The VLDB Journal (2020) 29:1311–1336
https://doi.org/10.1007/s00778-020-00613-w

REGULAR PAPER

A game-based framework for crowdsourced data labeling

Jingru Yang[1] · Ju Fan[1] · Zhewei Wei[1] · Guoliang Li[2] · Tongyu Liu[1] · Xiaoyong Du[1]

Received: 16 September 2019 / Revised: 30 January 2020 / Accepted: 12 April 2020 / Published online: 19 May 2020

Abstract

Data labeling, which assigns data with multiple classes, is indispensable for many applications, such as machine learning and data integration. However, existing labeling solutions either incur expensive cost for large datasets or produce noisy results. This paper introduces a cost-effective labeling approach and focuses on the labeling rule generation problem that aims to generate high-quality rules to largely reduce the labeling cost while preserving quality. To address the problem, we first generate candidate rules and then devise a game-based crowdsourcing approach CROWDGAME to select high-quality rules by considering *coverage* and *accuracy*. CROWDGAME employs two groups of crowd workers: One group answers rule validation tasks (whether a rule is valid) to play a role of *rule generator*, while the other group answers tuple checking tasks (whether the label of a data tuple is correct) to play a role of *rule refuter*. We let the two groups play a two-player game: Rule generator identifies high-quality rules with large coverage, while rule refuter tries to refute its opponent rule generator by checking some tuples that provide enough evidence to reject rules with low accuracy. This paper studies the challenges in CROWDGAME. The first is to balance the trade-off between coverage and accuracy. We define the loss of a rule by considering the two factors. The second is rule accuracy estimation. We utilize *Bayesian estimation* to combine both rule validation and tuple checking tasks. The third is to select crowdsourcing tasks to fulfill the game-based framework for minimizing the loss. We introduce a minimax strategy and develop efficient task selection algorithms. We also develop a hybrid crowd-machine method for effective label assignment under budget-constrained crowdsourcing settings. We conduct experiments on entity matching and relation extraction, and the results show that our method outperforms state-of-the-art solutions.

Keywords Crowdsourcing · Data labeling · Labeling rules

1 Introduction

In many applications, such as data integration and machine learning (ML), it is indispensable to obtain large-scale *labeled datasets* with high quality. For example, deep learning (DL) has become a major advancement in machine learning and achieves state-of-the-art performance in many tasks, such as image recognition and natural language processing [20]. However, most of the DL methods require massive training sets to achieve superior performance [37], which usually causes significant labeling costs or efforts.

To address the problem, crowdsourcing can be utilized to harness the crowd to directly label tuples in a dataset at relatively low cost (see a survey [23]). However, as many datasets contain tens of thousands to millions of tuples, such *tuple-level labeling* still inevitably incurs large labeling cost. Another approach is to leverage *labeling rules* that label multiple tuples. For example, in *relation extraction* that identifies structural relations, say `spouse` relation, from unstructured data, labeling rules like "*A is married to B*" can be used to label tuples like `Michelle Obama` and `Barack Obama`. Unfortunately, it is challenging to construct labeling rules. *Handcrafted* rules from domain experts are not scalable, as

✉ Ju Fan
fanj@ruc.edu.cn

Jingru Yang
hinsonver@ruc.edu.cn

Zhewei Wei
zhewei@ruc.edu.cn

Guoliang Li
liguoliang@tsinghua.edu.cn

Tongyu Liu
ltyzzz@ruc.edu.cn

Xiaoyong Du
duyong@ruc.edu.cn

[1] Renmin University of China, Beijing 100872, China

[2] Tsinghua University, Beijing 100084, China

ID	Product Name
s_1	Sony VAIO Silver Laptop
s_2	Apple Pro Black Notebook
s_3	Apple MacBook Pro Silver Laptop
s_4	Apple 8GB Silver 4G iPod
s_5	MacBook Pro Notebook Silver

(a) Product Records

Blocking Rule	Coverage
r_1: (Sony, Apple)	e_1, e_2, e_3
r_2: (VAIO, MacBook)	e_2, e_4
r_3: (Black, Silver)	e_1, e_5, e_6, e_7
r_4: (Laptop, Notebook)	e_1, e_4, e_5, e_9

(b) Blocking Rules

Tuple	True Label
$e_1 = (s_1, s_2)$	-1
$e_2 = (s_1, s_3)$	-1
$e_3 = (s_1, s_4)$	-1
$e_4 = (s_1, s_5)$	-1
$e_5 = (s_2, s_3)$	1
$e_6 = (s_2, s_4)$	-1
$e_7 = (s_2, s_5)$	1
$e_8 = (s_3, s_4)$	-1
$e_9 = (s_3, s_5)$	1
$e_{10} = (s_4, s_5)$	-1

(c) Tuples (Record Pairs)

Fig. 1 Rule-based data labeling in entity matching

it is time and effort consuming to handcraft many rules with large coverage. *Weak-supervision* rules automatically generated [32,33], e.g., distant supervision rules, can largely cover the tuples; however, they may be very noisy and provide wrong labels.

In this paper, we study the data labeling problem that assigns data with multiple classes (i.e., labels), and focus on examining *labeling rule generation using crowdsourcing* to reduce labeling cost while preserving high quality. We aim at generating "high-quality" rules using two objectives: 1) *high coverage*: selecting the rules that cover as many tuples as possible to label the data. Intuitively, the larger the coverage is, the higher the cost on tuple-level labeling could be reduced and 2) *high accuracy*: preferring the rules that induce few wrong labels on their covered tuples.

Example 1 To illustrate the problem, let us consider an application of entity matching (EM) [7], i.e., identifying whether a pair of product records is the same entity, as shown in Fig. 1[1]. In the application, we regard each product pair as a tuple and aim to assign each tuple with one of the two possible labels: `matched` and `unmatched`. To save labeling cost, we can introduce blocking rules that discriminate product pairs and assign the product pairs as `unmatched`. Some example blocking rules, each of which is represented by two keywords, are shown in Fig. 1b. For example, r_1 : (`Sony`, `Apple`) expresses that any product pair, with one product containing keyword `Sony` and the other containing `Apple`, is labeled as `unmatched`. Among these rules, we can see that r_4 has larger coverage, i.e., covering more tuples, than r_2. However, r_4 is less accurate, as it only correctly labels two out of four tuples (e_1 and e_4). Thus, our goal is to select labeling rules with large coverage and high accuracy.

Labeling rule generation is very challenging as there may be many rules with diverse quality. Even worse, although easy to know coverage of rules, it is hard to obtain rule accuracy. To address this problem, we propose to utilize crowdsourcing for rule selection. A straightforward approach employs the crowd to answer a *rule validation* task is to check whether a rule is valid or invalid. Unfortunately, the crowd may give low-quality answers for a rule validation task, as a rule may cover many tuples and the workers cannot examine all the tuples covered by the rule. To alleviate this, we introduce a type of *tuple checking* task, which asks the crowd to give the label of a tuple and utilizes the result to validate/invalidate rules that cover the tuple. However, it is expensive to ask many tuple checking tasks.

We devise a two-pronged crowdsourcing scheme that first uses rule validation tasks as a *coarse pre-evaluation* step and then applies tuple checking tasks as a *fine post-evaluation* step. To effectively utilize the two types of tasks, we introduce a *game-based* crowdsourcing approach CrowdGame. It employs two groups of crowd workers: One group answers rule validation tasks to play a role of *rule generator*, while the other group answers tuple checking tasks to play a role of *rule refuter*. We let the two groups play a *two-player game*: Rule generator identifies high-quality rules with large coverage and accuracies, while rule refuter tries to refute rule generator by checking some tuples that provide enough evidence to "reject" more rules.

We study the research challenges in our game-based crowdsourcing. First, it is challenging to formalize the *quality* of a rule by trading off its accuracy and coverage. We introduce the *loss* of a rule set that combines the uncovered tuples and the incorrectly covered tuples. We aim to select a rule set to minimize the loss. Second, it is hard to obtain the real accuracy of a rule. To address the challenge, we utilize the *Bayesian estimation* technique. We regard crowd rule validation results as a *prior*, which captures crowd judgment without inspecting any specific tuples. As the prior may not be precise, we then use the crowd results on tuple checking as "data observation" to adjust the prior, so as to obtain a *posterior* of rule accuracy. Third, it is hard to obtain high-quality rules to minimize the loss under our framework. We develop a *minimax strategy*: Rule generator plays as a *minimizer* to identify rules to minimize the loss; rule refuter plays as a *maximizer* to check tuples for maximizing the loss. We iteratively call rule generator and rule refuter to select the rules. Fourth, in many circumstances, the generated rules are insufficient to cover all the tuples. To address this, we introduce a hybrid label assignment method to combine crowdsourcing and deep learning models.

To summarize, we make the following contributions:

[1] Note that the matching criterion is the same product *model* and the same *manufacture*, without considering specifications like color and storage.

(1) We propose a data labeling approach using game-based crowdsourcing for labeling rule generation. We employ two groups of crowd workers and let the workers play a *two-player game* to select high-quality rules.
(2) We introduce the *loss* of a rule set that combines uncovered and incorrectly covered tuples to balance coverage and accuracy. We estimate accuracy of a rule by combining rule validation and tuple checking through Bayesian estimation and develop effective task selection algorithms.
(3) We develop a hybrid crowd-machine label assignment method for data labeling under total crowdsourcing budget constraints.
(4) We conducted experiments on real entity matching and relation extraction datasets. The results show that our approach outperforms state-of-the-art solutions on tuple-level crowdsourcing and ML-based consolidation of labeling rules.

This article extends our conference version [49], where the main extension is summarized as follows:

– We introduce a hybrid crowd-machine label assignment method to address the budget-constrained crowdsourcing setting. Section 6 presents the method, and Sect. 7.6 reports the experimental results.
– We provide proofs for all the lemmas and theorems in the paper (Appendix A).
– We add additional experiments, such as empirical observations of candidate rules (Sect. 7.2), effect of parameter settings (Sect. 7.7), and examples of labeling rules (Appendix B).

2 Problem formulation

This paper studies the *data labeling* problem. Given a set $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$ of data tuples, the problem aims to annotate each tuple $e_i \in \mathcal{E}$ with one of the l possible labels, denoted by $\mathcal{L} = \{L_1, L_2, \ldots, L_l\}$. Without loss of generality, this paper focuses on the *binary labeling* problem that considers two possible labels, denoted as $\mathcal{L} = \{L_1 = -1, L_2 = 1\}$. For example, in the application of entity matching, each tuple is a product record pair, and it needs to be labeled with one of the two labels $L_1 = -1$ (unmatched) and $L_2 = 1$ (matched). Another application is relation extraction [29] from the text data (e.g., sentences), which identifies whether a tuple consisting of two entities, say `Barack Obama` and `Michelle Obama`, have a target relation (label $L_2 = 1$), say `spouse`, or not ($L_1 = -1$).

2.1 Overview of our framework

We introduce a cost-effective data labeling framework as shown in Fig. 2. The framework makes use of the *labeling rules* (*rules* for simplicity), each of which can be used to label multiple tuples in $\mathcal{E}$, to reduce the cost.

Definition 1 (Labeling rule) A labeling rule is a function $r_j : \mathcal{E} \to \{L, \texttt{nil}\}$ that maps tuple $e_i \in \mathcal{E}$ into either a label $L \in \mathcal{L}$ or `nil` (which means r_j does not cover e_i). In particular, let $\mathcal{C}(r_j) = \{e | r_j(e) \neq \texttt{nil}\}$ denote the covered tuple set of r_j, $\mathcal{C}(\mathcal{R}) = \{e | \exists r \in \mathcal{R}, r(e) \neq \texttt{nil}\}$ denote the covered set of a rule set $\mathcal{R}$, and $|\mathcal{C}(\mathcal{R})|$ denote the size of $\mathcal{C}(\mathcal{R})$, called the coverage of the rule set $\mathcal{R}$.

Our framework labels a set of unlabeled tuples utilizing labeling rules in two phases.

Phase I: Crowdsourced rule generation. This phase aims at generating "high-quality" rules, where rule quality is measured by *coverage* and *accuracy*.

We first construct candidate rules, which may have various coverage and accuracy. There are two widely used ways to construct candidate rules: *handcrafted* rules from domain experts and *weak-supervision* rules automatically generated by algorithms. Handcrafted rules ask users to write domain-specific labeling heuristics based on their domain knowledge. However, the rules are not scalable, especially for large datasets, as it is time and effort consuming to handcraft many rules with large coverage. Weak-supervision rules automatically generated are introduced [32,33], e.g., distant supervision rules in information extraction, like utilizing textual patterns, such as "*A marries B*," as rules for labeling `spouse` relation between entities A and B. Weak-supervision rules can largely cover the tuples; however, some of them may be very unreliable that provide wrong labels.

To address this problem, we propose to study a problem of *rule generation using crowdsourcing*, to leverage the crowd on identifying good rules from noisy candidates. We will formalize this problem in Sect. 2.2.

Phase II: Label Assignment. This phase first labels tuples using the high-quality rules generated in the previous phase. However, as the rules may not cover all tuples, there exist a proportion of tuples, which are not assigned with labels. For these tuples, we devise techniques by considering the following two settings:

– If there is no crowdsourcing budget constraint, we simply use conventional tuple-level labeling [4,43] to label all the tuples, where answer inference, such as transitivity, can also be applied.
– If there is some crowdsourcing budget constraint, where not all the tuples can be crowdsourced, we first judiciously select the tuples with higher "utility" for crowdsourcing until the budget is used up. Then, we leverage

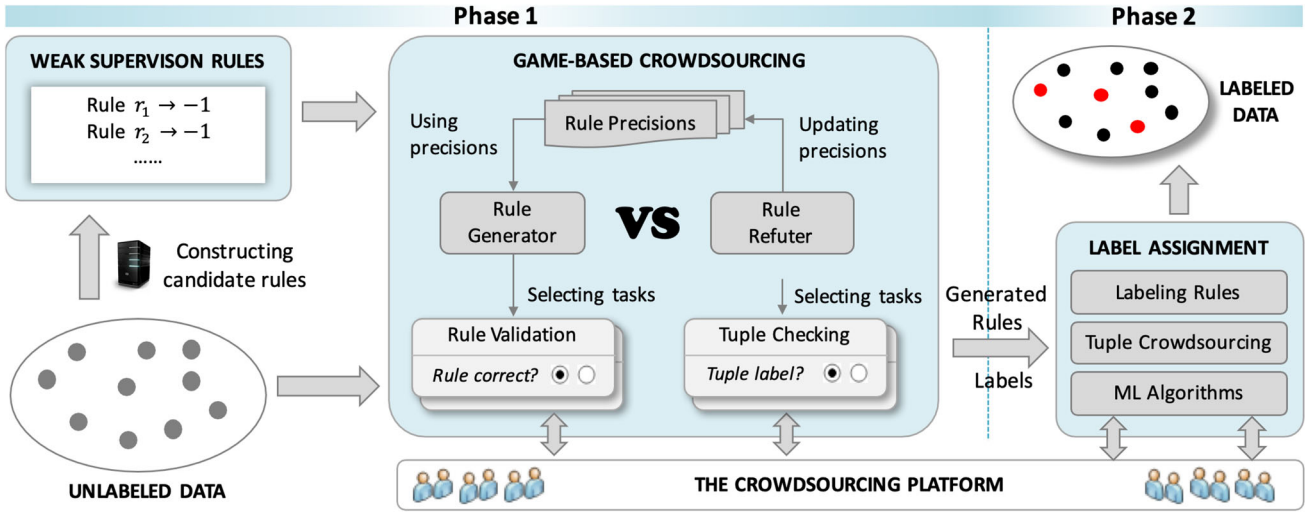


Fig. 2 Framework of data labeling with game-based crowdsourcing

machine learning algorithms to infer labels for the remaining tuples.

Recall the entity matching task example in Fig. 1. Our framework uses crowdsourcing to select the rules with large coverage and accuracy from the candidates (i.e., Phase I). Suppose that $\{r_1, r_3\}$ are selected, and then, six tuples can be labeled by the rules in Phase II. Now, as the other four tuples are still not covered, the framework can label them using conventional crowdsourcing or utilize machine learning algorithms if there are crowdsourcing budget constraints.

For ease of presentation, this paper focuses on the "unary" case that all rules annotate only one label ($|\mathcal{L}| = 1$), e.g., $L_1 = -1$ in our entity matching example. We will discuss a more general case that some rules label $L_1 = -1$, while others provide $L_2 = 1$ in Appendix C.

2.2 Labeling rule generation

Labeling rule generation in the first phase of our framework is the focus of this paper. Intuitively, the problem aims to identify "high-quality" rules with the following two objectives:

- *High coverage*: selecting the rules that cover as many tuples as possible. According to our framework, the larger the coverage of rules is, the higher the cost on tuple-level labeling (Phase II) could be reduced.
- *High accuracy*: preferring the rules that induce few wrong labels on their covered tuples.

There may be a trade-off between coverage and accuracy. On the one hand, some labeling scenarios prefer *accuracy*. For instance, in most of entity matching tasks, ground-truth labels are very skew. Thus, rule generation prefers not to induce too many errors, which may lead to low quality (e.g., F-measure) of the overall entity matching process. On the other hand, some scenarios prefer *coverage* for more labeling cost reduction.

To balance the preference among coverage and accuracy, we introduce the *loss* of a rule set $\mathcal{R}$ that considers two factors. Formally, consider a set $\mathcal{R}$ of rules that annotate label $L \in \mathcal{L}$ to tuple set $\mathcal{E}$. The first factor is the number of uncovered tuples $|\mathcal{E}| - |\mathcal{C}(\mathcal{R})|$ that formalizes the loss in terms of the coverage, and this factor is easy to compute. In contrast, the number of errors, i.e., incorrectly labeled tuples, is hard to obtain, as true labels of tuples are unknown. Thus, we introduce $P(y_i \neq L)$ that denotes the probability that true label y_i of tuple e_i is not L, and consider the expected number of errors $\sum_{e_i \in \mathcal{C}(\mathcal{R})} P(y_i \neq L)$ as the second factor. We define the *loss* of a rule set $\mathcal{R}$ as follows:

Definition 2 (Loss of rule set) The loss $\Phi(\mathcal{R})$ of a rule set $\mathcal{R}$ is defined as a combination of the number of uncovered tuples $|\mathcal{E}| - |\mathcal{C}(\mathcal{R})|$ and the expected number of errors $\sum_{e_i \in \mathcal{C}(\mathcal{R})} P(y_i \neq L)$:

$$\Phi(\mathcal{R}) = \gamma(|\mathcal{E}| - |\mathcal{C}(\mathcal{R})|) + (1-\gamma) \sum_{e_i \in \mathcal{C}(\mathcal{R})} P(y_i \neq L), \quad (1)$$

where γ is a tunable parameter between [0, 1] to balance the preference among coverage and accuracy of the generated rule set $\mathcal{R}$.

