Cost-Effective Data Annotation using Game-Based Crowdsourcing

Jingru Yang[†], Ju Fan[†][*], Zhewei Wei[‡][†], Guoliang Li[§], Tongyu Liu[†], Xiaoyong Du[†]
[†] *DEKE Lab & School of Information, Renmin University of China, China*
[§] *Department of Computer Science, Tsinghua University, China*
[‡] *SKLSDE Lab, Beihang University, China*
{hinsonver, fanj, zhewei, liuty, duyong}@ruc.edu.cn; liguoliang@tsinghua.edu.cn

ABSTRACT

Large-scale data annotation is indispensable for many applications, such as machine learning and data integration. However, existing annotation solutions either incur expensive cost for large datasets or produce noisy results. This paper introduces a cost-effective annotation 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 *precision*. 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 annotated 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 and precision, while rule refuter tries to refute its opponent rule generator by checking some tuples that provide enough evidence to reject rules covering the tuples. This paper studies the challenges in CrowdGame. The first is to balance the trade-off between coverage and precision. We define the loss of a rule by considering the two factors. The second is rule precision 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 conduct experiments on entity matching and relation extraction, and the results show that our method outperforms state-of-the-art solutions.

PVLDB Reference Format:
Jingru Yang, Ju Fan, Zhewei Wei, Guoliang Li, Tongyu Liu, Xiaoyong Du. Cost-Effective Data Annotation using Game-Based Crowdsourcing. *PVLDB*, 12(1): 57-70, 2018.
DOI: https://doi.org/10.14778/3275536.3275541

1. INTRODUCTION

In many applications, such as data integration and machine learning (ML), it is indispensable to obtain large-scale *annotated 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 annotation costs or efforts.

To address the problem, crowdsourcing can be utilized to harness the crowd to directly annotate 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 annotation* still inevitably incurs large annotation cost. Another approach is to leverage *labeling rules* (or rules for simplicity) that annotate 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 annotate tuples like `Michelle Obama` and `Barack Obama`. In entity matching, the *blocking* rules can quickly eliminate many record pairs which are obviously non-matched. Unfortunately, it is challenging to construct labeling rules. *Hand-crafted* rules from domain experts are not scalable, as it is time and effort consuming to handcraft many rules with large coverage. *Weak-supervision* rules automatically generated [33, 32], e.g., distant supervision rules, can largely cover the tuples; however, they may be very noisy and provide wrong labels.

In this paper we introduce a rule-based cost-effective data annotation approach. In particular, this paper focuses on studying *labeling rule generation using crowdsourcing* to significantly reduce annotation cost while still preserving high quality. To this end, we aim at generating "high-quality" labeling rules using two objectives. 1) *high coverage*: selecting the rules that cover as many tuples as possible to annotate the data. Intuitively, the larger the coverage is, the higher the cost on tuple-level annotation could be reduced. 2) *high precision*: preferring the rules that induce few wrong labels on their covered tuples.

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 precision. To address this problem, we propose to utilize crowdsourcing for rule selection. A straightforward approach employs the crowd to answer a *rule validation* task to check whether a rule is valid or invalid. Unfortunately, the crowd may give low-quality answers for a rule validation task, as a

[*] Ju Fan is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 12, No. 1
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3275536.3275541

rule may cover many tuples and the workers cannot examine all the tuples covered by the rule. To alleviate this, we can ask the crowd to answer a *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 precisions, while rule refuter tries to refute rule generator by checking some tuples that provide enough evidence to "reject" more rules.

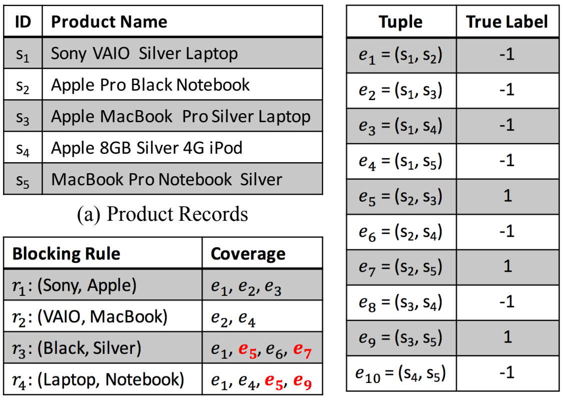
We study the research challenges in our game-based crowdsourcing. First, it is challenging to formalize the *quality* of a rule by trading-off its precision and coverage. We introduce the *loss* of a rule set that combines the uncovered tuples and the incorrectly covered tuples. We aim to select a rule set to minimize the loss. Second, it is hard to obtain the real precision 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 precision. 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 rules until the crowdsourcing budget is used up.

