution in relational data. TKDD **1**(1), 5 (2007)
9. Bilenko, M., Mooney, R.J.: Adaptive duplicate detection using learnable string similarity measures. In: SIGKDD (2003)
10. Breunig, M.M., Kriegel, H., Ng, R.T., Sander, J.: Lof: identifying density-based local outliers. In: SIGMOD (2000)
11. Bunescu, R.C., Pasca, M.: Using encyclopedic knowledge for named entity disambiguation. In: EACL (2006)
12. Campos, G.O., Zimek, A., Sander, J., Campello, R.J., Micenková, B., Schubert, E., Assent, I., Houle, M.E.: On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. Data Min. Knowl. Discov. **30**(4), 891–927 (2016)
13. Chai, C., Li, G., Li, J., Deng, D., Feng, J.: Cost-effective crowdsourced entity resolution: a partial-order approach. In: SIGMOD
14. Chang, C.-C., Lin, C.-J.: Libsvm: a library for support vector machines. ACM TIST **2**(3), 27 (2011)
15. Charikar, M., Guruswami, V., Wirth, A.: Clustering with qualitative information. J. Comput. Syst. Sci. **71**(3), 360–383 (2005)
16. Chawla, S., Makarychev, K., Schramm, T., Yaroslavtsev, G.: Near optimal lp rounding algorithm for correlation clustering on complete and complete k-partite graphs. In: Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing, pp. 219–228. ACM (2015)
17. Chu, X., Ilyas, I.F., Koutris, P.: Distributed data deduplication. In: PVLDB (2016)
18. Cucerzan, S.: Large-scale named entity disambiguation based on wikipedia data. In: EMNLP-CoNLL (2007)
19. Cunningham, P., Delany, S.J.: k-nearest neighbour classifiers. Multiple Classif. Syst. **34**(8), 1–17 (2007)
20. Das, R., Zaheer, M., Dyer, C.: Gaussian LDA for topic models with word embeddings. In: ACL (2015)
21. Das, S., GC, P.S., Doan, A., Naughton, J.F., Krishnan, G., Deep, R., Arcaute, E., Raghavendra, V., Park, Y.: Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In: SIGMOD (2017)
22. Demaine, E.D., Emanuel, D., Fiat, A., Immorlica, N.: Correlation clustering in general weighted graphs. Theor. Comput. Sci. **361**(2–3), 172–187 (2006)
23. Demaine, E.D., Immorlica, N.: Correlation clustering with partial information. In: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, pp. 1–13. Springer (2003)
24. Demartini, G., Difallah, D.E., Cudré-Mauroux, P.: Zencrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In: WWW (2012)
25. Ebraheem, M., Thirumuruganathan, S., Joty, S., Ouzzani, M., Tang, N.: Distributed representations of tuples for entity resolution. In: VLDB (2018)
26. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: a survey. IEEE Trans. Knowl. Data Eng. **19**(1), 1–16 (2007)
27. Eshel, Y., Cohen, N., Radinsky, K., Markovitch, S., Yamada, I., Levy, O.: Named entity disambiguation for noisy text. In: CoNLL (2017)
28. Fellegi, I., Sunter, A.: A theory for record linkage. J. Am. Stat. Assoc. **64**(328), 1183–1210 (1969)
29. Francis-Landau, M., Durrett, G., Klein, D.: Capturing semantic similarity for entity linking with convolutional neural networks. In: Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 1256–1261 (2016)
30. Gabrilovich, E., Markovitch, S. et al.: Computing semantic relatedness using wikipedia-based explicit semantic analysis. In: IJCAI (2007)
31. Gentile, A.L., Zhang, Z., Xia, L., Iria, J.: Graph-based semantic relatedness for named entity disambiguation. In: International Conference on Software, Services and Semantic Technologies (2009)

32. Gokhale, C., Das, S., Doan, A., Naughton, J.F., Rampalli, N., Shavlik, J., Zhu, X.: Corleone: Hands-off crowdsourcing for entity matching. In: SIGMOD (2014)
33. Gravano, L., Ipeirotis, P.G., Jagadish, H.V., Koudas, N., Muthukrishnan, S., Srivastava, D. et al.: Approximate string joins in a database (almost) for free. In: VLDB (2001)
34. Hakimov, S., Oto, S.A., Dogdu, E.: Named entity recognition and disambiguation using linked data and graph-based centrality scoring. In: International workshop on semantic web information management. ACM (2012)