For example, consider $\mathcal{R}_1 = \{r_1\}$ covering three tuples without errors and $\mathcal{R}_2 = \{r_1, r_3\}$ covering more but with errors (e_5 and e_7). As entity matching prefers accuracy over coverage on the blocking rules, one needs to set a small parameter γ, say $\gamma = 0.1$. Under this setting, we have

Table 1 Table of notations

y_i	True label of tuple e_i
r_j; $\mathcal{R}$	A labeling rule; a set of rules
$\mathcal{C}(r)$ ($\mathcal{C}(\mathcal{R})$)	A set of tuples covered by rule r (rule set $\mathcal{R}$)
L	Label annotated by $\mathcal{R}$ to tuples
$\lambda_j(\hat{\lambda}_j)$	Accuracy (accuracy estimate) of rule r_j
$\Phi(\mathcal{R})$	Loss of a rule set $\mathcal{R}$

$\Phi(\mathcal{R}_1) < \Phi(\mathcal{R}_2)$, which shows that a larger set $\mathcal{R}_2$ is worse than a smaller set $\mathcal{R}_1$.

Now, we are ready to define the problem of rule generation. Let $\mathcal{R}^C = \{r_1, r_2, \ldots, r_n\}$ denote a set of candidate rules. The problem is defined as below.

Definition 3 (Rule generation) Given a set $\mathcal{R}^C$ of candidate rules, it selects a subset $\mathcal{R}^*$ that minimizes the loss, i.e., $\mathcal{R}^* = \arg_{\mathcal{R}} \min_{\mathcal{R} \subseteq \mathcal{R}^C} \Phi(\mathcal{R})$.

Labeling rule generation is very challenging as there may be many rules with diverse coverage and accuracy. Even worse, although easy to know coverage of rules, it is hard to obtain rule accuracy $P(y_i \neq L)$. To address the problem, this paper focuses on using crowdsourcing to achieve the above loss minimization, which will be presented in Sections 3, 4, and 5.

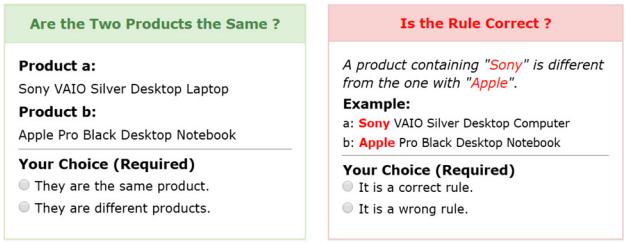
For ease of presentation, we summarize frequently used notations (some only introduced later) in Table 1.

3 Crowdsourced rule generation

This section presents an overview of rule generation using crowdsourcing. For ease of presentation, we first assume that candidate rules have already been well constructed and focus on presenting a *game-based* crowdsourcing framework for selecting high-quality rules from the candidates in Section 3.1. To solve the introduced game, we propose a *minimax crowdsourcing objective* in Section 3.2. Finally, we discuss how to construct candidate rules in Section 3.3.

3.1 Game-based crowdsourcing

The central obstacle of rule generation is that there may be many rules with *diverse* and *unknown* accuracy. A naïve crowdsourcing approach is to first evaluate each rule by sampling some covered tuples and checking them through crowdsourcing. For example, Fig. 3a shows an example crowdsourcing task for such *tuple checking*, i.e., checking whether two product records are matched. However, as crowdsourcing budget (affordable number of tasks) is usually much smaller than data size, such "aimless" checking



(a) Tuple checking task. **(b)** Rule validation task.

Fig. 3 A two-pronged crowdsourcing scheme

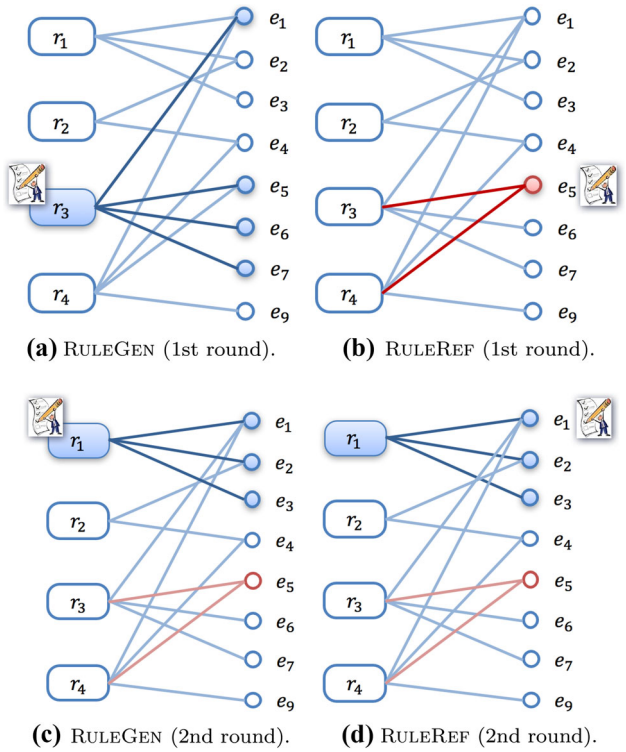
without focusing on specific rules may result in inaccurate rule evaluation, thus misleading rule selection.

Two-pronged task scheme We devise a two-pronged crowdsourcing task scheme that first leverages the crowd to directly validate a rule as a *coarse pre-evaluation* step and then applies tuple checking tasks on validated rules as a *fine post-evaluation* step. To this end, we introduce another type of task, *rule validation*. Figure 3b shows such a task to validate rule r_1 (`Sony`, `Apple`). Intuitively, human is good at understanding rules and roughly judges the validity of rules, e.g., r_1 is valid as the brand information (`Sony` and `Apple`) is useful to discriminate products. However, it turns out that rule validation tasks may produce *false positives* because the crowd may not be comprehensive enough as they usually neglect some negative cases where a rule fails. This intuition is supported by our empirical observations in Section 7.2. Thus, the fine-grained tuple checking tasks are also indispensable.

A game-based crowdsourcing approach There is usually a trade-off between these two types of tasks. On the one hand, assigning more budget on rule validation will lead to fewer tuple checking tasks, resulting in less accurate evaluation on rules. On the other hand, assigning more budget on tuple checking, although being more confident on the validated rules, may miss the chance for validating more good rules.

To effectively utilize these two types of tasks and balance their trade-off, we introduce a *game-based* crowdsourcing approach CROWDGAME, as illustrated in the central part of Fig. 2. It employs two groups of crowd workers from a crowdsourcing platform (e.g., Amazon Mechanical Turk): One group answers rule validation tasks to play a role of *rule generator* (RULEGEN), while the other answers tuple checking tasks to play a role of *rule refuter* (RULEREF). To unify these two groups, we consider that RULEGEN and RULEREF play a *two-player game* with our rule set loss $\Phi(\mathcal{R})$ in Eq. (1) as the game value function:

- RULEGEN plays as a *minimizer*: It identifies some rules $\mathcal{R}$ from candidates $\mathcal{R}^C$ for crowdsourcing via rule validation tasks and tries to minimize the loss.
- RULEREF plays as a *maximizer*: It tries to refute its opponent RULEGEN by checking some tuples that provide



(a) RuleGen (1st round). **(b)** RuleRef (1st round).

(c) RuleGen (2nd round). **(d)** RuleRef (2nd round).

Fig. 4 Example of game-based crowdsourcing

enough evidence to "reject" more rules in $\mathcal{R}$, i.e., maximizing the rule set loss $\Phi(\mathcal{R})$.

A well-known mechanism to solve such games is the *minimax strategy* in game theory: The minimizer aims to minimize the maximum possible loss made by the maximizer. Before delving into formalization of this strategy, let us consider the following intuitive example:

Example 2 Consider our example under a budget with four tasks to ask workers and $\gamma = 0.1$. We first consider a baseline rule-validation-only strategy that crowdsources rules r_1 to r_4. Suppose that all rules are validated except r_4 (as `laptop` and `notebook` are synonym), and we generate rule set $\mathcal{R}_1 = \{r_1, r_2, r_3\}$ with loss $\Phi(\mathcal{R}_1) = 3*0.1 + 2*0.9 = 2.1$ (three uncovered tuples and two error labels). Figure 4 shows how CrowdGame works, which is like a round-based board game between two players. In the first round, RuleGen selects r_3 for rule validation, as it covers four tuples and can largely reduce the first term of the loss in Eq. (1). However, its opponent RuleRef finds a "counterexample" e_5 using a tuple checking task. Based on this, RuleRef refutes both r_3 and r_4 and rejects their covered tuples to maximize the loss. Next in the second round, RuleGen selects another crowd-validated rule r_1, while RuleRef crowdsources e_1, finds e_1 is correctly labeled, and finds no "evidence" to refute r_1. As the budget is used up, we find a better rule set $\mathcal{R}_2 = \{r_1\}$ than $\mathcal{R}_1$ with a smaller loss $\Phi(\mathcal{R}_2) = 0.7$.

3.2 Formalization of minimax objective

For illustration purpose, Example 2 shows an extreme case that one counterexample is enough to refute all rules covering the tuple. However, in many applications, rules that are 100% correct may only cover a very small proportion of data. Thus, we need to tolerate some rules which are not perfect but with high "accuracy".

We first introduce the *accuracy*, denoted by λ_j, of rule r_j as the ratio of the tuples in $\mathcal{C}(r_j)$ correctly annotated with label L of r_j, i.e.:

$$\lambda_j = \frac{\sum_{e_i \in \mathcal{C}(r_j)} \mathbb{1}_{\{y_i = L\}}}{|\mathcal{C}(r_j)|}, \tag{2}$$

where $\mathbb{1}_{\{y_i=L\}}$ is an indicator function that returns 1 if $y_i = L$ or 0 otherwise. In particular, let $\Lambda^{\mathcal{R}}$ denote the accuracies of all rules in $\mathcal{R}$.

Rule accuracy λ_j is actually unknown to us, and thus, we need to estimate it using tuple checking crowdsourcing tasks. Let $\hat{\lambda}_j(\mathcal{E}_{\mathrm{q}})$ be an estimator of λ_j for rule r_j based on a set $\mathcal{E}_{\mathrm{q}}$ of tuples checked by the crowd, and $\hat{\Lambda}^{\mathcal{R}}(\mathcal{E}_{\mathrm{q}}) = \{\hat{\lambda}_j(\mathcal{E}_{\mathrm{q}})\}$ is the set of estimators, each of which is used for evaluating rule $r_j \in \mathcal{R}$.

We use $\hat{\Lambda}^{\mathcal{R}}(\mathcal{E}_{\mathrm{q}})$ to compute our overall loss defined in Eq. (1). Formally, let $\mathcal{R}^i \subseteq \mathcal{R}$ denote the set of rules in $\mathcal{R}$ covering tuple e_i, i.e., $\mathcal{R}^i = \{r_j \in \mathcal{R} | r_j(e_i) \neq \mathtt{nil}\}$. For ease of presentation, we denote $P(y_i = L)$ as $P(a_i)$ if the context is clear. Then, we introduce $\Phi(\mathcal{R}|\mathcal{E}_{\mathrm{q}})$ to denote the *estimated* loss based on a set $\mathcal{E}_{\mathrm{q}}$ of tuples checked by the crowd, i.e.:

$$\begin{aligned}\Phi(\mathcal{R}|\mathcal{E}_{\mathrm{q}}) &= \gamma(|\mathcal{E}| - |\mathcal{C}(\mathcal{R})|) + (1-\gamma) \sum_{e_i \in \mathcal{C}(\mathcal{R})} 1 - P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_{\mathrm{q}})) \\ &= \gamma|\mathcal{E}| - (1-\gamma) \sum_{e_i \in \mathcal{C}(\mathcal{R})} \{P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_{\mathrm{q}})) - \frac{1-2\gamma}{1-\gamma}\}\end{aligned} \tag{3}$$

The key in Eq. (3) is $P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_{\mathrm{q}}))$, which captures our *confidence* about whether $y_i = L$ (e_i is correctly labeled) given the observations that e_i is covered by rule $\mathcal{R}_i$ with accuracies $\Lambda^{\mathcal{R}^i}(\mathcal{E}_{\mathrm{q}})$. Next, we discuss how to compute $P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_{\mathrm{q}}))$. First, if e_i is only covered by a single rule $r_j \in \mathcal{R}^i$, we can consider e_i is sampled from Bernoulli distribution with parameter $\hat{\lambda}_j(\mathcal{E}_{\mathrm{q}})$ and thus the probability that e_i is correctly labeled is $\hat{\lambda}_j(\mathcal{E}_{\mathrm{q}})$. Second, if e_i is covered by multiple rules, the $P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_{\mathrm{q}}))$ is not easy to estimate, because rules may have arbitrary kinds of correlation. In this paper, we use a "conservative" strategy that computes $P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_{\mathrm{q}}))$ as the *maximum* rule accuracy among $\Lambda^{\mathcal{R}^i}$, i.e., $P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_{\mathrm{q}})) = \max_{r_j \in \mathcal{R}^i} \hat{\lambda}_j(\mathcal{E}_{\mathrm{q}})$. The rational is that e_i is covered by rule r_j^* with the largest accuracy, and

its $P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_q))$ is at least $\hat{\lambda}_j^*$. Consider our example in Fig. 4. Suppose that we have already estimated $\hat{\lambda}_3 = 0.5$ and $\hat{\lambda}_1 = 1$ using $\mathcal{E}_q$. Then, we compute $P(a_7|\hat{\Lambda}^{\mathcal{R}^7}(\mathcal{E}_q))$ for e_7 as 0.5, since e_7 is only covered by λ_3. We compute $P(a_1|\hat{\Lambda}^{\mathcal{R}^1}(\mathcal{E}_q))$ for e_1 as 1.0, as we know that e_1 is covered by a perfect rule r_1. We will study more complicated methods for computing $P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_q))$ in the future work.

Now, we are ready to formalize the minimax objective. Let $\mathcal{R}_q$ and $\mathcal{E}_q$, respectively, denote the sets of rules and tuples, which are selected by RuleGen and RuleRef, for crowdsourcing. Given a crowdsourcing budget constraint k on number of crowdsourcing tasks for rule generation, the minimax objective is defined as

$$\begin{aligned}
\mathcal{O}^{\mathcal{R}_q^*,\mathcal{E}_q^*} &= \min_{\mathcal{R}_q}\max_{\mathcal{E}_q} \Phi(\mathcal{R}_q|\mathcal{E}_q) \\
&\Longleftrightarrow \max_{\mathcal{R}_q}\min_{\mathcal{E}_q} \sum_{e_i\in\mathcal{C}(\mathcal{R}_q)} \left\{ P(a_i|\hat{\Lambda}^{\mathcal{R}_i}(\mathcal{E}_q)) - \frac{1-2\gamma}{1-\gamma} \right\} \\
&\Longleftrightarrow \max_{\mathcal{R}_q}\min_{\mathcal{E}_q} \sum_{e_i\in\mathcal{C}(\mathcal{R}_q)} \left\{ \max_{r_j\in\mathcal{R}^i} \hat{\lambda}_j(\mathcal{E}_q) - \frac{1-2\gamma}{1-\gamma} \right\} \qquad (4)
\end{aligned}$$

such that task numbers $|\mathcal{R}_q| + |\mathcal{E}_q| \le k$. For ease of presentation, we introduce the notation $\mathcal{J}^{\mathcal{R}_q,\mathcal{E}_q}$ where

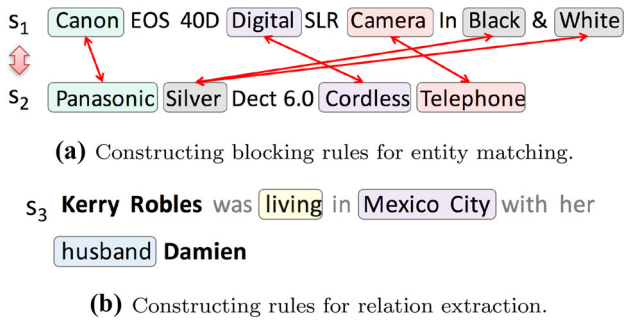
$$\mathcal{J}^{\mathcal{R}_q,\mathcal{E}_q} = \sum_{e_i\in\mathcal{C}(\mathcal{R}_q)} \left\{ \max_{r_j\in\mathcal{R}^i} \hat{\lambda}_j(\mathcal{E}_q) - \frac{1-2\gamma}{1-\gamma} \right\} \qquad (5)$$

Based on Eq. (4), we can better illustrate the intuition of CrowdGame. Overall, CrowdGame aims to find the optimal task sets $\mathcal{R}_q^*$ and $\mathcal{E}_q^*$ with constraint $|\mathcal{R}_q| + |\mathcal{E}_q| \le k$. Specifically, RuleGen would prune rules with $\hat{\lambda}_j < \frac{1-2\gamma}{1-\gamma}$ as they are useless for the maximization. Moreover, RuleGen prefers to validate rules with large coverage and high accuracy to minimize the loss. On the contrary, RuleRef aims to check tuples which are helpful to identify low-accuracy rules that cover many tuples, so as to maximize the loss. These two players iteratively select tasks until crowdsourcing budget is used up.

We highlight the challenges in the above process. The first challenge is how to select rule validation and tuple checking tasks for crowdsourcing to achieve the minimax objective. To address this challenge, we propose task selection algorithms in Sect. 4. Second, as shown in Eq. (4), the objective is based on rule accuracy estimators, e.g., $\hat{\lambda}_j(\mathcal{E}_q)$. We introduce effective estimation techniques in Sect. 5.

3.3 Candidate rules construction

This section presents our methods to create candidate rules from the raw text data for *entity matching* (EM) and *relation extraction* (RE). Note that our approach can also utilize other sources of candidate rule construction, e.g., from domain experts and using alternative algorithms. Development of more effective candidate rule generation is not the focus of this paper.



(a) Constructing blocking rules for entity matching.

(b) Constructing rules for relation extraction.

Fig. 5 Candidate rules construction

Candidate rules for entity matching The first application is entity matching for records with textual description, as shown in our running example. We want to construct candidate *blocking rules* annotating label $L_1 = -1$ to record pairs. Note that, although blocking rules are extensively studied (see a survey [7]), most of the approaches are based on structured data, and there is limited work on generating blocking rules from unstructured text. The idea of our approach is to automatically identify *keyword pairs*, which are effective to discriminate record pairs, from raw text. For example, in Fig. 5a, keyword pairs, such as (`Canon`, `Panasonic`) and (`Camera`, `Telephone`), tend to be capable of discriminating products, because it is rare that records corresponding to the same electronic product mention more than one manufacture name or product type. More precisely, we want to discover the word pair (w_a, w_b) such that any record s_a containing w_a and another record s_b containing w_b cannot be matched.