To summarize, we make the following contributions.

(1) We propose a data annotation 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 precision. We estimate precision of a rule by combining rule validation and tuple checking through Bayesian estimation.
(3) 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.

2. PROBLEM FORMULATION

This paper studies the *data annotation* 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 annotation* problem that considers two possible labels, denoted as $\mathcal{L} = \{L_1 = -1, L_2 = 1\}$. To illustrate the problem, let us consider an application of entity matching (EM) [7], i.e., identifying whether any pair of product records is the same entity, as shown in Figure 1. Note that the matching criterion in the example is the same product *model* and the same *manufacture*, without considering specifications like color and storage. In this application, each tuple is a product record pair, which is to be annotated 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 ($L_1 = -1$).



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)

Figure 1: Data annotation in entity matching.

2.1 Overview of Our Framework

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

Definition 1 (Labeling Rule). *A labeling rule is a function $r_j : \mathcal{E} \rightarrow \{L, \texttt{nil}\}$ 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 takes a set of unlabeled tuples as input and annotates them 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 *precision*. To this end, we first construct candidate rules, which may have various coverage and precision. There are two widely used ways to construct candidate rules: *hand-crafted* rules from domain experts and *weak-supervision* rules automatically generated by algorithms. Hand-crafted rules ask users to write domain-specific labeling heuristics based on their domain knowledge. However, hand-crafted rules are not scalable, especially for large datasets, as it is time and effort consuming to handcraft many rules with large coverage. Thus, weak-supervision rules automatically generated are introduced [33, 32], 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

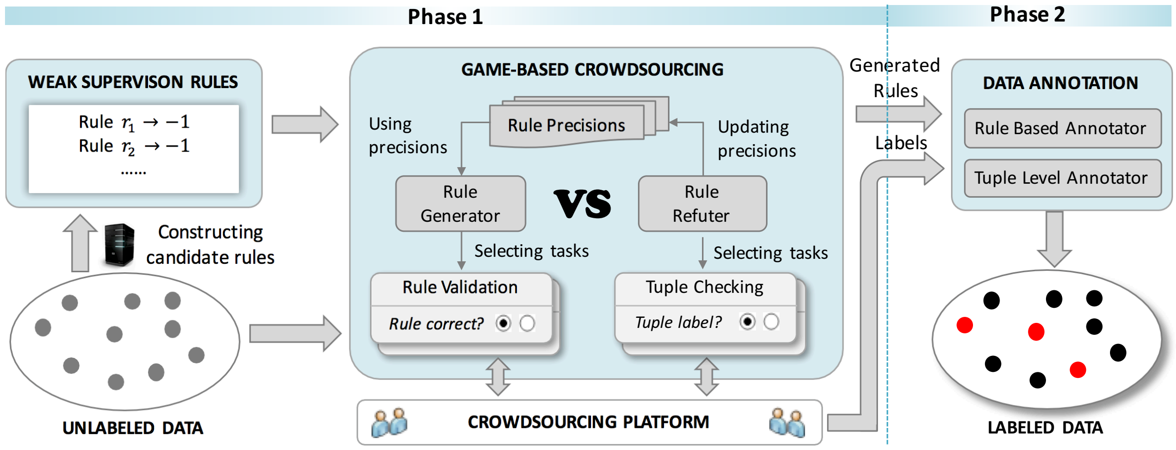


Figure 2: Framework of data annotation with game-based crowdsourcing.

provide wrong labels. Note that candidate rule construction will be presented in Section 6.

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 Section 2.2.