35. Hao, S., Tang, N., Li, G., Feng, J.: Discovering mis-categorized entities. In: ICDE (2018)
36. Hao, S., Xu, Y., Tang, N., Li, G., Feng, J.: Cleaning your wrong google scholar entries. In: ICDE demo (2018)
37. Hazman, M., El-Beltagy, S.R., Rafea, A.: A survey of ontology learning approaches. Database **7**, 6 (2011)
38. He, Z., Xu, X., Deng, S.: Discovering cluster-based local outliers. Pattern Recognit. Lett.. **24**(9–10), 1641–1650 (2003)
39. Hochba, D.S.: Approximation algorithms for np-hard problems. ACM Sigact News **28**(2), 40–52 (1997)
40. Hu, Z., Huang, P., Deng, Y., Gao, Y., Xing, E.: Entity hierarchy embedding. In: ACL (2015)
41. Jiang, Y., Li, G., Feng, J., Li, W.: String similarity joins: an experimental evaluation. In: PVLDB (2014)
42. Karpinski, M., Schudy, W.: Linear time approximation schemes for the Gale-Berlekamp game and related minimization problems. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, pp. 313–322. ACM (2009)
43. Kolb, L., Thor, A., Rahm, E.: Dedoop: efficient deduplication with hadoop. In: PVLDB (2012)
44. Köpcke, H., Rahm, E.: Training selection for tuning entity matching. In: Program Committee Workshop on Management of Uncertain Data, p. 3 (2008)
45. Liaw, A., Wiener, M., et al.: Classification and regression by randomforest. R News **2**(3), 18–22 (2002)
46. Lippmann, R.P.: Anintroduction to computing with neural nets. IEEE ASSP Mag. **4**(2), 4–22 (1987)
47. McAuley, J., Targett, C., Shi, Q., Van Den Hengel, A.: Image-based recommendations on styles and substitutes. In: SIGIR (2015)
48. Phua, C., Alahakoon, D., Lee, V.: Minority report in fraud detection: classification of skewed data. ACM SIGKDD Explor. Newslett. **6**(1), 50–59 (2004)
49. Quinlan, J.R.: C4. 5: Programs for Machine Learning. Elsevier, Amsterdam (2014)
50. Ramaswamy, S., Rastogi, R., Shim, K.: Efficient algorithms for mining outliers from large data sets. In: ACM Sigmod Record, vol. 29, pp. 427–438. ACM (2000)
51. Rish, I. et al.: An empirical study of the Naive Bayes classifier. In: IJCAI workshop (2001)
52. Sarawagi, S., Bhamidipaty, A.: Interactive deduplication using active learning. In: KDD (2002)
53. Singh, R., Meduri, V.V., Elmagarmid, A.K., Madden, S., Papotti, P., Quiané-Ruiz, J., Solar-Lezama, A., Tang, N.: Synthesizing entity matching rules by examples. In: PVLDB (2017)
54. Singla, P., Domingos, P.: Entity resolution with Markov logic. In: ICDM (2006)
55. Steinwart, I., Hush, D., Scovel, C.: A classification framework for anomaly detection. J. Mach. Learn. Res. **6**(Feb), 211–232 (2005)
56. Sun, Y., Lin, L., Tang, D., Yang, N., Ji, Z., Wang, X.: Modeling mention, context and entity with neural networks for entity disambiguation. In: IJCAI (2015)
57. Swamy, C.: Correlation clustering: maximizing agreements via semidefinite programming. In: Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 526–527. Society for Industrial and Applied Mathematics (2004)
58. Vesdapunt, N., Bellare, K., Dalvi, N.N.: Crowdsourcing algorithms for entity resolution. In: Proceedings of VLDB Endow (2015)
59. Vilalta, R., Ma, S.: Predicting rare events in temporal domains. In: ICDM (2002)
60. Wang, J., Kraska, T., Franklin, M.J., Feng, J.: Crowder: crowdsourcing entity resolution. In: PVLDB (2012)
61. Wang, J., Li, G., Kraska, T., Franklin, M. J., Feng, J.: Leveraging transitive relations for crowdsourced joins. In: SIGMOD (2013)
62. Wang, J., Li, G., Yu, J. X., Feng, J.: Entity matching: How similar is similar. In: PVLDB (2011)
63. Weiss, G. M., Hirsh, H.: Learning to predict rare events in event sequences. In: KDD (1998)
64. Yamada, I., Shindo, H., Takeda, H., Takefuji, Y.: Joint learning of the embedding of words and entities for named entity disambiguation. In: The SIGNLL Conference on Computational Natural Language Learning (2016)
65. Zimek, A., Campello, R.J., Sander, J.: Ensembles for unsupervised outlier detection: challenges and research questions a position paper. ACM SIGKDD Explor. Newslett. **15**(1), 11–22 (2014)