The challenge is how to *automatically* discover these "discriminating" keyword pairs. We observe that such keyword pairs usually have similar *semantics*, e.g., manufacture and product type. Based on this, we utilize word embedding techniques [27,28], which are good at capturing semantic similarity. We leverage the *word2vec* toolkit[2] to generate an embedding (i.e., a numerical vector) for each word, where words with similar semantics are also close to each other in the embedding space. Then, we identify keyword pairs from each record pair (s_a, s_b) using the Word Mover's Distance (WMD) [19]. The idea of WMD is to optimally align words from s_a to s_b, such that the distance that the embedded words of s_a "travel" to the embedded words of s_b is minimized (see [35] for more details). Figure 5a illustrates an example of using WMD to align keywords between two

[2] https://code.google.com/p/word2vec/

records, where the alignment is shown as red arrows. Using the WMD method, we identify keyword pairs from multiple record pairs and remove the ones with frequency smaller than a threshold (e.g., 10 in our experiments).

The WMD technique is also used to compute the refute probability $P(e_i^{\times})$ described in Sect. 4.2. The intuition is that refute probability captures how likely the crowd will label a tuple as matched (label $+1$) and thus refute a blocking rule. As ground truth of labels is unknown, we use the similarity between the two records in a tuple, which is measured by WMD, to estimate the probability: The more similar the records are, the more likely the crowd will label the tuple as matched. The similarity-based idea is also used in other crowdsourced entity matching works [4,44].

Candidate rules for relation extraction Relation extraction aims to discover a target relation of two entities in a sentence or a paragraph, e.g., `spouse` relation between `Kerry Robles` and `Damien` in Fig. 5b. This paper utilizes keywords around the entities as rules for labeling $+1$ (entities have the relation) or -1 (entities do not have the relation). For example, keyword `husband` can be good at identifying the `spouse` relation (i.e., labeling $+1$), while `brother` can be regarded as a rule to label -1.

We apply *distant supervision* [29], which is commonly used in relation extraction, to identify such rules, based on a small amount of known positive entity pairs and negative ones. For example, given a positive pair (`Kerry Robles`, `Damien`), we can identify the words around these entities, e.g., `living`, `Mexico City`, and `husband` (stop words like `was` and `with` are removed), as the rules labeling $+1$. Similarly, we can identify rules that label -1 from negative entity pairs. We remove the keywords with frequency smaller than a threshold (5 in our experiments) and take the remaining ones as candidate rules. One issue is how to identify some phrases, e.g., `Mexico City`. We use point-wise mutual information (PMI) discussed in [5] to decide whether two successive words, say w_i and w_j, can form a phrase. Specifically, we consider the joint probability $P(w_i, w_j)$ and marginal probabilities $P(w_i)$ and $P(w_j)$, where the probability can be computed by the relative frequency in a dataset. Then, PMI is calculated by $\log_2 \frac{P(w_i,w_j)}{P(w_i)P(w_j)}$. Intuitively, the larger the PMI is, the more likely that w_i and w_j are frequently used as a phrase. We select the phrases whose PMI scores are above a threshold (e.g., $\log_2 100$ in our experiments).

To compute refute probability $P(e_i^{\times})$, we devise the following method. Based on the small amount of positive and negative tuples mentioned above, we train a logistic regression classifier using the bag-of-words features. Given any unlabeled tuple, we extract the bag-of-words feature from it and take the output of the classifier as refute probability of the tuple.

4 Task selection algorithms

To achieve the minimax objective, we develop an *iterative crowdsourcing* algorithm, the pseudo-code of which is illustrated in Algorithm 1. Overall, it runs in iterations until $\mathcal{B}_{\mathrm{R}}$ (crowdsourcing budget) tasks are crowdsourced, where each iteration consists of a RuleGen step and a RuleRef step.

Algorithm 1: MinimaxSelect ($\mathcal{R}^{\mathrm{C}}$, $\mathcal{E}$, $\mathcal{B}_{\mathrm{R}}$, b)

Input: $\mathcal{R}^{\mathrm{C}}$: candidate rules; $\mathcal{E}$: tuples to be labeled; $\mathcal{B}_{\mathrm{R}}$: a budget; b: a crowdsourcing batch
Output: $\mathcal{R}_{\mathrm{q}}$: a set of generated rules

1 Initialize $\mathcal{R}_{\mathrm{q}} \leftarrow \emptyset$, $\mathcal{E}_{\mathrm{q}} \leftarrow \emptyset$;
2 **for** *each iteration* t **do**
  /* Rule Generator Step */
3 Select $\mathcal{R}_{\mathrm{q}}^{(t)} \leftarrow \arg_{\mathcal{R}} \max_{\mathcal{R} \subseteq \mathcal{R}^{\mathrm{C}} - \mathcal{R}_{\mathrm{q}}, |\mathcal{R}|=b} \Delta g(\mathcal{R}|\mathcal{J}^{\mathcal{R}_{\mathrm{q}},\mathcal{E}_{\mathrm{q}}})$;
4 Crowdsource $\mathcal{R}_{\mathrm{q}}^{(t)}$ as rule validation tasks ;
5 Add the crowd validated rules in $\mathcal{R}_{\mathrm{q}}^{(t)}$ into $\mathcal{R}_{\mathrm{q}}$;
6 Update objective $\mathcal{J}^{\mathcal{R}_{\mathrm{q}},\mathcal{E}_{\mathrm{q}}}$;
7 $\mathcal{R}^{\mathrm{C}} \leftarrow \mathcal{R}^{\mathrm{C}} - \mathcal{R}_{\mathrm{q}}^{(t)}$;
  /* Rule Refuter Step */
8 Select $\mathcal{E}_{\mathrm{q}}^{(t)} \leftarrow \arg_{\mathcal{E}} \min_{\mathcal{E} \in \mathcal{E} - \mathcal{E}_{\mathrm{q}}, |\mathcal{E}|=b} \Delta f(\mathcal{E}_{\mathrm{q}}|\mathcal{J}^{\mathcal{R}_{\mathrm{q}},\mathcal{E}_{\mathrm{q}}})$;
9 Crowdsource $\mathcal{E}_{\mathrm{q}}^{(t)}$ as tuple checking tasks ;
10 Add the crowd-checked $\mathcal{E}_{\mathrm{q}}^{(t)}$ into $\mathcal{E}_{\mathrm{q}}$;
11 Update accuracy $\hat{\Lambda}^{\mathcal{R}_{\mathrm{q}}}(\mathcal{E}_{\mathrm{q}})$;
12 Update budget $\mathcal{B}_{\mathrm{R}} \leftarrow \mathcal{B}_{\mathrm{R}} - 2b$;
13 **if** $\mathcal{B}_{\mathrm{R}} = 0$ **then break** ;
14 Remove rules from $\mathcal{R}_{\mathrm{q}}$ with $\hat{\lambda}_j \leq \frac{1-2\gamma}{1-\gamma}$;
15 Return $\mathcal{R}_{\mathrm{q}}$;

Rule generator step In this step, RuleGen selects rule validation tasks. Due to the latency issue of crowdsourcing [15], it is very time consuming to crowdsource tasks one by one. Thus, we apply a commonly used *batch mode* which selects b tasks and crowdsources them together, where b is a parameter. Specifically, RuleGen selects a b-sized rule set $\mathcal{R}_{\mathrm{q}}^{(t)}$ that maximizes the *rule selection criterion* denoted by $\Delta g(\mathcal{R}|\mathcal{J}^{\mathcal{R}_{\mathrm{q}},\mathcal{E}_{\mathrm{q}}})$ in the tth iteration (line 1). We will introduce the criterion $\Delta g(\mathcal{R}|\mathcal{J}^{\mathcal{R}_{\mathrm{q}},\mathcal{E}_{\mathrm{q}}})$ and present an algorithm for selecting rules based on the criterion in Sect. 4.1. After selecting $\mathcal{R}_{\mathrm{q}}^{(t)}$, RuleGen crowdsources $\mathcal{R}_{\mathrm{q}}^{(t)}$, adds the crowd-validated rules into $\mathcal{R}_{\mathrm{q}}$, and updates objective $\mathcal{J}^{\mathcal{R}_{\mathrm{q}},\mathcal{E}_{\mathrm{q}}}$. Note that we do not consider the rules failed crowd validation, because they have *much lower* accuracy than the validated ones, and incorporating them will largely increase the loss.

Rule refuter step In this step, RuleRef selects a batch of b tuple checking tasks $\mathcal{E}_{\mathrm{q}}^{(t)}$, so as to minimize the *tuple selection criterion* denoted by $\Delta f(\mathcal{E}|\mathcal{J}^{\mathcal{R}_{\mathrm{q}},\mathcal{E}_{\mathrm{q}}})$. We will discuss the criterion and a selection algorithm in Section 4.2. After obtaining the crowd answers for $\mathcal{E}_{\mathrm{q}}^{(t)}$, RuleRef adds $\mathcal{E}_{\mathrm{q}}^{(t)}$ into $\mathcal{E}_{\mathrm{q}}$ and updates the accuracy estimates $\hat{\Lambda}^{\mathcal{R}_{\mathrm{q}}}(\mathcal{E}_{\mathrm{q}})$.

For simplicity, we slightly abuse the notations to also use $\mathcal{R}$ ($\mathcal{E}$) to represent a rule set (tuple set) selected by RuleGen (RuleRef) in each iteration.

The last step of the iteration is to update budget $\mathcal{B}_R$ and check whether the algorithm terminates (i.e., the budget is used up). The algorithm continues to iteration $t+1$ if $\mathcal{B}_R > 0$. Otherwise, it "cleans up" the generated rule set $\mathcal{R}_q$ by removing bad rules with $\hat{\lambda}_j \leq \frac{1-2\gamma}{1-\gamma}$ as they are useless based on our objective (see Section 3.2), and returns $\mathcal{R}_q$ as result.

Consider the example in Fig. 4 with $\mathcal{B}_R = 4$ and $b = 1$. In the first iteration, RuleGen and RuleRef, respectively, select r_3 and e_5 for crowdsourcing. Based on the crowdsourcing answers, the algorithm updates accuracy estimates and continues to the second iteration as shown in Fig. 4c,d. After these two iterations, the budget is used up, and the algorithm returns $\mathcal{R}_q = \{r_1\}$ as the result.

4.1 Task selection for rule generator

The basic idea of task selection for RuleGen, as observed from Eq. (4), is to maximize the objective $\mathcal{J}^{\mathcal{R}_q,\mathcal{E}_q} = \sum_{e_i \in \mathcal{C}(\mathcal{R}_q)} \left\{ \max_{r_j \in \mathcal{R}^i} \hat{\lambda}_j(\mathcal{E}_q) - \frac{1-2\gamma}{1-\gamma} \right\}$, given current accuracy estimation $\hat{\Lambda}(\mathcal{E}_q)$. However, as task selection is before crowdsourcing the tasks, the essential challenge for RuleGen is that it does not know which rules will pass the crowd validation. To address this problem, we follow the existing crowdsourcing works [10,43] to consider *each possible* case of the validated rules, denoted by $\mathcal{R}^{\surd} \subseteq \mathcal{R}$, and measure the the *expected improvement* on $\mathcal{J}^{\mathcal{R}_q,\mathcal{E}_q}$ that $\mathcal{R}^{\surd}$ achieves.

Formally, let $P(\mathcal{R}^{\surd})$ denote the probability that the crowd returns $\mathcal{R}^{\surd} \subseteq \mathcal{R}$ as the validated rules, and rules in $\mathcal{R} - \mathcal{R}^{\surd}$ fail the validation. And $P(r)$ is the probability that an individual rule r passes the validation. As the rules in $\mathcal{R}$ are independently crowdsourced to the workers, we have $P(\mathcal{R}^{\surd}) = \prod_{r \in \mathcal{R}^{\surd}} P(r) \cdot \prod_{r' \in \mathcal{R} - \mathcal{R}^{\surd}} \left(1 - P(r')\right)$. For example, consider rule set $\mathcal{R}_1 = \{r_1, r_3\}$ shown in Fig. 4: There are four possible values for $\mathcal{R}_1^{\surd}$, i.e., $\emptyset$, i.e., $\{r_1\}$, $\{r_3\}$, and $\{r_1, r_3\}$. Let us also consider a simple case that the probability $P(r)$ for each rule r is $1/2$. Then, all the probabilities of the above four values are $1/4$. We will study how to adopt more effective $P(r)$ in future work.

Now, we are ready to define the *rule selection criterion*, denoted as $\Delta g(\mathcal{R}|\mathcal{J}^{\mathcal{R}_q,\mathcal{E}_q})$, as the expected improvement on our objective $\mathcal{J}^{\mathcal{R}_q,\mathcal{E}_q}$ achieved by rule set $\mathcal{R}$. For ease of presentation, we omit the superscript of $\mathcal{J}^{\mathcal{R}_q,\mathcal{E}_q}$ and simply use $\mathcal{J}$ if the context is clear. Formally, the rule selection criterion $\Delta g(\mathcal{R}|\mathcal{J})$ can be computed as:

$$\Delta g(\mathcal{R}|\mathcal{J}) = \sum_{\mathcal{R}^{\surd}} P(\mathcal{R}^{\surd}) \cdot \left(\mathcal{J}^{\mathcal{R}^{\surd} \cup \mathcal{R}_q, \mathcal{E}_q} - \mathcal{J}^{\mathcal{R}_q,\mathcal{E}_q}\right). \quad (6)$$

For instance, consider $\mathcal{R} = \{r_1\}$ in our previous example and $\mathcal{R}_q = \emptyset$. Suppose that we have estimated accuracy $\hat{\lambda}_1 = 1.0$ and let $P(r_1) = 0.5$ and $\gamma = 0.1$. Then, we have $\Delta g(\mathcal{R}|\mathcal{J}) = P(r_1) \cdot \sum_{e_i \in \mathcal{C}(r_1)} \left\{\hat{\lambda}_1 - \frac{1-2\gamma}{1-\gamma}\right\} = 0.33$.

Based on the criterion, we formalize the problem of task selection for RuleGen as follows:

Definition 4 (Task Selection for RuleGen) Given a batch size b and current objective $\mathcal{J}^{\mathcal{R}_q,\mathcal{E}_q}$, it finds b rules from remaining candidates that maximize rule selection criterion, i.e., $\mathcal{R}^* = \arg_{\mathcal{R}} \max_{\mathcal{R} \subseteq \mathcal{R}^c - \mathcal{R}_q, |\mathcal{R}|=b} \Delta g(\mathcal{R}|\mathcal{J})$.

Theorem 1 *The problem of task selection for* RuleGen *is NP-hard.*

Note that all the proofs in the paper can be found in Appendix A.

Nevertheless, although the theorem shows that obtaining the best rule set is intractable in general, we can show that the criterion $\Delta g(\mathcal{R}|\mathcal{J})$ possesses two good properties, namely monotonicity and submodularity, which enable us to develop a greedy selection strategy with theoretical guarantee. Recall that $\Delta g(\mathcal{R}|\mathcal{J})$ is monotone iff $\Delta g(\mathcal{R}_1|\mathcal{J}) \leq \Delta g(\mathcal{R}_2|\mathcal{J})$ for any sets $\mathcal{R}_1 \subseteq \mathcal{R}_2$. And $\Delta g(\mathcal{R}|\mathcal{J})$ is submodular iff $\Delta g(\mathcal{R}_1 \cup \{r\}|\mathcal{J}) - \Delta g(\mathcal{R}_1|\mathcal{J}) \geq \Delta g(\mathcal{R}_2 \cup \{r\}|\mathcal{J}) - \Delta g(\mathcal{R}_2|\mathcal{J})$ for any sets $\mathcal{R}_1 \subseteq \mathcal{R}_2$, which intuitively indicates a "diminishing returns" effect.

Lemma 1 *The rule selection criterion $\Delta g(\mathcal{R}|\mathcal{J})$ is monotone and submodular with respect to $\mathcal{R}$.*

Based on Lemma 1, we develop a greedy-based approximation algorithm. The algorithm first initializes $\mathcal{R} = \emptyset$. Then, it inserts rules into $\mathcal{R}$ based on our criterion $\Delta g(\mathcal{R}|\mathcal{J})$ in b iterations where b is the batch size of crowdsourcing. In each iteration, it finds the best rule r^* such that the margin is maximized, i.e., $r^* = \arg_{\lambda} \max \Delta g(\mathcal{R} \cup \{r\}|\mathcal{J}) - \Delta g(\mathcal{R}|\mathcal{J})$. Then, it inserts the selected r^* into $\mathcal{R}$ and continues to the next iteration. Finally, it returns the b-sized $\mathcal{R}$. Due to the monotonicity and submodularity of our selection criterion, the greedy algorithm has an approximation ratio of $1 - 1/e$ where e is the base of the natural logarithm.

Note that the computation of $\Delta g(\mathcal{R}|\mathcal{J})$ does not have to actually enumerate all the exponential cases of $\mathcal{R}^{\surd}$. In fact, given a new rule r, $\Delta g(\mathcal{R} \cup \{r\}|\mathcal{J})$ can be incrementally computed based on $\Delta g(\mathcal{R}|\mathcal{J})$.

4.2 Task selection for rule refuter

As the opponent of RuleGen, RuleRef aims to minimize $\mathcal{J}^{\mathcal{R}_q,\mathcal{E}_q}$ by checking tuples to re-estimate rule accuracy. The idea of RuleRef is illustrated in Fig. 6. Given tuple e_i, it considers two factors.

1) The first one is the *refute probability*, denoted by $P(e_i^{\times})$ that the crowd identifies e_i is not correctly labeled by rules.

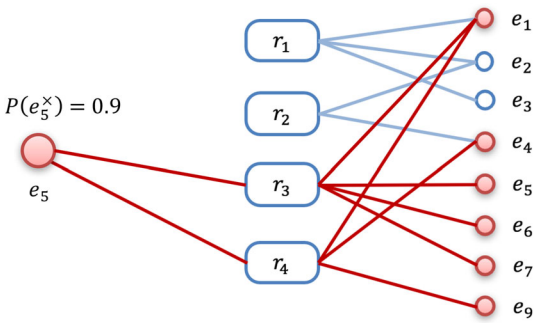


Fig. 6 Illustration of RULEREF task selection

We will discuss how to estimate refute probability later. Our intention is to identify and refute those rules covering tuples with higher refute probability and to keep those with lower refute probability. The main reason is that the lower the refute probability of a tuple is, the more likely that the tuple is correctly labeled by the rules. Take entity matching as an example, where refute probability of a tuple is estimated by the similarity between the two products in the tuple. The lower the refute probability is, the less likely that the products are similar, and the more likely that the label of -1 given by the rules is correct. Thus, those rules covering tuples with lower refute probability are more likely to be accurate. Given a set of tuples $\mathcal{E}$, we denote the subset of refuted ones as $\mathcal{E}^\times$. We assume the refute probabilities of the tuples in $\mathcal{E}$ are independent to each other, i.e.: $P(\mathcal{E}^\times) = \prod_{e_i \in \mathcal{E}^\times} P(e_i^\times) \prod_{e_i \in \mathcal{E}-\mathcal{E}^\times} 1 - P(e_i^\times)$.