Phase II: Data annotation using rules. This phase is to annotate the tuples using the labeling rules generated in the previous phase. Note that, for the tuples not covered by the rules, we devise tuple-level annotation using conventional crowdsourcing approaches [43, 4], where tuple-level inference, such as transitivity, can also be applied.

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 the example below. We will discuss a more general case that some rules label $L_1 = -1$ while others provide $L_2 = 1$ in our technical report [48].

EXAMPLE 1. *Consider the blocking rules that annotate -1 for entity matching in Figure 1(b). Each rule, represented by two keywords, expresses how we discriminate product pairs covered by the rule. For example, r_1 : (*`Sony`*,* `Apple`*) expresses that any tuple, say $e_1 = \{s_1, s_2\}$, that satisfies s_1 containing* `Sony` *and s_2 containing* `Apple` *(or s_1 containing* `Apple` *and s_2 containing* `Sony`*) cannot be matched. We can see that r_1 covers three tuples, $\{e_1, e_2, e_3\}$, and their labels are $L_1 = -1$ (unmatched). We may also observe that some rules may not be precise: r_4 only correctly labels two out of four tuples (e_1 and e_4), because rule (*`Laptop`*,* `Notebook`*) is very weak to discriminate the products. Thus, our framework uses crowdsourcing to select the rules with large coverage and precision from the candidates. Suppose that $\{r_1, r_3\}$ are selected, and then 6 tuples can be annotated by the rules. For the other 4 tuples not covered, the framework can annotate them using conventional crowdsourcing.*

2.2 Labeling Rule Generation

This paper focuses on the labeling rule generation problem in the first phase of our framework. Intuitively, the problem aims to identify "high-quality" rules with the following two objectives. 1) *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 annotation (Phase II) could be reduced. 2) *high precision*: preferring the rules that induce few wrong labels on their covered tuples.

There may be a tradeoff between coverage and precision. On the one hand, some annotation scenarios prefer *precision*. For instance, in most of entity matching tasks, ground-truth labels are very skew, e.g., 7 tuples with label -1 vs. 3 tuples with 1 as shown in Figure 1(c). Thus, rule generation prefers not to induce too much 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 annotation cost reduction.

To balance the preference among coverage and precision, we introduce the *loss* of a rule set $\mathcal{R}$ that considers the following two factors. 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 $\sum_{e_i \in \mathcal{C}(\mathcal{R})} P(y_i \neq L)$,*

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

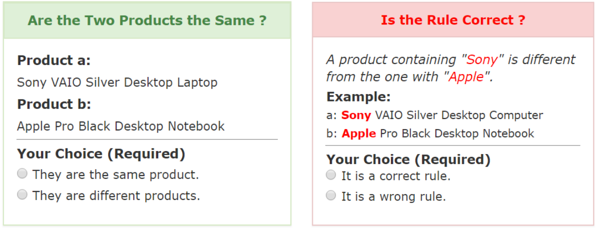
where γ is a parameter between $[0, 1]$ to balance the preference among coverage and quality of generated rules.

For example, consider $\mathcal{R}_1 = \{r_1\}$ covering three tuples without errors and $\mathcal{R}_2 = \{r_1, r_3\}$ covering more with wrongly labeled tuples (e_5 and e_7). As entity matching prefers precision over coverage on the blocking rules, one needs to set a small parameter γ, say $\gamma = 0.1$. Obviously, under this setting, we have $\Phi(\mathcal{R}_1) < \Phi(\mathcal{R}_2)$, which shows that a larger set $\mathcal{R}_2$ is worse than a smaller set $\mathcal{R}_1$.

Let $\mathcal{R}^c = \{r_1, r_2, \ldots, r_n\}$ denote a set of candidate rules generated by the candidate rule producing step. Now, we define the problem of rule generation as below.

Table 1: Table of Notations.

e_i; $\mathcal{E}$	a tuple to be annotated; a set of tuples
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
λ_j ($\hat{\lambda}_j$)	precision (precision estimate) of rule r_j
$\Phi(\mathcal{R})$	loss of a rule set $\mathcal{R}$



Are the Two Products the Same ?

Product a:

Sony VAIO Silver Desktop Laptop

Product b:

Apple Pro Black Desktop Notebook

Your Choice (Required)

○ They are the same product.

○ They are different products.

(a) Tuple checking task.

Is the Rule Correct ?

A product containing "Sony" is different from the one with "Apple".

Example:

a: **Sony** VAIO Silver Desktop Computer

b: **Apple** Pro Black Desktop Notebook

Your Choice (Required)

○ It is a correct rule.

○ It is a wrong rule.

(b) Rule validation task.

Figure 3: A two-pronged crowdsourcing scheme.

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

As the probability $P(y_i \neq L)$ is hard to obtain, this paper focuses on using crowdsourcing to achieve the above loss minimization, which will be presented in Section 3. For ease of presentation, we summarize frequently used notations (some only introduced later) in Table 1.

3. CROWDSOURCED RULE GENERATION

3.1 Game-Based Crowdsourcing

Labeling rule generation is very challenging as there may be many rules with *diverse* and *unknown* precision. A naïve crowdsourcing approach is to first evaluate each rule by sampling some covered tuples and checking them through crowdsourcing. For example, Figure 3(a) 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 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*. For example, Figure 3(b) 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. This intuition is supported by our empirical observations in our technical report [48]. 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. Thus, the fine-grained tuple checking tasks are also indispensable.

A game-based crowdsourcing approach. Due to the fixed amount of crowdsourcing budget, there is usually a tradeoff 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 tradeoff, we introduce a *game-based* crowdsourcing approach CrowdGame, as illustrated in the central part of Figure 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 Equation (1) as the game value function.

- RuleGen plays as a *minimizer*: it identifies some rules $\mathcal{R}$ from the candidates $\mathcal{R}^{\mathrm{C}}$ 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 enough evidence to "reject" more rules in $\mathcal{R}$, i.e., maximizing the rule set loss $\Phi(\mathcal{R})$.

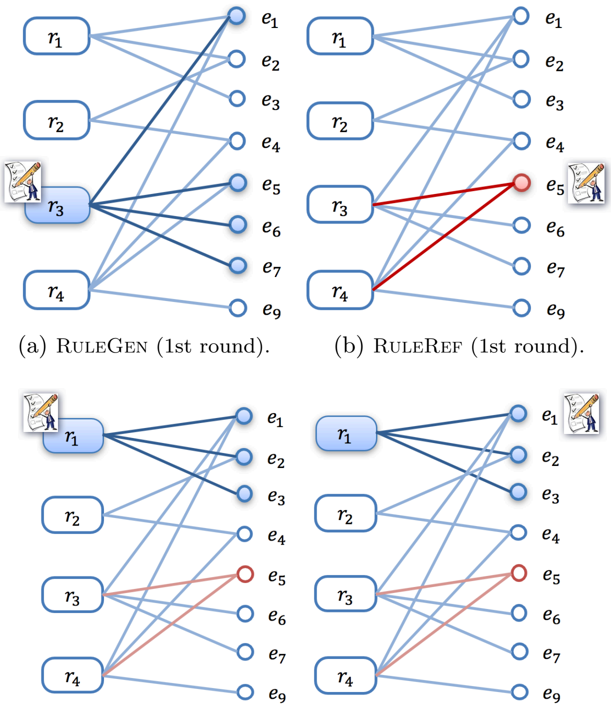
A well-known mechanism to solve such games is the *minimax strategy* in game theory: the minimizer aims to minimize the maximum possible loss made by the maximizer.

Example 2. *Consider our example under a budget with 4 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 $\Phi(\mathcal{R}_1) = 3*0.1+2*0.9 = 2.1$ (3 uncovered tuples and 2 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 4 tuples and can largely reduce the first term of the loss in Equation (1). However, its opponent* RuleRef *finds a "counter-example" 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

Note that, for illustration purpose, Example 2 shows an extreme case that one counter-example 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 "precision".