2) The second factor is the *impact* of refuted tuple $e_i^\times$, denoted by $\mathcal{I}(e_i^\times)$. Suppose that refuting e_5 would lower accuracy estimates of r_3 and r_4 and thus have chance to reduce the term $\max_{r_j \in \mathcal{R}^i} \hat{\lambda}_j(\mathcal{E}_\mathrm{q}) - \frac{1-2\gamma}{1-\gamma}$ for six tuples in the objective $\mathcal{J}^{\mathcal{R}_\mathrm{q},\mathcal{E}_\mathrm{q}}$. For example, consider an extreme case that $\hat{\lambda}_3$ and $\hat{\lambda}_4$ re-estimated to 0 after checking e_5. Then, tuples e_5, e_6, e_7, and e_9 would be "eliminated" from $\mathcal{J}^{\mathcal{R}_\mathrm{q},\mathcal{E}_\mathrm{q}}$, as the maximum accuracy associated with them changes to 0. Thus, RULEREF successfully reduces the objective. Formally, we denote the amount of such "reduction" as impact $\mathcal{I}(e_i^\times)$, i.e.,

$$\begin{aligned}\mathcal{I}(e_i^\times) &= \mathcal{J}^{\mathcal{R}_\mathrm{q},\mathcal{E}_\mathrm{q}} - \mathcal{J}^{\mathcal{R}_\mathrm{q},\mathcal{E}_\mathrm{q}\cup\{e_i^\times\}} \\ &= \sum_{e_l \in \mathcal{C}(\mathcal{R}_\mathrm{q})} \max_{r_j \in \mathcal{R}^l} \left\{\hat{\lambda}_j(\mathcal{E}_\mathrm{q}) - \hat{\lambda}_j(\mathcal{E}_\mathrm{q} \cup \{e_i^\times\})\right\}. \end{aligned} \tag{7}$$

In particular, given a set $\mathcal{E}^\times$ of refuted tuples, we have $\mathcal{I}(\mathcal{E}^\times) = \sum_{e_l \in \mathcal{C}(\mathcal{R}_\mathrm{q})} \max_{r_j \in \mathcal{R}^l} \left\{\hat{\lambda}_j(\mathcal{E}_\mathrm{q}) - \hat{\lambda}_j(\mathcal{E}_\mathrm{q} \cup \mathcal{E}^\times)\right\}$.

Now, we are ready to define the *tuple selection criterion* $\Delta f(\mathcal{E}|\mathcal{J})$ using the above two factors:

$$\Delta f(\mathcal{E}|\mathcal{J}) = -\sum_{\mathcal{E}^\times} P(\mathcal{E}^\times) \cdot \mathcal{I}(\mathcal{E}^\times). \tag{8}$$

Definition 5 (Task Selection for RULEREF) Given a batch size b and current objective $\mathcal{J}^{\mathcal{R}_\mathrm{q},\mathcal{E}_\mathrm{q}}$, it finds b tuples from unchecked tuples that minimize the tuple selection criterion, i.e., $\mathcal{E}^* = \arg_{\mathcal{E}} \min_{\mathcal{E} \subseteq \mathcal{E}-\mathcal{E}_\mathrm{q}, |\mathcal{E}|=b} \Delta f(\mathcal{E}|\mathcal{J})$.

Theorem 2 *The problem of task selection for* RULEREF *is NP-hard.*

Unfortunately, RULEREF selection criterion does not have the submodularity property, which makes optimization very complex. In this paper, we utilize a greedy-based approximation algorithm that iteratively inserts e^* with the maximum margin $\sum_{\mathcal{E}^\times} P(\mathcal{E}^\times) \cdot \mathcal{I}(\mathcal{E}^\times)$ into $\mathcal{E}$ in b iterations. Moreover, similar to RULEGEN, $\Delta f(\mathcal{E}|\mathcal{J})$ can be incrementally computed without the exponential enumeration on $P(\mathcal{E}^\times)$.

Recall that we have discussed how to obtain the refute probability $P(e_i^\times)$ for entity matching and relation extraction in Section 3.3.

5 Rule accuracy estimation

The challenge in estimating rule accuracy $\hat{\lambda}_j(\mathcal{E}_\mathrm{q})$ is how to effectively utilize both rule validation and tuple checking tasks. Intuitively, we utilize rule validation tasks as "coarse pre-evaluation" and use tuple checking tasks as "fine post-evaluation". For example, consider rule r_3 : (`Black`, `Silver`) shown in Fig. 1. Suppose that r_3 successfully passed rule validation, which makes us to roughly evaluate r_3 as a good rule. However, after checking tuples covered by r_3, we find errors and thus refine the accuracy evaluation.

To formalize the intuition, we utilize the *Bayesian estimation* technique [2]. We regard crowd rule validation results as a *prior*, which captures crowd judgment on r_j without inspecting any specific tuples. As the prior may not be precise, we then use the crowd results on tuple checking as "data observation" to adjust the prior, so as to obtain a *posterior* of rule accuracy. Formally, let $p(\lambda|r^\surd, \mathcal{E}_\mathrm{q})$ denote the probability distribution of accuracy λ of rule r given the fact that r is validated by the crowd (denoted by $r^\surd$) and checked by a set $\mathcal{E}_\mathrm{q}$ of tuples. Then, following the Bayesian rule and assuming that rule validation and tuple checking results are conditionally independent given λ, we have

$$p(\lambda|r^\surd, \mathcal{E}_\mathrm{q}) = \frac{p(\mathcal{E}_\mathrm{q}|\lambda) \cdot p(\lambda|r^\surd)}{p(\mathcal{E}_\mathrm{q}|r^\surd)}, \tag{9}$$

where $p(\lambda|r^\surd)$ is the *prior* distribution of λ given that rule r has passed rule validation, $p(\mathcal{E}_\mathrm{q}|\lambda)$ is the likelihood of observing tuple checking result $\mathcal{E}_\mathrm{q}$ given accuracy λ, and $p(\lambda|r^\surd, \mathcal{E}_\mathrm{q})$ is the *posterior* distribution to be estimated. Besides, $p(\mathcal{E}_\mathrm{q}|r^\surd)$ can be regarded as a normalization factor.

Likelihood of tuple observations Recall that, given a set $\mathcal{E}_\mathrm{q}$ of checked tuples, we use $\mathcal{E}_\mathrm{q}^\times$ and $\mathcal{E}_\mathrm{q}^\surd$ to, respectively,

denote the subsets of $\mathcal{E}_q$ refuted and passed by the crowd (see Section 4.2 for more details on tuple refuting). Clearly, we have $\mathcal{E}_q^{\times} \cup \mathcal{E}_q^{\surd} = \mathcal{E}_q$ and $\mathcal{E}_q^{\times} \cap \mathcal{E}_q^{\surd} = \emptyset$. Then, given accuracy λ, we consider that $\mathcal{E}_q$ follows a *binomial distribution* with λ as its parameter, i.e.:

$$p(\mathcal{E}_q|\lambda) = \binom{|\mathcal{E}_q|}{|\mathcal{E}^{\times}|} \cdot \lambda^{|\mathcal{E}_q^{\surd}|}(1-\lambda)^{|\mathcal{E}_q^{\times}|}, \tag{10}$$

which considers all the $\binom{|\mathcal{E}_q|}{|\mathcal{E}^{\times}|}$ cases of sampling $|\mathcal{E}_q^{\surd}|$ passed and $|\mathcal{E}_q^{\times}|$ refuted tuples from $\mathcal{E}_q$.

Prior of rule validation To model the prior distribution of a rule validated by the crowd, we use the *beta distribution*, which is commonly used in Bayesian estimation for binomial distributions, i.e.:

$$p(\lambda|r^{\surd}) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\lambda^{\alpha-1}(1-\lambda)^{\beta-1}, \tag{11}$$

where $\Gamma(\cdot)$ is the gamma function (see [2] for details), and α and β are parameters of beta distribution.

As beta distribution is *conjugate* to binomial distribution, we can compute the posterior distribution as

$$p(\lambda|r^{\surd}, \mathcal{E}_q) = \frac{\Gamma(\alpha+\beta+|\mathcal{E}_q|)}{\Gamma(\alpha+|\mathcal{E}_q^{\surd}|)\Gamma(\beta+|\mathcal{E}^{\times}|)} \times \lambda^{\alpha+|\mathcal{E}_q^{\surd}|-1}(1-\lambda)^{\beta+|\mathcal{E}^{\times}|-1}$$

which is also a beta distribution with the parameters $\alpha + |\mathcal{E}_q^{\surd}|$ and $\beta + |\mathcal{E}^{\times}|$. Then, we compute the estimate $\hat{\lambda}$ as the expectation of λ to minimize the squared error:

$$\hat{\lambda}(\mathcal{E}_q) = \mathbb{E}[\lambda|r^{\surd}, \mathcal{E}_q] = \frac{\alpha + |\mathcal{E}_q^{\surd}|}{\alpha+\beta+|\mathcal{E}_q|}. \tag{12}$$

Lemma 2 *Expectation of squared error,* $\mathbb{E}[(\lambda-\hat{\lambda})^2]$*, is minimized at the estimate* $\hat{\lambda}$ *computed by Eq.* (12) [2].

Example 3 To illustrate the Bayesian estimation method, let us consider an example shown in Fig. 7 with the prior, likelihood, and posterior distributions over λ. In this example, we use a beta distribution denoted as `beta(4, 1)` with $\alpha = 4$ and $\beta = 1$. Then, we examine how the "data observation" is used to adjust the prior. When crowdsourcing three tuples and receiving two passed and one refuted answers, we can obtain the likelihood $p(\mathcal{E}_q|\lambda)$ shown as the red dotted line, and the posterior shown as the red solid line. Applying Eq. (12), we estimate $\hat{\lambda} = 0.75$. Similarly, after crowdsourcing seven tuples with four passed and three refuted answers, we obtain the curves shown as green lines and estimate $\hat{\lambda}$ as 0.67. We can see that, although both of the cases have one more passed tuple than the refuted ones, we have a lower estimate, because more refuted tuples are observed.

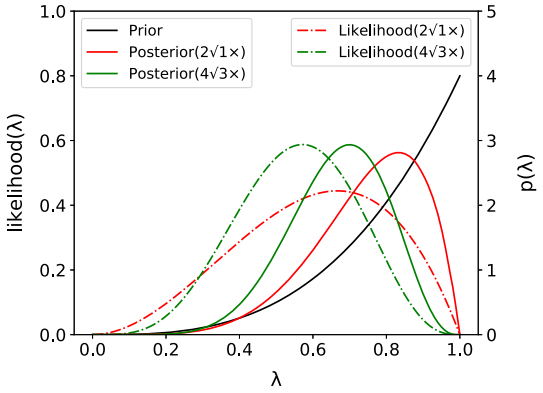


Fig. 7 Illustration of Bayesian estimation

Remarks First, we want to emphasize that the set $\mathcal{E}_q$ of checked tuples will be incrementally updated as RuleRef selects more tuple checking tasks, as illustrated in Section 4.2. From Eq. (12), we can clearly see how a refuted tuple $e_i^{\times}$ can decrease accuracy estimation: With numerator fixed and denominator added by 1, estimate $\hat{\lambda}$ becomes smaller. Second, we explain how to choose α and β. The basic idea is to sample some rules passed crowd validation and use them to estimate α and β. One simple method is to use the mean $\hat{\mu}$ and variance $\hat{\sigma}^2$ of accuracy λ calculated from the sample. Beta distribution has the following properties on statistics $\mu = \frac{\alpha}{\alpha+\beta}$ and $\sigma^2 = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$. Based on the statistics, we solve the parameters as $\alpha = (\frac{1-\hat{\mu}}{\hat{\sigma}^2} - \frac{1}{\hat{\mu}})\hat{\mu}^2$ and $\beta = \alpha(\frac{1}{\hat{\mu}} - 1)$. We can also apply more sophisticated techniques in [3] for parameter estimation.

6 Hybrid crowd-machine label assignment

After generating high-quality labeling rules via game-based crowdsourcing, CrowdGame can utilize the rules to assign labels to the tuples. It is usually the case that the rules only cover a proportion of tuples, while the ones not covered by the rules are still lacking of labels. It is trivial to solve the problem if there is no crowdsourcing budget constraint: We can simply crowdsource all the unlabeled tuples using tuple-checking tasks to obtain their labels. However, the problem is challenging given certain budget constraint, when not all the tuples can be crowdsourced for assigning labels.

To address the challenge, we study a *budget constraint label assignment* problem defined as below.

Definition 6 (Budget Constraint Label Assignment) Consider a set $\mathcal{E}^{\mathrm{U}}$ of tuples that are not labeled by the generated rules, and assume a crowdsourcing budget $\mathcal{B}_{\mathrm{L}}$ such that $\mathcal{B}_{\mathrm{L}} < |\mathcal{E}^{\mathrm{U}}|$. The

problem aims to assign labels to all the tuples in $\mathcal{E}^{\mathrm{U}}$ such that the number of tasks allowed for crowdsourcing is not greater than $\mathcal{B}_{\mathrm{L}}$.

We introduce a hybrid crowd-machine approach to solve the problem. The basic idea is to first select the most "beneficial" tuples for crowdsourcing until budget $\mathcal{B}_{\mathrm{L}}$ is used up. Then, it uses the labels to train a discriminative machine learning model for inferring labels of the un-crowdsourced tuples. There are two challenges when fulfilling the approach. The first one is how to determine what constitutes a "beneficial" tuple for crowdsourcing. We study this challenge by developing a tuple selection algorithm in Sect. 6.1. The second challenge is how to design a discriminative model which can generalize beyond the existing labels, increasing labeling coverage. We discuss two models for entity matching and relation extraction, respectively, in Sect. 6.2.

Remark Crowdsourcing budgets for rule generation (i.e., $\mathcal{B}_{\mathrm{R}}$ in Algorithm 1) and label assignment (i.e., $\mathcal{B}_{\mathrm{L}}$ in this section) can be determined by dividing a total crowdsourcing budget $\mathcal{B}$ proportionally via a ratio factor within [0, 1]. We will empirically study the effect of such ratio factor in the experiments.

6.1 Crowdsourcing tuple selection

In practice, most binary labeling applications do not treat the two labels L_1 and L_2 equally. Instead, they have their own labeling target. For example, entity matching aims to identify matched record pairs, while relation extraction focuses on entities with specified relation. In both applications, positive label ($L_2 = 1$) is more important than negative label ($L_1 = -1$). Similar applications include image recognition, outlier detection, etc. For ease of presentation, the more important label is called as *target label*, denoted by L_{T}.

Thus, the basic idea of our tuple selection strategy is to make full use of the limited crowdsourcing budget to identify as many target labels as possible. The benefit of the strategy is twofold. First, as the crowd is more accurate than a discriminative model, the strategy allows more potential target labels to be judged by the crowd, thus improving the precision/recall of the labeling results. Second, many binary labeling applications have the *class skewness* problem; for example, target (positive) labels are much fewer than the other (negative) labels. Thus, identifying more target labels would alleviate the skewness problem, which is then beneficial to the training of the discriminative model.

To determine which tuples are more likely to have target label, we design the following scoring function. Formally, given a tuple e_i, we compute its selection score, denoted by $S(e_i)$ by considering two parts.

1) RuleGen-based score $S_g(e_i)$, which measures if the generated rules in RuleGen support that e_i has target label L_{T}. Recall the notations defined in Eq. (3): The score can be estimated by $P(y_i = L_{\mathrm{T}}|\hat{\Lambda}^{\mathcal{R}^i})$, which is the probability that whether $y_i = L_{\mathrm{T}}$ given the observations that e_i is covered by rule $\mathcal{R}_i$ with accuracy $\Lambda^{\mathcal{R}^i}$. As discussed in Section 3.2, given a tuple covered by multiple rules, we use a "conservative" strategy to compute the probability of its label by considering the maximum rule accuracy, i.e.:

$$S_g(e_i) = P(y_i = L_{\mathrm{T}}|\hat{\Lambda}^{\mathcal{R}^i}) = \begin{cases} \max_{r_j \in \mathcal{R}^i} \hat{\lambda}_j & L_{\mathrm{T}} = L \\ 1 - \max_{r_j \in \mathcal{R}^i} \hat{\lambda}_j & L_{\mathrm{T}} \neq L \end{cases}$$

Example 4 Consider again our example in Fig. 4. Suppose that we have already estimated the rule accuracy as $\hat{\lambda}_3 = 0.6$ and $\hat{\lambda}_1 = 0.8$. As both rules annotate, label $L_1 = -1$, which is not the target label $L_2 = 1$ in entity matching. We thus estimate the score for tuple e_1, which is covered by rules r_1 and r_3, as $S_g(e_1) = 1 - \max\{\hat{\lambda}_3, \hat{\lambda}_1\} = 0.2$.

2) RuleRef-based score $S_f(e_i)$, which measures if RuleRef supports that e_i has target label L_{T}. In particular, if $L_{\mathrm{T}} \neq L$, which is usually the case in entity matching, this score is essentially the refute probability $P(e_i^{\times})$, defined in Section 3.3. Moreover, as discussed in Section 3.3, this probability is, respectively, estimated as the similarity between records in entity matching and the output of a logistic regression classifier in relation extraction. On the other hand, if $L_{\mathrm{T}} = L$, then $S_f(e_i)$ can be trivially computed by $1 - P(e_i^{\times})$. For instance, considering tuple e_5 in Fig. 6, its score can be computed as $S_f(e_i) = P(e_5^{\times}) = 0.9$.

Overall, we compute the tuple section score $S(e_i)$ by averaging $S_g(e_i)$ and $S_f(e_i)$, which are, respectively, determined by RuleGen and RuleRef. Based on our empirical study, we will see the consideration of both sources of scores is beneficial in Section 7.