We first introduce the *precision*, 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., $\lambda_j = \frac{\sum_{e_i \in \mathcal{C}(r_j)} \mathbb{1}_{\{y_i = L\}}}{|\mathcal{C}(r_j)|}$, 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 precisions of all rules in $\mathcal{R}$. Rule precision λ_j is actually unknown to us, and thus



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

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

Figure 4: Example of game-based crowdsourcing.

we need to estimate it using tuple checking crowdsourcing tasks. Formally, let $\hat{\lambda}_j(\mathcal{E}_q)$ be an estimator of λ_j for rule r_j based on a set $\mathcal{E}_q$ of tuples checked by the crowd, and $\hat{\Lambda}^{\mathcal{R}}(\mathcal{E}_q) = \{\hat{\lambda}_j(\mathcal{E}_q)\}$ is the set of estimators, each of which is used for evaluating an individual rule $r_j \in \mathcal{R}$. We use $\hat{\Lambda}^{\mathcal{R}}(\mathcal{E}_q)$ to compute our overall loss defined in Equation (1). Formally, let $\mathcal{R}^i \subseteq \mathcal{R}$ denote the set of rules in $\mathcal{R}$ covering tuple e_i, i.e., $\mathcal{R}^i = \{r_j \in \mathcal{R} | r_j(e_i) \neq \texttt{nil}\}$. For ease of presentation, we denote $P(y_i = L)$ as $P(a_i)$ if the context is clear. Then, we introduce $\Phi(\mathcal{R}|\mathcal{E}_q)$ to denote the *estimated* loss based on a set $\mathcal{E}_q$ of tuples checked by the crowd, i.e.,

$$\Phi(\mathcal{R}|\mathcal{E}_q) = \gamma(|\mathcal{E}| - |\mathcal{C}(\mathcal{R})|) + (1-\gamma)\sum_{e_i \in \mathcal{C}(\mathcal{R})} 1 - P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_q))$$

$$= \gamma|\mathcal{E}| - (1-\gamma)\sum_{e_i \in \mathcal{C}(\mathcal{R})} \Big\{P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_q)) - \frac{1-2\gamma}{1-\gamma}\Big\} \qquad (2)$$

The key in Equation (2) is $P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_q))$, 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 precisions $\Lambda^{\mathcal{R}^i}(\mathcal{E}_q)$. Next, we discuss how to compute $P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_q))$. 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 $\hat{\lambda}_j(\mathcal{E}_q)$ and thus the probability that e_i is correctly labeled is $\hat{\lambda}_j(\mathcal{E}_q)$. Second, if e_i is covered by multiple rules, the $P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_q))$ is not easy to estimate, because rules may have various kinds of correlation. In this paper, we use a "conservative" strategy that computes $P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_q))$ as the *maximum* rule precision among $\Lambda^{\mathcal{R}^i}$, i.e., $P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_q)) = \max_{r_j \in \mathcal{R}^i} \hat{\lambda}_j(\mathcal{E}_q)$. The rational is that, e_i is covered by rule r_j^* with the largest precision, and its $P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_q))$ is at least $\hat{\lambda}_j^*$. Consider our example in Figure 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 $P(a_1|\hat{\Lambda}^{\mathcal{R}^7}(\mathcal{E}_q))$ 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 based on rule precision estimation. 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, the minimax objective is defined as

$$\mathcal{O}^{\mathcal{R}_q^*, \mathcal{E}_q^*} = \min_{\mathcal{R}_q} \max_{\mathcal{E}_q} \Phi(\mathcal{R}_q|\mathcal{E}_q)$$

$$\iff \max_{\mathcal{R}_q} \min_{\mathcal{E}_q} \sum_{e_i \in \mathcal{C}(\mathcal{R}_q)} \Big\{P(a_i|\hat{\Lambda}^{\mathcal{R}_i}(\mathcal{E}_q)) - \frac{1-2\gamma}{1-\gamma}\Big\}$$

$$\iff \max_{\mathcal{R}_q} \min_{\mathcal{E}_q} \sum_{e_i \in \mathcal{C}(\mathcal{R}_q)} \Big\{\max_{r_j \in \mathcal{R}^i} \hat{\lambda}_j(\mathcal{E}_q) - \frac{1-2\gamma}{1-\gamma}\Big\} \qquad (3)$$

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

$$\mathcal{J}^{\mathcal{R}_q, \mathcal{E}_q} = \sum_{e_i \in \mathcal{C}(\mathcal{R}_q)} \Big\{\max_{r_j \in \mathcal{R}^i} \hat{\lambda}_j(\mathcal{E}_q) - \frac{1-2\gamma}{1-\gamma}\Big\} \qquad (4)$$