Algorithm 2: TupleSelect ($\mathcal{R}_{\mathrm{Q}}$, $\mathcal{E}_{\mathrm{Q}}$, $\mathcal{E}^{\mathrm{U}}$, $\mathcal{B}_{\mathrm{L}}$, b)

Input: $\mathcal{R}_{\mathrm{Q}}$: generated rules; $\mathcal{E}^{\mathrm{U}}$: unlabeled tuples;
$\mathcal{B}_{\mathrm{L}}$: budget for label assignment;
b: a crowdsourcing batch

Output: $\mathcal{E}_{\mathrm{l}}$: a set of labled tuples

```
Initialize E_l ← ∅ ;
for each iteration i do
    Sort tuples in E^U based on function S(e_i) ;
    Crowdsource the top b tuples, denoted by E_l^(i) via tuple checking tasks ;
    Add the crowd-checked tuples E_l^(i) into E_l ;
    E^U ← E^U − E_l^(i) ;
    Update rule accuracy estimation ;
    Update budget B_L ← B_L − b ;
    if B_L = 0 then break ;
Return E_l ;
```

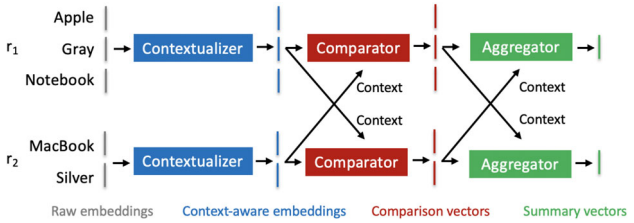
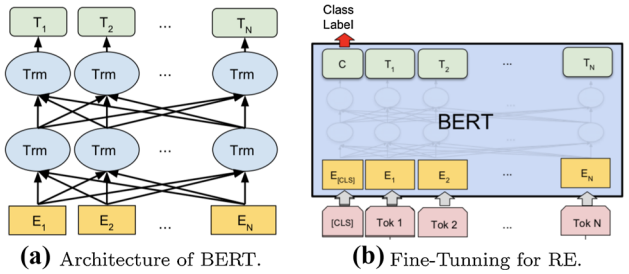


Fig. 8 Utilizing DeepMatcher for entity matching



(a) Architecture of BERT. **(b)** Fine-Tunning for RE.

Fig. 9 Utilizing BERT for relation extraction

Next, we present an algorithm of crowdsourcing tuple selection based on the scoring function introduced above. One issue is that the rule accuracy estimation is adjusted as more tuples are crowdsourced, which would impact the RULEGEN-based scores. To address the issue, we develop an iterative crowdsourcing tuple selection method, the pseudo-code of which is shown in Algorithm 2. Overall, it runs in iterations until $\mathcal{B}_L$ tasks are crowdsourced. It applies a batch mode of selecting b tasks in each iteration, which is similar to Algorithm 1. In each iteration, it selects the tuples according to our scoring function $S(e_i)$ and updates rule accuracy estimation after crowd answers are collected.

6.2 Discriminative models for inference

After rule generation and tuple selection, no more budget can be used for crowdsourcing. If there still exist unlabeled tuples, we then use a discriminative machine learning model, which can learn to generalize beyond the existing labels. We train the model on the existing labeled tuples from crowdsourcing and use the model to further increasing coverage of data labeling. In our work, we employ the state-of-the-art ML models for our two applications, i.e., DeepMatcher [31] for entity matching and BERT [7] for relation extraction. Next, we give a brief description of the models as they are not the focus of the paper, and we encourage the interested readers to refer to the original papers.

DeepMatcher for entity matching We adopt the *hybrid* mode of DeepMatcher, which achieves the best performance on entity matching. As shown in Fig. 8, it takes as input two records, say r_1 and r_2, and uses the following modules to produce a matching probability. First, a *Contextualizer* module is introduced to model the context of each word in a record, which is beneficial for disambiguation; for example, "`Apple`" in r_1 is more like a manufacture due to its context. It takes as input raw word embeddings (gray lines in the figure) and uses a bidirectional GRU model to output a context-aware embedding sequence (blue lines). Then, a *Comparator* module "aligns" each word in a record, say r_1, to the words in r_2 using an attention mechanism. For example, the alignment for `notebook` would attend more to `macbook` than `silver`. Based on the attention weights, it uses a two-layer Highway Net with ReLU to generate a comparison vector for the word (red lines), by a weighted combination of the aligned words. Next, an *Aggregator* module aggregates the word comparison vectors in each record to produce a single summary vector (green lines), by using a bidirectional RNN with attention mechanism. Finally, DeepMatcher uses a two-layer fully connected ReLU Highway Net to implement a classifier over the two summary vectors, for producing the matching probability.

BERT for relation extraction The basic idea is to first utilize BERT [7] to *pretrain* language representation models on our dataset, as BERT achieves state-of-the-art results for many NLP tasks. Then, we use the labels for relation extraction to *fine-tune* the model.

BERT Pretraining is a joint training of the following two tasks: (1) masked language model: BERT masks out 15% of the words in the input randomly and runs a multilayer bidirectional transformer encoder that predicts the masked words to learn better bidirectional representations. (2) Next sentence prediction: To learn relationships between sentences, BERT performs a task that predicts whether a sentence is the next sentence that comes after another one. The BERT architecture used in our work is shown in Fig. 9a. We adopt the architecture of $\text{BERT}_{\text{LARGE}}$ with parameters L=24, H=1024, and A, where L, H, and A are, respectively, the numbers of layers, hidden states, and self-attention heads.

For fine-tuning, we use the final hidden state of a sentence to be extracted, shown as token [CLS] $C \in R^h$ in Fig. 9b as the representation of the sentence. We feed the hidden state to a softmax layer to output the predicted label, e.g., whether or not a spouse relation can be extracted. Then, we can train the entire model by minimizing a loss defined on the predicted and ground-truth labels.

7 Experiments

This section evaluates the performance of our approach. We first evaluate different task selection strategies for crowdsourced rule generation. Then, we compare our approach

Table 2 Statistics of datasets and crowd answers

	Abt-Buy	Ama-Goo	Ebay	Spouse
# +1 tuples	1090	1273	2057	424
# −1 tuples	227,715	179,525	107,847	5493
# cand-rules	16,344	15,157	13,903	360
Rule labels	−1	−1	−1	−1, 1
Crowd accuracy on tuple checking	95.61%	93.53%	99.87%	99.05%

with the state-of-the-art methods in two real-world applications: entity matching and relation extraction. In addition, we also evaluate how parameter settings, such as crowdsourcing batch size b and weight γ in loss, affect the performance.

7.1 Experiment setup

Datasets We use standard datasets, which are commonly used in the existing works, to evaluate the approaches. Table 2 shows the statistics of the datasets. For entity matching, we evaluate the approaches on three real datasets. 1) `Abt-Buy` contains electronics product records from two Web sites: Abt and BestBuy. We regard each tuple e_i as a pair of records with one from Abt and the other from BestBuy, where each record has a text description as illustrated in Fig. 1. Following the existing works in entity matching [4,43], we prune the pairs with similarity smaller than a threshold 0.3 (we use WMD [19] to measure similarity) and obtain 1090 tuples with label 1 (`matched`) and 227, 715 tuples with label −1 (`unmatched`). 2) `Ama-Goo` contains software products from two Web sites: Amazon and Google. Similar to `Abt-Buy`, we obtain 1273 tuples with label 1 and 179, 525 tuples with label −1. 3) `Ebay` contains beauty products collected from Web site Ebay. Using the above method, we, respectively, obtain 2057 and 107, 847 tuples with labels 1 and −1. For these three datasets, we construct candidate rules, which only annotate the −1 label for discriminating records. Statistics of candidate rules are also found in Table 2.

For relation extraction, we use a `Spouse` dataset to identify whether two persons in a sentence have spouse relation. The `Spouse` dataset contains 2591 news articles[3]. We segment each article into sentences and identify entities mentioned in the sentences. We consider each tuple as a pair of entities occurring in the same sentence and obtain 424 tuples with label 1 (entities have spouse relation) and 5493 tuples with label −1 (entities do not have spouse relation). We construct candidate rules and obtain 360 rules. Note the rules on this dataset can annotate both 1 (e.g., `husband`) and −1 (e.g., `brother`) labels. Ground truth of each of the above datasets is already included in the original dataset.

[3] http://research.signalmedia.co/newsir16/signal-dataset.html

Crowdsourcing on AMT We use Amazon Mechanical Turk (AMT, https://www.mturk.com/) as the platform for publishing crowdsourcing tasks. Examples of the two task types are shown in Fig. 3.

For fair comparison, we crowdsource all candidate rules for crowd validation to collect worker answers, so as to *run different strategies on the same crowd answers*. For tuple checking on the `Spouse` dataset, we also ask the crowd to check all the tuples. For the EM datasets, we crowdsource all the +1 tuples for collecting crowd answers. Nevertheless, as there are a huge number of −1 tuples, we use a sampling-based method. We sample 5% of the −1 tuples for each dataset to estimate the crowd accuracy on tuple checking (Table 2). Then, for the rest of the −1 tuples, we use the estimated accuracy to simulate the crowd answers: Given a tuple, we simulate its crowd answer as its ground truth with the probability equals to the accuracy, and the opposite otherwise. We use a batch mode to put ten tasks in an HIT, and spend $0.01 for each HIT. We use a qualification test to only allow workers with at least 150 approved HITs and 95% approval rate. We do not use more strict qualification (e.g., more approved HITs or Master workers), because, on one hand, our worker qualification is sufficient for providing reliable crowdsourcing results after majority voting, as shown in Table 2. One the other hand, we have a large number of crowdsourcing tasks in our experiments. Thus, although more strict worker qualification would further improve the accuracy, it will cost us much more money and take much longer time.

Parameter settings First, γ is the weight balancing quality and coverage in our loss function in Eq. (1). Due to the label skewness in entity matching as observed in Table 2, we set $\gamma = 0.001$ to prefer quality over coverage. In a similar way, we set $\gamma = 0.1$ for relation extraction. Second, parameters α and β of beta distribution can be set based on our discussion in Section 5. We use (350, 1) for entity matching and (4, 1) for relation extraction. Third, batch size b of RuleGen/RuleRef (Algorithm 1) is set to 20.

7.2 Empirical observations of candidate rules

This section provides observations for the quality of initial (candidate) rules constructed by our method in Section 3.3.

Table 3 Ratios of perfect ($\lambda = 1$) versus imperfect ($\lambda < 1$) candidate rules constructed on the four datasets

Datasets	Ratio	
	Perfect rule	Imperfect rule
Abt-Buy	0.333	0.667
Ama-Goo	0.427	0.573
Ebay	0.433	0.567
Spouse	0.105	0.895

We report the ratios of perfect rules (accuracy 1.0) and imperfect rules (accuracy < 1.0) in initial rule sets on our datasets. As observed from Table 3, the result shows that there are less perfect rules than the imperfect rules. This demonstrates that it is not easy to identify high-quality rules from noisy candidates.

We then investigate the crowd performance on rule validation tasks. We compute the real accuracy of each rule based on true labels of tuples covered by the rule. Then, for the set of rules passed (failed) crowdsourcing validation, we use a histogram to show the *accuracy distribution*, where the x-axis is values of accuracy and y-axis is the percentage of rules with the corresponding accuracy values. Note that this figure shows accuracy distribution of rules which *pass/fail the workers' validation*, where the denominator is not the number of rules in candidate set. Figure 10 shows the distributions on our four datasets. We have the following observations: First, most of the crowd-validated rules have high accuracies. For example, about 90% of crowd-validated rules have perfect accuracy 1.0 on datasets Ama-Goo and Ebay. In contrast, accuracies of rules that failed the validation are diverse and thus unreliable. Second, although capable of identifying good rules, crowd validation may introduce *false positives*. For example, on the Spouse dataset, there are some less precise rules that also passed the validation. This is because crowd may not be comprehensive enough as they usually neglect some negative cases where a rule fails. This motivates us to use rule validation task as a coarse rule evaluation and tuple checking tasks as fine rule evaluation, so as to eliminate the false positives.

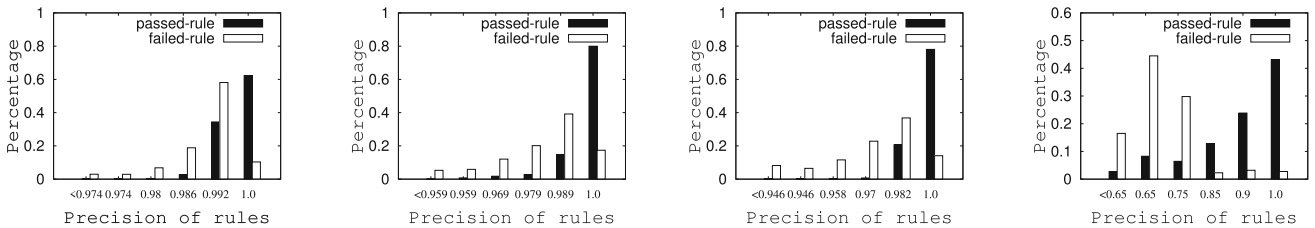
We also observe that values of rule accuracy on the entity matching datasets are larger than those on the relation datasets. This is because labels on entity matching datasets are very skew. However, a minor difference on rule accuracy may have significant effects on the final F_1 score, because there are very few positives. This increases the challenge of high-quality rule generation.

7.3 Evaluation on minimax crowdsourcing

This section evaluates the minimax crowdsourcing objective and task selection algorithms. We compare different alternative task selection strategies, by varying crowdsourcing budget $\mathcal{B}_R$ for rule generation.

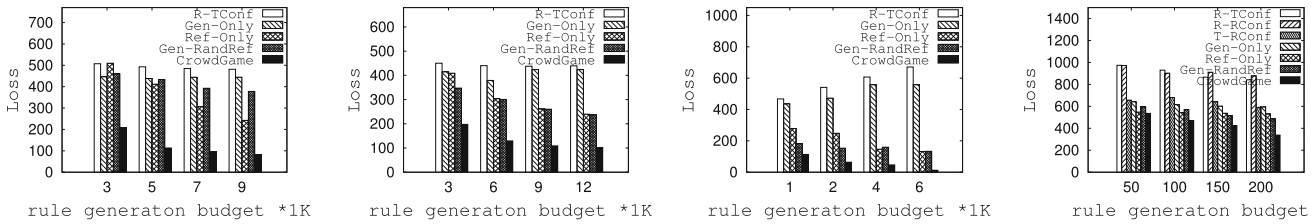
- Gen-Only only utilizes rule validation tasks and uses the prior as accuracy estimates, instead of using tuple checking tasks for refinement. Then, it utilizes the criterion of RuleGen for task selection.
- Ref-Only only utilizes tuple checking tasks. As there is no rule validation tasks, in each iteration, it selects a batch of rules that maximize the coverage, and assumes that they have passed the validation. Then, it utilizes the criterion of RuleRef for selecting tuple checking tasks for crowdsourcing.
- Gen-RandRef considers both components: RuleGen and RuleRef. However, the RuleRef uses a random strategy to select tuples for checking.
- CrowdGame is our game-based approach.

We also compare with simpler *conflict*-based heuristics. T-TConf selects the rules covering the largest number of "conflicting" tuple. As tuples labels are unknown, we consider a tuple is conflicting with a rule if its refute probability is larger than threshold 0.5. For relation extraction where conflicting rules exist, we also use another two baselines. T-RConf selects the rules that have the largest conflicts with *other rules*. Given a rule, it counts the number of tuples covered by the rule which are also annotated by other rule(s) with a conflicting label. We select the rules with the largest such numbers. T-RConf selects tuples that have the largest conflicting labels from the rules covering the tuples.



(a) Abt-Buy dataset. **(b)** Ama-Goo dataset. **(c)** Ebay dataset. **(d)** Spouse dataset.

Fig. 10 Accuracy distributions of crowdsourced rules passed/failed rule validation.



(a) Abt-Buy dataset. **(b)** Ama-Goo dataset. **(c)** Ebay dataset. **(d)** Spouse dataset.

Fig. 11 Evaluating game-based crowdsourcing with different strategies

Figure 11 shows the experimental results. Conflict-based heuristics R-TConf and R-RConf perform the worst, because the selected rules are with more conflicts and tend to cover tuples with opposite true labels. These rules may be either invalidated by the crowd or be selected to incur more errors and larger overall loss. T-RConf performs better than R-TConf and R-RConf, because tuples covered by conflicting rules can be used to refute some "bad" rules. However, it cannot beat our methods in the framework of Algorithm 1, as it may not find tuples with the largest refuting impact.

Gen-Only achieves the inferior performance, because, without tuple checking, the selection criterion used in RuleGen is to essentially identify rule with large coverage. However, without refuting false positive ones, rules with large coverage are more harmful as they tend to induce more errors. Ref-Only performs better with the increase of budget k. For example, the loss decreases from 509 to 242 as the budget increases from 3000 to 9000 on the Abt-Buy dataset. This is because more checked tuples lead to better accuracy estimation and thus facilitate to refute bad rules. Moreover, Ref-Only is in general better than Gen-Only. Gen-RandRef is a straightforward approach that combines RuleGen and RuleRef. It can reduce loss in some cases, which shows the superiority of combining rule validation and tuple checking. However, it only achieves limited loss reduction, and it is sometimes even worse than Ref-Only (Fig. 11a). This is because a random refuter strategy may not be able to find the rules with the largest impact and thus performs weak to refute bad rules.

CrowdGame achieves the best performance. For example, the loss achieved by CrowdGame is an order of magnitude smaller than that of the alternatives on the Ebay dataset. This significant improvement is achieved by the minimax objective formalized in the game-based crowdsourcing, where RuleGen can find good rules, while RuleRef refutes bad rules in a two-player game. Moreover, our task selection algorithm can effectively select tasks to fulfill the minimax objective. We may observe that, on the Spouse dataset, CrowdGame has little improvement compared to Gen-RandRef when the budget is large. This is because the number of candidate rules is small on this dataset (i.e., 360 as shown in Table 2). Under such circumstance, checking a large number of tuples may also be enough to identify good rules.

7.4 Comparisons for entity matching (EM)

This section evaluates how CrowdGame boosts entity matching and compares with state-of-the-art approaches. Note that it considers the setting of no budget constraint in label assignment. Specifically, recall our two-phase framework introduced in Section 2.1: Phase I uses some crowd budget $\mathcal{B}_R$ for generating blocking rules, and Phase II applies the rules and crowdsources tuple checking tasks until all the tuples are labeled.