Based on Equation (3), 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 $|\mathcal{R}_q| + |\mathcal{E}_q| \leq k$. 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 precision to minimize the loss. On the contrary, RuleRef aims to check tuples which are helpful to identify low-precision rules that cover many tuples, so as to effectively 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 Section 4. Second, as shown in Equation (3), the objective is based on rule precision estimators, e.g., $\hat{\lambda}_j(\mathcal{E}_q)$. We introduce effective estimation techniques in Section 5.

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 k (crowdsourcing budget) tasks are crowdsourced, where each iteration consists of a RuleGen step and a RuleRef step.

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 $\mathcal{R}_q^{(t)}$ that maximizes the *rule selection criterion* denoted by $\Delta g(\mathcal{R}|\mathcal{J}^{\mathcal{R}_q, \mathcal{E}_q})$ in the t-th iteration (line 1). We will introduce the criterion $\Delta g(\mathcal{R}|\mathcal{J}^{\mathcal{R}_q, \mathcal{E}_q})$ and present an algorithm

```
Algorithm 1: MinimaxSelect ($\mathcal{R}^{c}$, $\mathcal{E}$, $k$, $b$)
Input: $\mathcal{R}^{c}$: candidate rules; $\mathcal{E}$: tuples to be labeled;
       $k$: a budget; $b$: a crowdsourcing batch
Output: $\mathcal{R}_{q}$: a set of generated rules
Initialize $\mathcal{R}_{q} \leftarrow \emptyset$, $\mathcal{E}_{q} \leftarrow \emptyset$ ;
for each iteration $t$ do
    /* Rule Generator Step */
    Select $\mathcal{R}_{q}^{(t)} \leftarrow \arg_{\mathcal{R}} \max_{\mathcal{R} \subseteq \mathcal{R}^{c} - \mathcal{R}_{q}, |\mathcal{R}|=b} \Delta g(\mathcal{R}|\mathcal{J}^{\mathcal{R}_{q},\mathcal{E}_{q}})$ ;
    Crowdsource $\mathcal{R}_{q}^{(t)}$ as rule validation tasks ;
    Add the crowd validated rules in $\mathcal{R}_{q}^{(t)}$ into $\mathcal{R}_{q}$ ;
    Update objective $\mathcal{J}^{\mathcal{R}_{q},\mathcal{E}_{q}}$ ;
    $\mathcal{R}^{c} \leftarrow \mathcal{R}^{c} - \mathcal{R}_{q}^{(t)}$ ;
    /* Rule Refuter Step */
    Select $\mathcal{E}_{q}^{(t)} \leftarrow \arg_{\mathcal{E}} \min_{\mathcal{E} \in \mathcal{E} - \mathcal{E}_{q}, |\mathcal{E}|=b} \Delta f(\mathcal{E}_{q}|\mathcal{J}^{\mathcal{R}_{q},\mathcal{E}_{q}})$ ;
    Crowdsource $\mathcal{E}_{q}^{(t)}$ as tuple checking tasks ;
    Add the crowd-checked $\mathcal{E}_{q}^{(t)}$ into $\mathcal{E}_{q}$ ;
    Update precision $\hat{\Lambda}^{\mathcal{R}_{q}}(\mathcal{E}_{q})$;
    Update budget $k \leftarrow k - 2b$ ;
    if $k = 0$ then break ;
Remove rules from $\mathcal{R}_{q}$ with $\hat{\lambda}_{j} \leq \frac{1-2\gamma}{1-\gamma}$ ;
Return $\mathcal{R}_{q}$ ;
```