For evaluation, In Phase I, we measure *rule coverage* as the ratio of tuples covered by the rules. We also examine the extent of "errors" incurred by the rules using *false negative* (FN) rate, which is the ratio of true matches "killed off" by the generated rules. Intuitively, rule generation in Phase I performs well if it has large coverage and low FN rate. In Phase II, we measure the performance using *precision* (the number of correct matches divided the number of returned ones), *recall* (the number of correct matches divided the number of all true matches), and F_1 score ($\frac{2 \cdot precision \cdot recall}{precision + recall}$). We also measure the total crowdsourcing cost in EM, including the rule generation crowd budget $\mathcal{B}_R$ in Phase I and the number of pair-based tasks in Phase II.

As shown in Table 4, increasing rule generation budget can improve both quality and cost. For instance, on the Abt-Buy dataset, with the increase in rule generation budget from 3000 to 9000, the coverage of the generated rules improves from 0.764 to 0.924, while the FN rate remains at a very low level. This validates that CrowdGame can select high coverage rules while incurring insignificant errors. Moreover, this can also effectively boost the overall EM process. The total cost is reduced from 61, 739 to 26, 381 due to larger rule coverage. The precision improves from 0.927 to 0.969. This is because more high-quality rules are selected to correct the crowd errors in tuple checking (e.g., some workers misjudge unmatched pairs with matched ones). On the other hand, more budget for RuleRef can identity more bad rules (especially those with large coverage) and thus reduces false positive rules to improve recall.

Table 4 Using CROWDGAME for entity matching (EM)

Dataset	Rule gen crowd budget	Phase I		Phase II				Total crowd cost (Phases I & II)
		Rule coverage	FN rate	Precision	Recall	F_1	Crowd cost	
Abt-Buy	3000	0.764	0.083	0.927	0.916	0.921	58,739	61,739
	5000	0.867	0.037	0.942	0.928	0.935	34,189	39,189
	7000	0.898	0.033	0.960	0.955	0.957	24,269	31,269
	9000	0.924	0.028	0.969	0.957	0.963	17,381	26,381
Ama-Goo	3000	0.528	0.003	0.925	0.996	0.959	89,671	92,671
	6000	0.697	0.002	0.947	0.998	0.972	57,685	63,685
	9000	0.767	0.002	0.959	0.998	0.978	43,864	52,864
	12,000	0.799	0.002	0.966	0.997	0.981	36,115	48,115
Ebay	1000	0.504	0.005	0.995	0.969	0.982	33,321	34,321
	2000	0.785	0.004	0.995	0.985	0.990	18,761	20,761
	4000	0.902	0.004	0.999	0.988	0.993	5292	9292
	6000	0.966	0.003	1.000	0.996	0.998	1410	7410

We compare CROWDGAME with state-of-the-art approaches, where we set rule generation budget to 9000, 12,000, and 6000 on the three datasets, respectively. In our experiments, all the approaches have the same termination condition that all the tuples are labeled. We first compare CROWDGAME with the crowdsourced EM approaches: `Trans` [44], `PartOrder` [4], and `ACD` [46]. We get source codes of these approaches from the authors. Note that these baselines do not consider labeling rules. Instead, they select some "representative" tuples (record pairs) for crowdsourcing and use *tuple-level* inference, such as transitivity [44,46] and partial order [4]. As shown in Table 5, CROWDGAME significantly reduces the total crowdsourcing cost over `Trans` and `ACD`, nearly by an order of magnitude. This shows that rules generated by CROWDGAME are much more powerful than the transitivity to prune unmatched pairs. For quality, `Trans` may "amplify" crowd errors through transitivity. `ACD` addresses this issue by using adaptive task selection. CROWDGAME also outperforms `Trans` and `ACD` on F_1 score, since it utilizes the game-based framework with minimax objective to optimize the quality. Second, although `PartOrder` achieves much less total cost, its F_1 is very low, e.g., 0.486 on the `Ama-Goo` dataset. This is because `PartOrder` utilizes the partial order among tuples determined by similarity between records. Although performing well on structured data, `PartOrder` has inferior performance on our datasets, because textual similarity is very unreliable for such inference.

We also compare CROWDGAME with `Snorkel` (with the crowdsourcing setting described in [32]). `Snorkel` first relies on crowdsourcing (as well as labeling functions) to label some tuples and then uses these labels to train a discriminative ML model to label all the remaining ones. As a result, we need to determine the size of the training set, i.e., the crowdsourcing budget, for the discriminative ML model. For fair comparison, we set up two experimental settings for `Snorkel`, which are denoted as $\texttt{Snorkel}^1$ and $\texttt{Snorkel}^2$, respectively. $\texttt{Snorkel}^1$ uses the same total crowdsourcing cost of CROWDGAME as budget, e.g., 26, 381 on the `Abt-Buy` dataset, to obtain labels from the crowd, which makes sure that the two approaches rely the same human (crowd) efforts. Second, $\texttt{Snorkel}^2$ examines how much crowdsourcing cost it needs to achieve similar results with CROWDGAME. It continues to collect crowdsourcing

Table 5 Comparison with EM methods

Dataset	Method	$F1$ of EM	Total crowd cost
Abt-Buy	Trans	0.864	203,715
	PartOrder	0	1063
	ACD	0.887	216,025
	$\texttt{Snorkel}^1$	0.909	26,381
	$\texttt{Snorkel}^2$	0.961	86,720
	CROWDGAME	0.963	26,381
Ama-Goo	Trans	0.896	158,525
	PartOrder	0.486	763
	ACD	0.919	167,958
	$\texttt{Snorkel}^1$	0.923	48,115
	$\texttt{Snorkel}^2$	0.98	149,540
	CROWDGAME	0.982	48,115
Ebay	Trans	0.971	50,163
	PartOrder	0.553	170
	ACD	0.998	57,637
	$\texttt{Snorkel}^1$	0.857	7410
	$\texttt{Snorkel}^2$	0.997	39,920
	CROWDGAME	0.998	7410

Table 6 Using CROWDGAME for relation extraction on the `Spouse` dataset

Rule gen crowd budget	Phase I			Phase II				Total Crowd Cost (Phases I & II)
	Rule coverage	FN rate	FP rate	Precision	Recall	F_1	Crowd cost	
50	0.587	0.019	0.734	0.504	0.747	0.602	2227	2277
100	0.687	0.027	0.537	0.545	0.645	0.591	1686	1786
150	0.719	0.026	0.453	0.585	0.640	0.611	1511	1661
200	0.695	0.027	0.149	0.810	0.635	0.712	1643	1843

labels in batches, where 20 tasks are put in each batch, until the difference between F_1 obtained by `Snorkel`[2] and CROWDGAME is very small (i.e., ≤ 0.002). Another issue is which tuples should be selected for crowdsourcing in `Snorkel`. We select the tuples with higher pairwise similarity measured by WMD, in order to obtain a tuple set with more balanced labels. Specifically, due to label skewness of EM datasets, random tuple selection may end up with very rare $+1$ tuples selected, which is not good for model training in `Snorkel`. In contrast, the selection by similarity will increase the chance of finding $+1$ tuples.

As shown in Table 5, the experimental results show that, under the same crowdsourcing cost, CROWDGAME outperforms `Snorkel`[1] on quality, e.g., achieving $6-15\%$ improvements on F_1. This quality boost is because of the high-quality rules identified by CROWDGAME, which label a large number of tuples with high accuracy. To achieve similar labeling quality, CROWDGAME can save the cost by up to 81% compared with `Snorkel`[2]. This shows that the rules generated by CROWDGAME can reduce the cost more effectively.

7.5 Comparison for relation extraction (RE)

This section evaluates CROWDGAME for RE on the `Spouse` dataset. Note that this section also considers the setting of no crowdsourcing budget constraint in label assignment. We construct candidate rules using the method in Section 3.3. Different from CROWDGAME for EM that only considers $L_1 = -1$ rules, CROWDGAME for relation extraction generates both $L_1 = -1$ and $L_2 = 1$ rules in Phase I. Thus, besides FN rate, we introduce *false positive* (FP) rate (the ratio of FP over all negatives) to measure the "errors" incurred by $L_2 = 1$ rules in Phase I. Then, Phase II exploits crowdsourcing to check all the entity pairs not covered by the rules.

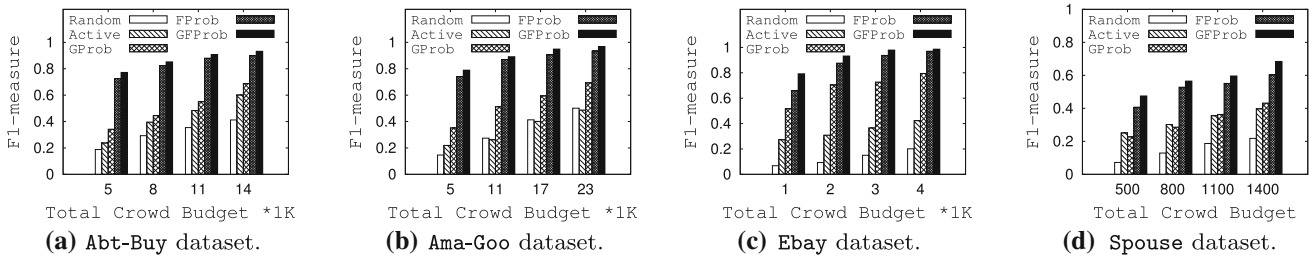
Table 6 shows the performance of CROWDGAME. With the increase in the rule generation budget, the total crowd cost is largely reduced because rule coverage is improved from 0.587 to 0.695. One interesting observation is that, when increasing the budget from 150 to 200, the precision is improved from 0.585 to 0.810 while rule coverage and recall slightly decrease. This is because RULEREF is able to identify and refute more low-accuracy rules and thus significantly reduces false positives for relation extraction.

Table 7 Comparison with `Snorkel` in RE

Method	Precision	Recall	F_1
`Snorkel (ManRule)`	0.389	0.608	0.474
`Snorkel (ManRule+Crowd)`	0.519	0.696	0.595
CROWDGAME	0.81	0.635	0.712

We compare CROWDGAME with `Snorkel` for relation extraction. Note that `Snorkel` is fed with labeling functions designed by users in our experiments, which are denoted by *manual rules* in our experiments. We consider the following two settings of `Snorkel`. First, we feed a set of manual rules provided by the original paper [32] to `Snorkel`, which consist of the following two kinds: 1) some keywords summarized by domain experts, such as "wife" (labeling $+1$), "ex-husband" (labeling $+1$), and "father" (labeling -1) and 2) a set of 6126 entity pairs with `spouse` relation extracted from an external knowledge base, DBPedia[4]. Second, we take both these manual rules and crowd labels as labeling functions. Note that we use the similarity-based method (same to the EM scenario) to select tuples for crowdsourcing to obtain tuples with more balanced classes, and the number of selected tuples is the same to the total crowd cost of CROWDGAME, e.g., the same crowd cost 1843 for both `Snorkel` and CROWDGAME. As observed from Table 7, `Snorkel` with only manual rules achieves inferior quality. The reason is that the manual rules are based on some generally summarized keywords and external knowledge, which are not specifically designed for the `Spouse` dataset. Moreover, further considering crowd labels, `Snorkel` achieves better precision and recall, as it can learn better ML models due to the additional crowd efforts. However, CROWDGAME still achieves the best performance by a margin of 0.11 on F_1, at the same crowd cost. This is because our approach can identify high-quality rules from the candidates; especially, the refuter can effectively eliminate error-prone rules, thus resulting in superior precision.

[4] http://wiki.dbpedia.org/



(a) Abt-Buy dataset. **(b)** Ama-Goo dataset. **(c)** Ebay dataset. **(d)** Spouse dataset.

Fig. 12 Evaluation tuple extraction strategies for label assignment under various total crowd budgets

7.6 Comparisons under total budget constraints

This section evaluates the setting that a budget constraint $\mathcal{B}$ on the total number of crowdsourcing tasks is given by the end user. Under this setting, the budget is divided proportionally into rule generation budget $\mathcal{B}_{\mathrm{R}}$ and label assignment budget $\mathcal{B}_{\mathrm{L}}$. We consider the default ratio between $\mathcal{B}_{\mathrm{R}}$ and $\mathcal{B}_{\mathrm{L}}$ over $\mathcal{B}$ as 0.3 in this section and will study the effect of the ratio later.

Evaluation on tuple selection. We evaluate our proposed crowdsourcing tuple selection method presented in Sect. 6.1 and compare the following alternatives:

- `Random` selects unlabeled tuples randomly.
- `Active` utilizes deep active learning with the least confidence strategy: It selects the most uncertain tuples with respect to the discriminative model, by using entropy to measure the uncertainty.
- `FProb` uses our strategy of identifying target labels, but it only considers RuleRef-based score.
- `GProb` also aims at identifying target labels, but considering only RuleGen-based score.
- `GFProb` takes both RuleGen-based and RuleRef-based scores into consideration for tuple selection.

Figure 12 shows the experimental results. `Active` is more effective than the naïve `Random`, because `Active` can find more appropriate tuples for better training of the discriminative model. However, both methods cannot achieve satisfactory results, since neither of them takes identifying tuples with target label as the criterion. As analyzed in Sect. 6.1, as the crowd is more accurate than the model, it is better to use the crowd to find as many target labels as possible, while letting the model play a complementary role. Among the three alternatives in our criterion, `GFProb` is better than `FProb` and `GProb`, which validates that the combination of RuleGen-based and RuleRef-based scores is more effective in identifying tuples with target label.

Comparisons for entity matching. We evaluate the approaches under crowdsourcing budget constraints in entity matching. As the crowdsourced EM approaches: `Trans` [44], `PartOrder` [4], and `ACD` [46], cannot support the budget constraint setting, we only compare with `Snorkel`. For both CrowdGame and `Snorkel`, we first use the total budget $\mathcal{B}$ for crowdsourced data labeling[5] and then apply DeepMatcher for label generalization. We evaluate the labeling quality (precision, recall, and F_1 score) in both of the two steps.

As observed in Table 8, CrowdGame outperforms `Snorkel` under each budget constraint on all datasets. This is mainly attributed to the following reasons. First, CrowdGame can identify much more matched tuples, i.e., achieving higher recall, after crowdsourcing, due to our proposed techniques of rule generation and tuple section. This essentially relieves the burden of the downstreaming DeepMatcher model. Second, CrowdGame can train a better DeepMatcher model to further increase the recall, without much loss of precision. This is because that CrowdGame provides a better training set for DeepMatcher, e.g., with more matched record pairs.

Comparisons for relation extraction. We evaluate the approaches under crowdsourcing budget constraints in relation extraction and compare with `Snorkel`. Table 9 shows the experimental results. Similar to the case in entity matching, CrowdGame can effectively utilize BERT to increase the recall without much loss in the precision, leading to an improvement of the overall F_1 score. Moreover, CrowdGame also outperforms `Snorkel` under various budget constraints.

7.7 Effect of parameter settings

This section evaluates the effect of parameters used in CrowdGame. Specifically, we examine four parameters: (1) the batch size b of crowdsourcing tasks in each iteration of Algorithm 1, (2) the weight γ that balances coverage and quality in our loss function in Eq. (1), (3) the proportion of budget allocation over rule generation ($\mathcal{B}_{\mathrm{R}}$) and tuple selection ($\mathcal{B}_{\mathrm{L}}$), and 4) the effect of refute probability $P(e_i^{\times})$ in RuleRef step.

[5] See how `Snorkel` uses crowdsourcing in Section 7.4

Table 8 Method comparisons under various crowdsourcing budget constraints in entity matching

Dataset	Total crowd budget	CrowdGame						Snorkel		
		After Crowdsourcing			After DeepMatcher					
		Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1
Abt-Buy	5000	0.984	0.582	0.731	0.863	0.699	0.772	0.982	0.57	0.721
	8000	0.982	0.714	0.827	0.909	0.801	0.852	0.976	0.678	0.800
	11,000	0.979	0.808	0.885	0.912	0.903	0.907	0.97	0.728	0.832
	14,000	0.975	0.879	0.925	0.94	0.925	0.932	0.965	0.773	0.858
Ama-Goo	5000	0.991	0.622	0.764	0.947	0.676	0.789	0.986	0.609	0.753
	11,000	0.983	0.799	0.882	0.951	0.839	0.891	0.977	0.782	0.869
	17,000	0.977	0.877	0.924	0.943	0.956	0.949	0.968	0.838	0.898
	23,000	0.973	0.925	0.925	0.94	0.925	0.932	0.965	0.773	0.858
Ebay	1000	1.000	0.522	0.686	0.932	0.688	0.792	1.000	0.525	0.689
	2000	1.000	0.829	0.907	0.959	0.907	0.932	0.999	0.768	0.868
	3000	0.999	0.944	0.971	0.976	0.982	0.979	0.997	0.856	0.921
	4000	0.997	0.970	0.983	0.988	0.985	0.986	0.997	0.897	0.944

Table 9 Method comparison under various crowdsourcing budget constraints in relation extraction

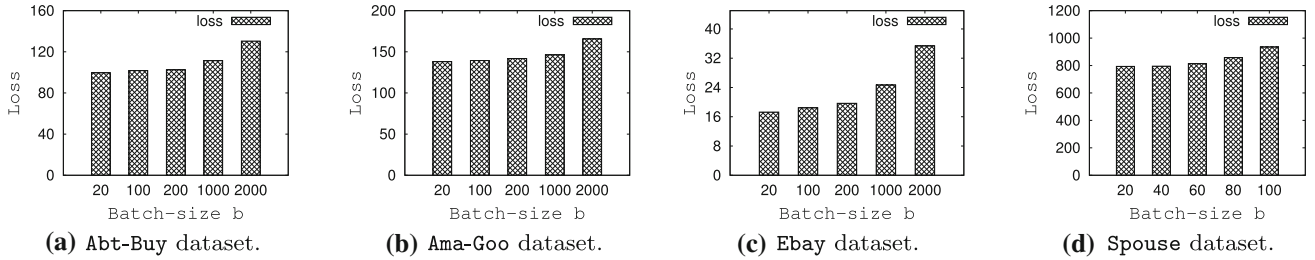
Total crowd budget	CrowdGame						Snorkel		
	After Crowdsourcing			After BERT					
	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1
200	0.575	0.268	0.366	0.529	0.431	0.475	0.298	0.507	0.375
500	0.603	0.316	0.415	0.569	0.561	0.565	0.379	0.582	0.459
800	0.568	0.55	0.559	0.521	0.696	0.596	0.402	0.603	0.482
1100	0.803	0.574	0.669	0.735	0.639	0.684	0.488	0.656	0.560

Batch size b We first examine the effect of batch size b in each iteration. We vary the batch size on each dataset and evaluate the loss. Note that we set the budgets as the maximum budgets of the datasets shown in Fig. 11. Figure 13 shows the experimental results. We can see that, with the increase in b, the loss also increases. The main reason is the followings. In CrowdGame, RuleGen and RuleRef iteratively crowdsource tasks with the goal of competing with each other and "adjust" their decisions based on the opponents' decisions. For example, given some rules selected with RuleGen, RuleRef decides to select tuples to refute the rules, which will further adjust RuleGen's later decisions on selecting more rules. Given a fixed budget, a larger batch size b means less "interactions" between RuleGen and RuleRef. Thus, RuleGen and RuleRef may not be able to adjust their decisions in time to minimize the overall loss. On the other hand, a very small batch size b would lead to high crowdsourcing latency (i.e., waiting time). Based on the experimental results, we suggest to set b to a reasonable number (e.g., $20-200$ in the figures) that balances loss and crowdsourcing latency.

Weight γ **in loss** Then, we examine the weight γ in the loss function. Observed from Eq. (5), $(1-2\gamma)/(1-\gamma)$ plays a *threshold* role that any rules with $\hat{\lambda}_j < (1-2\gamma)/(1-\gamma)$ will be pruned from RuleGen, as they are useless to minimize the loss. Thus, for each dataset, we vary $(1-2\gamma)/(1-\gamma)$ in the values: 0.999, 0.8, 0.6, 0.4, and 0.2, which result in the following γ values: 0.001, 0.167, 0.286, 0.375, and 0.444. Then, on each dataset, we plot coverage and error number under each of the γ values.

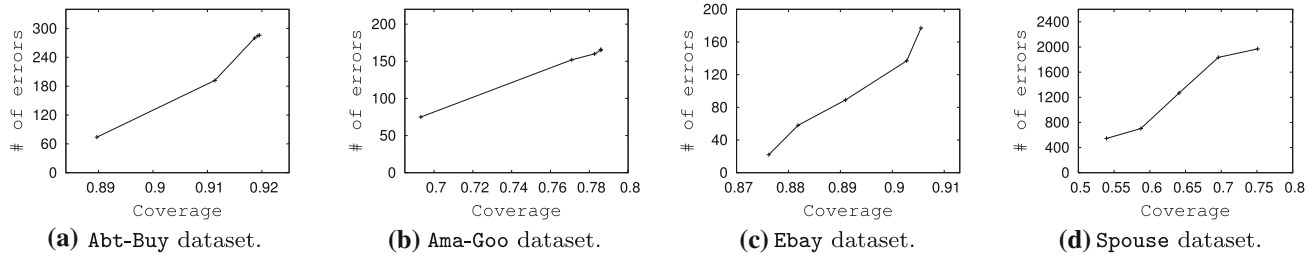
Figure 14 illustrates the results. With the increase in γ, the coverage increases, while the number of errors also increases. For example, when increasing γ from 0.001 to 0.167 (i.e., changing $(1-2\gamma)/(1-\gamma)$ from 0.999 to 0.8), the coverage increases from 0.890 to 0.912, while the number of errors also increases from 60 to 180, as shown in Fig. 14a. This is because the larger the γ is, the more preferable the coverage is and the less preferable the accuracy is. In particular, a larger γ will lead to a smaller threshold $\frac{1-2\gamma}{1-\gamma}$ on rule accuracy. Thus, the accuracy will degrade. As mentioned in Section 2, setting an appropriate γ depends on the targeted applications.

Budget allocation. We evaluate the budget allocation over rule generation and tuple selection. To this end, we vary the ratio of $\mathcal{B}_R$ over the total budget $\mathcal{B}$ from 0.1 to 0.9 and examine the labeling quality (before discriminative model). As shown in Fig. 15, with the increase in the ratio, the F_1 score increases



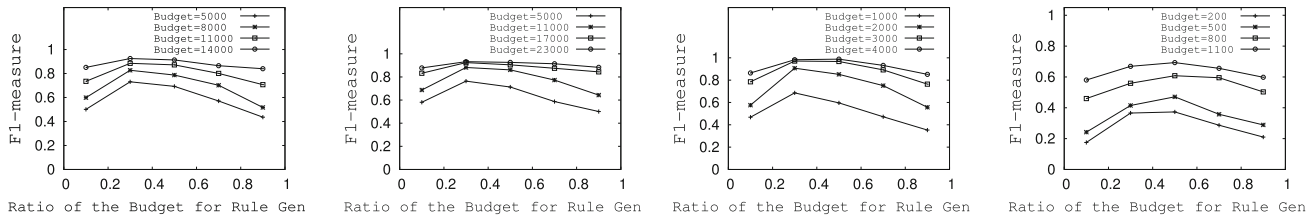
(a) Abt-Buy dataset. **(b)** Ama-Goo dataset. **(c)** Ebay dataset. **(d)** Spouse dataset.

Fig. 13 Effect of batch size b with respect to overall loss



(a) Abt-Buy dataset. **(b)** Ama-Goo dataset. **(c)** Ebay dataset. **(d)** Spouse dataset.

Fig. 14 Effect of weight γ on balancing quality and coverage of rule generation



(a) Abt-Buy dataset. **(b)** Ama-Goo dataset. **(c)** Ebay dataset. **(d)** Spouse dataset.

Fig. 15 Effect of the ratio of rule generation budget $\mathcal{B}_R$ over the total crowdsourcing budget $\mathcal{B}$

first and then decreases. This is attributed to the following reasons. On the one hand, when less budget is allocated to rule generation, CROWDGAME cannot generate enough labeling rules, resulting in a large number of unlabeled tuples. On the other hand, when the ratio is too high, the budget for selecting tuples with target label is insufficient. This would make the downstreaming discriminative model difficult to achieve high performance. Another observation is that as the total budget increases, the effect of the ratio on F_1 becomes less significant. This shows that CROWDGAME becomes more robust given sufficient budget.

Refute probability $P(e_i^{\times})$ We also empirically examine the effect of considering refute probability $P(e_i^{\times})$. We compare with a baseline without considering $P(e_i^{\times})$. As shown in Table 10, the experimental result validates the importance of refute probability $P(e_i^{\times})$. For example, by considering the probability on the Abt-Buy dataset, the loss can be reduced by about four times. The reason is that, without considering the refute probability, the selected tuples, although may have large impact, are less likely to be voted by the crowd as conflicting with the rules. Thus, such tuples are less useful to refute the rules in the RULEREF step.

Table 10 Effect of considering refute probability $P(e_i^{\times})$

Dataset	Rule gen crowd budget	Loss With $P(e_i^{\times})$	Loss Without $P(e_i^{\times})$
Abt-Buy	3000	208.6	452.8
	5000	113.1	414.4
	7000	95.2	390.4
	9000	82.3	389.8
Ama-Goo	3000	197.6	393.3
	6000	129.2	269.0
	9000	108.8	239.2
	12,000	102.0	232.8
Ebay	1000	113.3	170.4
	2000	63.6	152.9
	4000	46.5	158.6
	6000	12.5	160.4
Spouse	50	556.0	567.3
	100	470.0	524.5
	150	425.0	511.0
	200	338.0	491.6

8 Related work

Crowdsourced data labeling Recently, crowdsourcing has been extensively studied for harnessing the crowd intelligence. There is a large body of works on crowdsourcing (see a recent survey [23]), such as quality control [9,25,40,49,50,50] and crowd DB systems [11,12,22,26,31,39]. This paper pays special attention on *crowdsourced data labeling*, which acquires relatively low cost labeled data in a short time using crowdsourcing, with the focus on reviewing such works in entity matching and relation extraction. Crowdsourced entity matching (aka. crowdsourced entity resolution) [4,6,8,13,18,21,41–43,46,47] has been extensively studied recently. Existing works have studied many aspects in the field, including task generation [43], transitivity-based inference [4,42,44], partial-order inference [4], and task selection [41]. However, most of them only label tuples (i.e., record pairs) and do not consider generating *labeling rules* for reducing total crowd cost. One exception is the hands-off crowdsourcing approach [6,13]. However, the approach generates blocking rules on structured data using random forest, and the method cannot be applied to text data studied in our approach. Crowdsourcing is also applied in relation extraction [1,24]. However, similar to entity matching, most of the works focus on tuple-level labeling.

Weak-supervision labeling rules There are many works in the machine learning community to label large training sets using weak-supervision labeling rules. A well-known example is distant supervision [16,29,34,38], where the training sets are created with the aid of external resource such as knowledge bases. The distant supervision sources are usually noisy. To alleviate this problem, [34,38] label data with hand-specified dependency generative models. [16] uses multi-instance learning models to denoise different sources. When gold labels are not available, some methods estimate potential class labels based on noisy observations, e.g., spectral methods [30] and generative probabilistic models [17,48]. Some approaches are recently proposed to consolidate noisy or even contradictory rules [32,36]. Some works demonstrate that the proper use of weak-supervision rules can also boost the performance of deep learning methods [33]. Our approach and these works focus on different aspects of data labeling: They focus on "consolidating" *given* labeling rules (functions), while we pay more attention to generating high-quality rules. To this end, we leverage game-based crowdsourcing to select high-quality rules with large coverage and accuracy, which results in performance superiority shown in our experiments.

Generative Adversarial Networks The recent Generative Adversarial Network (GAN) also applies a minimax framework for training neural networks and has been widely applied in image and text processing [14,45]. Our approach is different from GAN in the following aspects. First of all, CROWDGAME uses the minimax framework to combine two types of tasks for data labeling, while GAN focuses on parameter learnings. Second, GAN uses algorithms such as stochastic gradient descent to optimize the parameters. In contrast, the optimization of CROWDGAME is rule/tuple task selection. Third, CROWDGAME needs to consider cost of crowdsourcing, which is not a concern of GAN.

9 Conclusion and future work

In this paper, we have studied the data labeling problem. We introduced *labeling rules* to reduce labeling cost while preserving high quality. We devised a crowdsourcing approach to generate rules with high coverage and accuracy. We developed a game-based crowdsourcing framework that employs a group of workers that answers rule validation tasks to play a role of *rule generator*, and another group that answers tuple checking tasks to play a role of *rule refuter*. We proposed a minimax optimization method to unify rule generator and rule refuter in a two-player game. We conducted experiments on entity matching and relation extraction to show performance superiority of our approach.

One important future work is to explore more broad data labeling scenarios. The key challenge is how to improve the effectiveness of crowd workers in rule validation. We argue that it mainly depends on the interpretability of machine-generated candidate rules, that is, whether the rules are easily understandable by the crowd. For example, the textual patterns for relation extraction are easily understandable, and our experiments have shown that workers are effective in validating them. In contrast, some domains or labeling tasks may construct rules which are not intuitive to the crowd. For instance, some schema matching works [10] construct instance similarity-based rules. Obviously, workers may find it difficult to judge which similarity function is appropriate or what threshold is effective.

Acknowledgements This work was supported by NSF of China (61632016, 61925205, U1711261, 61832017, 61972401, 61932001), Huawei, TAL education, Beijing Outstanding Young Scientist Program NO. BJJWZYJH012019100020098, Research Funds of Renmin University of China (18XNLG18, 18XNLG21), and the Fundamental Research Funds for the Central Universities.

A Proofs

A.1 Proof of Theorem 1

We can prove NP-hardness of the problem by a reduction from the k maximum coverage (KMC) problem, which is known to be NP-hard.

Recall that an instance of the KMC problem (E,$\mathcal{S}$,k) consists of a universe of elements E = $\{s_1, s_2, \cdots, s_n\}$, a collection of subsets of the universe E, i.e., $\mathcal{S} = \{S_1, S_2, \cdots, S_m\}$ where any $S_i \in \mathcal{S}$ satisfies $S_i \subseteq E$, and a number k. The objective is to select k subsets from $\mathcal{S}$, denoted by $\mathcal{S}'$, so that the number of covered tuples $\left|\bigcup_{S \in \mathcal{S}'} S\right|$ is maximized.

An instance of our problem consists of a set of tuples $\mathcal{E}$, a set of rules $\mathcal{R}$, and a number b. The optimization objective is to select b rules from $\mathcal{R}$ so that the expected rule selection criterion, according to Eq. 6, is maximized.

The reduction from KMC to our problem. We show next that for any instance (E,$\mathcal{S}$,k) of KMC, we can create a corresponding instance of our problem based on (E,$\mathcal{S}$,k) in polynomial time.

- We translate the set E of elements into the set $\mathcal{E} = \{e_1, e_2, \cdots, e_n\}$ of tuples in our problem.
- Given an element s_j in E, if $s_j \in S_i$, we set add a tuple e_j into the rule r_i whose accuracy and validation probability is 1. We set the parameter γ to 0.5. The gain of objective $\mathcal{J}^{r_i}$ calculates to 1 if $s_j \in S_i$, and 0 otherwise. Thus, each set S_i in the KMC problem corresponds to the rule r_i and the elements covered by S_i correspond to the tuples covered by r_i.
- We translate number k in KMC into b in our problem.

Equivalence of optimization objectives. We show the optimization objectives of the two problems are equivalent:

- Since in our instance the probability that an individual rule r passes the validation is 1, the validated rule set $\mathcal{R}^{\surd}$ is equivalent to the selected rule set $\mathcal{R}_q^{(t)}$, and $P(\mathcal{R}^{\surd}) = P(\mathcal{R}_q^{(t)}) = 1$.
- Since in our instance the rule accuracy is 1 and the parameter γ is 0.5, based on Equation 4, we know that $\mathcal{J}^{\mathcal{R}_q^{(t)}} = \left|\mathcal{C}(\mathcal{R}_q^{(t)})\right|$.

With $\mathcal{R}^{\surd} = \emptyset$, the expected rule selection criterion therefore becomes $\left|\mathcal{C}(\mathcal{R}_q^{(t)})\right|$. Since our problem is to find the b best rules, $\mathcal{R}_q^{(t)}$, that maximizes the expected criterion, this is equivalent to finding b best sets that maximize the set of covered elements.

A.2 Proof of Lemma 1

Consider two rule sets $\mathcal{R}_1 \subseteq \mathcal{R}_2$; we first prove the monotonicity as follows. For simplicity, we use Γ to denote $\frac{1-2\gamma}{1-\gamma}$ in this proof:

$$\Delta g(\mathcal{R}_2|\mathcal{J}) - \Delta g(\mathcal{R}_1|\mathcal{J}) = \sum_{\mathcal{R}_2^{\surd}} P(\mathcal{R}_2^{\surd}) \sum_{e_i} \max_{r_j} \hat{\lambda}_j - \Gamma - \sum_{\mathcal{R}_1^{\surd}} P(\mathcal{R}_1^{\surd}) \sum_{e_i} \max_{r_j} \hat{\lambda}_j - \Gamma$$

Since $\mathcal{R}_1 \subseteq \mathcal{R}_2$, for simplicity, we introduce $\mathcal{R}_3 = \mathcal{R}_2 - \mathcal{R}_1$. Then, for any $\mathcal{R}_2^{\surd}$, we can find a $\mathcal{R}_1^{\surd}$ and $\mathcal{R}_3^{\surd}$ such that $\mathcal{R}_2^{\surd} = \mathcal{R}_1^{\surd} \cup \mathcal{R}_3^{\surd}$. Based on this, we have

$$\Delta g(\mathcal{R}_2|\mathcal{J}) - \Delta g(\mathcal{R}_1|\mathcal{J}) = \sum_{\mathcal{R}_1^{\surd}} P(\mathcal{R}_1^{\surd}) \Big[\sum_{\mathcal{R}_3^{\surd}} P(\mathcal{R}_3^{\surd}) \Big(\sum_{e_i \in \mathcal{C}(\mathcal{R}_1^{\surd} \cup \mathcal{R}_3^{\surd})} \{\max_{r_j} \hat{\lambda}_j - \Gamma\} - \sum_{e_i \in \mathcal{C}(\mathcal{R}_1^{\surd})} \{\max_{r_j} \hat{\lambda}_j - \Gamma\} \Big) \Big].$$

It is not difficult to know $\sum_{e_i \in \mathcal{C}(\mathcal{R}_1^{\surd} \cup \mathcal{R}_3^{\surd})} \{\max_{r_j} \hat{\lambda}_j - \Gamma\} - \sum_{e_i \in \mathcal{C}(\mathcal{R}_1^{\surd})} \{\max_{r_j} \hat{\lambda}_j - \Gamma\} \geq 0$, and we prove monotonicity.

We next prove that $\Delta g(\mathcal{R}|\mathcal{J})$ is submodular. Given any rule r, using the previous equation, we have

$$\begin{aligned} \Delta g(\mathcal{R} \cup \{r\}|\mathcal{J}) - \Delta g(\mathcal{R}|\mathcal{J}) &= \sum_{\mathcal{R}^{\surd}} P(\mathcal{R}^{\surd}) P(r^{\surd}) \Big(\sum_{e_i \in \mathcal{C}(\mathcal{R}^{\surd} \cup \{r^{\surd}\})} \{\max_{r_j} \hat{\lambda}_j - \Gamma\} - \sum_{e_i \in \mathcal{C}(\mathcal{R}^{\surd})} \{\max_{r_j} \hat{\lambda}_j - \Gamma\} \Big) \\ &= -\Gamma + \sum_{\mathcal{R}^{\surd}} P(\mathcal{R}^{\surd}) P(r^{\surd}) \Big(\sum_{e_i \in \mathcal{C}(r^{\surd}) - \mathcal{C}(\mathcal{R}^{\surd})} \hat{\lambda} \\ &+ \sum_{e_i \in \mathcal{C}(r^{\surd}) \cap \mathcal{C}(\mathcal{R}^{\surd})} \max\{\hat{\lambda} - \max \Lambda^{\mathcal{R}^{(i)}}, 0\} \Big) \end{aligned} \tag{13}$$

From the equation, we can see that the above margin depends on the following two factors under each cases corresponding to $P(\mathcal{R}^{\surd})P(r^{\surd})$:

- Improvement on "additional" tuples covered by $r^{\surd}$, i.e., $\sum_{e_i \in \mathcal{C}(r^{\surd}) - \mathcal{C}(\mathcal{R}^{\surd})} \hat{\lambda}$.
- Improvement on the tuples already covered by $\mathcal{R}^{\surd}$.

Now, let us consider a rule set $\mathcal{R}_1 \subseteq \mathcal{R}_2$. It is not difficult to see that both of the above factors corresponding to $\mathcal{R}_2$ will not be greater than that of $\mathcal{R}_1$. Thus, we have $\Delta g(\mathcal{R}_1 \cup \{r\}|\mathcal{J}) - \Delta g(\mathcal{R}_1|\mathcal{J}) \geq \Delta g(\mathcal{R}_2 \cup \{r\}|\mathcal{J}) - \Delta g(\mathcal{R}_2|\mathcal{J})$, which proves the submodularity. Hence, we prove the lemma.

A.3 Proof of Theorem 2

To prove Theorem 2, let us consider a special case of the RuleRef task selection problem, as shown in Fig. 16. Each rule has the same accuracy $\hat{\lambda}_j = \lambda$, and each tuple has the

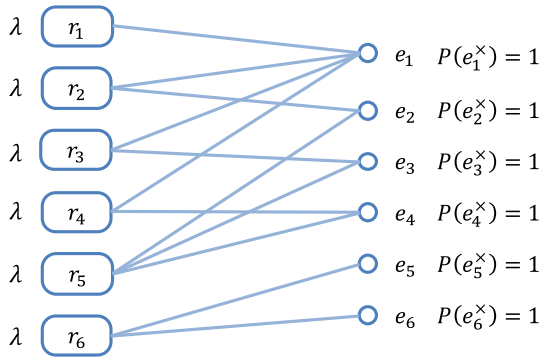


Fig. 16 Illustration of Theorem 2 proof

same refute probability $P(e_i^\times) = 1.0$. Moreover, we consider the "strict" refuting strategy used in Example 2: One counterexample is enough to refute all rules covering the tuple. And we consider the weight $\gamma = 0.5$. In this case, refuting a tuple, say e_1, will remove all the rules covering the tuple, say $\{r_1, r_2, r_3\}$. However, the removed rules cannot induce any impact defined in Section 4.2, as the tuples covered by $\{r_1, r_2, r_3\}$ are still covered by other un-refuted rules, and thus the maximum accuracy associated with these tuples is still λ. Suppose that we refute e_5, and then, we have an impact λ as maximum rule accuracy associated with e_6 becomes 0. Based on these examples, it is not difficult to see this special case of RuleRef task selection problem is equivalent to the following *maximum isolated node* problem:

Definition 7 (Maximum Isolated Node Problem) Given a bipartite graph over a rule node set $\mathcal{R}$ and a tuple node set $\mathcal{E}$, consider the following removal conditions: (1) If a tuple node is removed, then all the rule nodes connected to the tuple node as well as the edges associated with the rule nodes are removed; (2) a tuple node is called "isolated node" iff there is no edge associated with the tuple node. The problem finds k tuple nodes $\mathcal{E}' \subseteq \mathcal{E}$ such that the number of isolated nodes after the removal is maximized.

For example, in Fig. 16, after removing $\{e_2, e_3\}$, there is no isolated tuple nodes. On the contrary, after removing $\{e_1, e_2\}$, e_3 and e_4 become isolated tuple nodes.

We can prove the maximum isolated node problem is NP-hard by a reduction from the minimum vertex cover (MVC) problem, which is known as NP-hard. Recall that an instance of the MVC problem consists of a graph $G' = (V, E)$ of vertex set V and edge set E. The problem aims to find the minimum vertex subset $V' \subseteq V$ such that every edge $e \in E$ has at least one endpoint in V'.

Next, we show the reduction from the MVC problem to our maximum isolated node problem. Given any instance of the MVC problem $G' = (V, E)$, we create a tuple node set $\mathcal{E}$, each of which corresponds to a vertex in V, and a rule node set $\mathcal{R}$, each of which corresponds to an edge in E.

Then, suppose that our maximum isolated node problem is solved; given any number k, we can find a subset $\mathcal{E}' \subseteq \mathcal{E}$ of tuple nodes that the number of isolated nodes is maximized. So we can vary k from 1 to $|\mathcal{E}|$ to find the minimum k that satisfies all nodes in $\mathcal{E} - \mathcal{E}'$ are isolated. Given the above reduction, we can see that this actually solves the MVC problem, because isolating all tuple nodes is equivalent to find a vertex subset V' that covers all edge E in the MVC problem.

Thus, we prove that the maximum isolated node problem can be solved only if the MVC problem is solved. As the MVC problem is NP-hard, the maximum isolated node problem is NP-hard. Moreover, since the maximum isolated node problem is a special case of our RuleRef task selection problem formalized in Definition 5, we prove Theorem 2.

B Examples of Labeling Rules

We also provide some examples to better understand the intuition behind our method. Table 11 shows some high-quality rules validated by the crowd on four datasets. Take the rule (`Sony`, `Toshiba`) on the `Abt-Buy` dataset as an example: We can observe that applying a rule is equivalent to annotating over 2000 samples. Selecting these high-quality rules forms the basis for CrowdGame. For EM tasks, such good rules usually contain brand names, product names, the product functions, properties, abbreviations, and so on. For

Table 11 Examples of crowd-validated rules

Dataset	Example rule	Coverage
`Abt-Buy`	(Sony, Toshiba)	2,050
	(Canon, Samsung)	1,319
	(Camera, Vaio)	1,028
	(Player, TV)	909
	(Camera, Headphone)	770
`Ama-Goo`	(Macintosh, Windows)	907
	(Mac, Vista)	789
	(Adobe, Office)	769
	(Mac, Sony)	689
	(Microsoft, Photoshop)	568
`Ebay`	(Face, Lip)	1,166
	(Dior, NYX)	761
	(Blush, Cream)	742
	(Cream, Liner)	685
	(Lipstick, Powder)	628
Spouse	Mama (−1 rule)	33
	Dad (−1 rule)	27
	Lover (+1 rule)	26
	Rival (−1 rule)	26
	Assistant (−1 rule)	17

spouse relation dataset, the good rules usually consist of words related with kinship.

C Extension of Labeling Rule

We discuss a more general case that some rules in the candidates $\mathcal{R}^{\mathrm{C}}$ annotate label $L_1 = -1$ (called L_1 rules for simplicity), while others annotate $L_2 = 1$ (called L_2 rules). Consider our `spouse` relation extraction example that annotates $L_2 = 1$ if entities have spouse relation or $L_1 = -1$ otherwise. In this case, a tuple, e.g., entity pair (`Michelle Obama`, `Barack Obama`), could be covered by conflicting rules (textual patterns), e.g., a L_2 rule "`married with`" and a L_1 rule "`meets`."

CrowdGame devises a simple extension from Algorithm 1 by taking L_1 and L_2 rules *independently*. More specifically, let $\mathcal{R}_{\mathrm{q}}^{L_1}$ ($\mathcal{R}_{\mathrm{q}}^{L_2}$) denote the set of L_1 (L_2) rules selected by RuleGen for crowdsourcing. Recall that $\mathcal{E}_{\mathrm{q}}$ is the set of tuples selected by RuleRef for crowdsourcing. First, we extend the overall minimax optimization objective, denoted by $\tilde{\mathcal{J}}$, as a combination of objectives of L_1 and L_2 rules, i.e., $\tilde{\mathcal{J}} = \mathcal{J}^{\mathcal{R}_{\mathrm{q}}^{L_1},\mathcal{E}_{\mathrm{q}}} + \mathcal{J}^{\mathcal{R}_{\mathrm{q}}^{L_2},\mathcal{E}_{\mathrm{q}}}$, where $\mathcal{J}^{\mathcal{R}_{\mathrm{q}}^{L_1},\mathcal{E}_{\mathrm{q}}}$ ($\mathcal{J}^{\mathcal{R}_{\mathrm{q}}^{L_2},\mathcal{E}_{\mathrm{q}}}$) is defined in Eq. (5). Then, we run the iterative crowdsourcing framework in Algorithm 1. We present how RuleGen and RuleRef work in each iteration as follows:

- RuleGen only slightly extends the computation of rule selection criterion $\Delta g(\mathcal{R}|\mathcal{J})$ as the summation of 1) the expected improvement of L_1 rules $\mathcal{R}^{L_1}$ in $\mathcal{R}$ over $\mathcal{J}^{\mathcal{R}_{\mathrm{q}}^{L_1},\mathcal{E}_{\mathrm{q}}}$ and 2) the expected improvement of $\mathcal{R}^{L_2}$ over $\mathcal{J}^{\mathcal{R}_{\mathrm{q}}^{L_2},\mathcal{E}_{\mathrm{q}}}$, where the expected expectation is computed using Eq. (6). Then, RuleGen uses the greedy strategy to find an optimal rule set $\mathcal{R}^*$ that maximizes the criterion $\Delta g(\mathcal{R}|\mathcal{J})$.
- RuleRef extends the notation of $e_i^{\times}$ to $e_i^{L_1}$ (or $e_i^{L_2}$), which, respectively, means tuple e_i is checked and annotated with L_1 (or L_2). Then, given a checked tuple $e_i^{L_1}$ (or $e_i^{L_2}$), RuleRef considers it to refute the L_2 part (or the L_1 part) of objective $\tilde{\mathcal{J}}$ using Eq. (7). Based on this, given a tuple set $\mathcal{E}$, we consider every possible case of $(\mathcal{E}^{L_1}, \mathcal{E}^{L_2})$ where $\mathcal{E}^{L_1} \cup \mathcal{E}^{L_2} = \mathcal{E}$ and $\mathcal{E}^{L_1} \cap \mathcal{E}^{L_2} = \emptyset$, and revise Eq. (8) to $\Delta f(\mathcal{E}|\mathcal{J}) = -\sum_{\mathcal{E}^{L_1},\mathcal{E}^{L_2}} P(\mathcal{E}^{L_1})P(\mathcal{E}^{L_2}) \cdot (\mathcal{I}(\mathcal{E}^{L_1}) + \mathcal{I}(\mathcal{E}^{L_2}))$. Then, RuleRef utilizes this criterion for selecting tuples.

Using the above method, CrowdGame obtains a rule set $\mathcal{R}_{\mathrm{q}}$ returned by Algorithm 1. Then, let us use $\mathcal{R}_{\mathrm{q}}^i \subseteq \mathcal{R}_{\mathrm{q}}$ as the set of rules covering a tuple e_i. CrowdGame labels e_i using label of the rule in $\mathcal{R}_{\mathrm{q}}^i$ with the maximum accuracy.

References

1. Abad, A., Nabi, M., Moschitti, A.: Self-crowdsourcing training for relation extraction. In: ACL pp. 518–523 (2017)
2. Bishop, C.M.: Pattern Recognition and Machine Learning, Information Science and Statistics, 5th edn. Springer, Berlin (2007)
3. Bowman, K., Shenton, L.: Parameter estimation for the beta distribution. J. Stat. Comput. Simul. **43**(3–4), 217–228 (1992)
4. Chai, C., Li, G., Li, J., Deng, D., Feng, J.: Cost-effective crowdsourced entity resolution: a partial-order approach. In: SIGMOD, pp. 969–984 (2016)
5. Church, K.W., Hanks, P.: Word association norms, mutual information, and lexicography. Comput. Linguist. **16**(1), 22–29 (1990)
6. Das, S., P. S. G. C., 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, pp. 1431–1446 (2017)
7. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: a survey. IEEE Trans. Knowl. Data Eng. **19**(1), 1–16 (2007)
8. Fan, J., Li, G.: Human-in-the-loop rule learning for data integration. IEEE Data Eng. Bull. **41**(2), 104–115 (2018)
9. Fan, J., Li, G., Ooi, B. C., Tan, K., Feng, J.: icrowd: An adaptive crowdsourcing framework. In SIGMOD, pp. 1015–1030 (2015)
10. Fan, J., Lu, M., Ooi, B.C., Tan, W., Zhang, M.: A hybrid machine-crowdsourcing system for matching web tables. ICDE **2014**, 976–987 (2014)
11. Fan, J., Zhang, M., Kok, S., Lu, M., Ooi, B.C.: Crowdop: Query optimization for declarative crowdsourcing systems. IEEE Trans. Knowl. Data Eng. **27**(8), 2078–2092 (2015)
12. Franklin, M. J., Kossmann, D., Kraska, T., Ramesh, S., Xin, R.: Crowddb: answering queries with crowdsourcing. In: SIGMOD, pp. 61–72 (2011)
13. Gokhale, C., Das, S., Doan, A., Naughton, J.F., Rampalli, N., Shavlik, J.W., Zhu, X.: Corleone: Hands-off crowdsourcing for entity matching. In: SIGMOD, pp. 601–612 (2014)
14. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: NIPS, pp. 2672–2680 (2014)
15. Haas, D., Wang, J., Wu, E., Franklin, M.J.: Clamshell: Speeding up crowds for low-latency data labeling. PVLDB **9**(4), 372–383 (2015)
16. Hoffmann, R., Zhang, C., Ling, X., Zettlemoyer, L., Weld, D. S.: Knowledge-based weak supervision for information extraction of overlapping relations. In: Association for Computational Linguistics ACL, pp. 541–550 (2011)
17. Joglekar, M., Garcia-Molina, H., Parameswaran, A.: Comprehensive and reliable crowd assessment algorithms. In: Gehrke, J., Lehner, W., Shim, K., Cha, S.K., Lohman, G.M. (eds) ICDE. IEEE Computer Society, pp. 195–206. (2015) https://doi.org/10.1109/ICDE.2015.7113284
18. Khan, A.R., Garcia-Molina, H.: Attribute-based crowd entity resolution. In: CIKM, pp. 549–558 (2016)
19. Kusner, M.J., Sun, Y., Kolkin, N.I., Weinberger, K.Q.: From word embeddings to document distances. ICML **2015**, 957–966 (2015)
20. LeCun, Y., Bengio, Y., Hinton, G.E.: Deep learning. Nature **521**(7553), 436–444 (2015)
21. Li, G.: Human-in-the-loop data integration. PVLDB **10**(12), 2006–2017 (2017)
22. Li, G., Chai, C., Fan, J., Weng, X., Li, J., Zheng, Y., Li, Y., Yu, X., Zhang, X., Yuan, H.: CDB: optimizing queries with crowd-based selections and joins. In: SIGMOD, pp. 1463–1478 (2017)
23. Li, G., Wang, J., Zheng, Y., Franklin, M.J.: Crowdsourced data management: a survey. IEEE Trans. Knowl. Data Eng. **28**(9), 2296–2319 (2016)

24. Liu, A., Soderland, S., Bragg, J., Lin, C.H., Ling, X., Weld, D.S.: Effective crowd annotation for relation extraction. In: NAACL HLT, pp. 897–906 (2016)
25. Liu, X., Lu, M., Ooi, B.C., Shen, Y., Wu, S., Zhang, M.: CDAS: a crowdsourcing data analytics system. PVLDB **5**(10), 1040–1051 (2012)
26. Marcus, A., Wu, E., Karger, D.R., Madden, S., Miller, R.C.: Demonstration of qurk: a query processor for humanoperators. SIGMOD **2011**, 1315–1318 (2011)
27. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, (2013)
28. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: NIPS, pp. 3111–3119 (2013)
29. Mintz, M., Bills, S., Snow, R., Jurafsky, D.: Distant supervision for relation extraction without labeled data. ACL **2009**, 1003–1011 (2009)
30. Parisi, F., Strino, F., Nadler, B., Kluger, Y.: Ranking and combining multiple predictors without labeled data. Proc. Natl. Acad. Sci. USA **111**(4), 1253–8 (2014)
31. Park, H., Pang, R., Parameswaran, A.G., Garcia-Molina, H., Polyzotis, N., Widom, J.: Deco: a system for declarative crowdsourcing. PVLDB **5**(12), 1990–1993 (2012)
32. Ratner, A., Bach, S.H., Ehrenberg, H.R., Fries, J.A., Wu, S., Ré, C.: Snorkel: rapid training data creation with weak supervision. PVLDB **11**(3), 269–282 (2017)
33. Ratner, A.J., Sa, C.D., Wu, S., Selsam, D., Ré, C.: Data programming: creating large training sets, quickly. NIPS **2016**, 3567–3575 (2016)
34. Roth, B., Klakow, D.: Combining generative and discriminative model scores for distant supervision. In: EMNLP, pp. 24–29 (2013)
35. Rubner, Y., Tomasi, C., Guibas, L.J.: A metric for distributions with applications to image databases. ICCV, IEEE Computer Society, pp. 59–66 (1998). https://doi.org/10.1109/ICCV.1998.710701
36. Sheng, V.S., Provost, F., Ipeirotis, P.G.: Get another label? improving data quality and data mining using multiple, noisy labelers. In: SIGKDD, pp. 614–622. ACM (2008)
37. Sun, C., Shrivastava, A., Singh, S., Gupta, A.: Revisiting unreasonable effectiveness of data in deep learning era. CoRR, arXiv:1707.02968 (2017)
38. Takamatsu, S., Sato, I., Nakagawa, H.: Reducing wrong labels in distant supervision for relation extraction. In: Meeting of the Association for Computational Linguistics: Long Papers, pp. 721–729 (2012)
39. Tong, Y., Chen, L., Zhou, Z., Jagadish, H.V., Shou, L., Lv, W.: Slade: a smart large-scale task decomposer in crowdsourcing. IEEE Trans. Knowl. Data Eng. **30**(8), 1588–1601 (2018)
40. Tong, Y., She, J., Ding, B., Wang, L., Chen, L.: Online mobile micro-task allocation in spatial crowdsourcing. In: ICDE, pp. 49–60 (2016)
41. Verroios, V., Garcia-Molina, H., Papakonstantinou, Y.: Waldo: An adaptive human interface for crowd entity resolution. In: SIGMOD, pp. 1133–1148 (2017)
42. Vesdapunt, N., Bellare, K., Dalvi, N.N.: Crowdsourcing algorithms for entity resolution. In: PVLDB (2014)
43. Wang, J., Kraska, T., Franklin, M.J., Feng, J.: Crowder: Crowdsourcing entity resolution. In: PVLDB (2012)
44. Wang, J., Li, G., Kraska, T., Franklin, M.J., Feng, J.: Leveraging transitive relations for crowdsourced joins. In: SIGMOD, pp. 229–240 (2013)
45. Wang, J., Yu, L., Zhang, W., Gong, Y., Xu, Y., Wang, B., Zhang, P., Zhang, D.: Irgan: a minimax game for unifying generative and discriminative information retrieval models. In: SIGIR, pp. 515–524. ACM (2017)
46. Wang, S., Xiao, X., Lee, C.: Crowd-based deduplication: an adaptive approach. In: SIGMOD, pp. 1263–1277 (2015)
47. Whang, S.E., Lofgren, P., Garcia-Molina, H.: Question selection for crowd entity resolution. PVLDB **6**(6), 349–360 (2013)
48. Zhang, Y., Chen, X., Zhou, D., Jordan, M.I.: Spectral methods meet EM: a provably optimal algorithm for crowdsourcing. In: International Conference on Neural Information Processing Systems, pp. 1260–1268 (2014)
49. Zheng, Y., Li, G., Li, Y., Shan, C., Cheng, R.: Truth inference in crowdsourcing: is the problem solved? PVLDB **10**(5), 541–552 (2017)
50. Zheng, Y., Wang, J., Li, G., Cheng, R., Feng, J.: QASCA: a quality-aware task assignment system for crowdsourcing applications. In: SIGMOD, pp. 1031–1046 (2015)