for selecting rules based on the criterion in Section 4.1. After selecting $\mathcal{R}_{q}^{(t)}$, RuleGen crowdsources $\mathcal{R}_{q}^{(t)}$, adds the crowd-validated rules into $\mathcal{R}_{q}$, and updates objective $\mathcal{J}^{\mathcal{R}_{q},\mathcal{E}_{q}}$. Note that we do not consider the rules failed crowd validation, because they have *much lower* quality 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 $\mathcal{E}_{q}^{(t)}$, so as to minimize the *tuple selection criterion* denoted by $\Delta f(\mathcal{E}|\mathcal{J}^{\mathcal{R}_{q},\mathcal{E}_{q}})$. We will discuss the criterion and a selection algorithm in Section 4.2. After obtaining the crowd answers for $\mathcal{E}_{q}^{(t)}$, RuleRef adds $\mathcal{E}_{q}^{(t)}$ into $\mathcal{E}_{q}$ and updates the precision estimates $\hat{\Lambda}^{\mathcal{R}_{q}}(\mathcal{E}_{q})$.

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 k and check if the algorithm terminates (i.e., the budget is used up). The algorithm continues to iteration $t+1$ if $k > 0$. 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 Figure 4 with $k = 4$ 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 precision estimates and continues to the second iteration as shown in Figures 4(c) and 4(d). After these two iterations, the budget is used up, and the algorithm returns $\mathcal{R}_{q} = \{r_1\}$ as the result.

Next, we explain the task selection algorithms of RuleGen and RuleRef in the following two subsections.

4.1 Task Selection for Rule Generator

The basic idea of task selection for RuleGen, as observed from Equation (3), 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 precision 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 $\mathcal{R}^{\checkmark} \subseteq \mathcal{R}$, and measure the the *expected improvement* on $\mathcal{J}^{\mathcal{R}_{q},\mathcal{E}_{q}}$ that $\mathcal{R}^{\checkmark}$ achieves.

Formally, let $P(\mathcal{R}^{\checkmark})$ denote the probability that the crowd returns $\mathcal{R}^{\checkmark} \subseteq \mathcal{R}$ as the validated rules, and rules in $\mathcal{R} - \mathcal{R}^{\checkmark}$ 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 $P(\mathcal{R}^{\checkmark}) = \prod_{r \in \mathcal{R}^{\checkmark}} P(r) \cdot \prod_{r' \in \mathcal{R} - \mathcal{R}^{\checkmark}} \left(1 - P(r')\right)$. For example, consider rule set $\mathcal{R}_1 = \{r_1, r_3\}$ shown in Figure 4: there are four possible values for $\mathcal{R}_{1}^{\checkmark}$, 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,

$$\Delta g(\mathcal{R}|\mathcal{J}) = \sum_{\mathcal{R}^{\checkmark}} P(\mathcal{R}^{\checkmark}) \cdot \left(\mathcal{J}^{\mathcal{R}^{\checkmark} \cup \mathcal{R}_{q},\mathcal{E}_{q}} - \mathcal{J}^{\mathcal{R}_{q},\mathcal{E}_{q}} \right). \tag{5}$$

For instance, consider a rule set $\mathcal{R} = \{r_1\}$ in our previous example and $\mathcal{R}_{q} = \emptyset$. Suppose that we have estimated precision $\hat{\lambda}_1 = 1.0$ 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 value of 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 our technical report [48].

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 enables 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, the algorithm 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 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 $\mathcal{J}^{\mathcal{R}_{\mathtt{q}},\mathcal{E}_{\mathtt{q}}}$ by checking tuples to re-estimate rule precisions. Given tuple e_i, it considers the following two factors.

1) The first one is the *refute probability*, denoted by $P(e_i^{\times})$ that the crowd identifies that true label y_i of e_i is not label L provided by rules. Intuitively, the higher the refute probability is, the more preferable the tuple is. We will discuss how to estimate such probability later. 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 precision estimates of r_3 and r_4, and thus have chance to reduce the term $\max_{r_j \in \mathcal{R}^i} \hat{\lambda}_j(\mathcal{E}_{\mathtt{q}}) - \frac{1-2\gamma}{1-\gamma}$ for 6 tuples in the objective $\mathcal{J}^{\mathcal{R}_{\mathtt{q}},\mathcal{E}_{\mathtt{q}}}$. 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 $\mathcal{J}^{\mathcal{R}_{\mathtt{q}},\mathcal{E}_{\mathtt{q}}}$, as the maximum precision associated to 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.,

$$\begin{aligned} \mathcal{I}(e_i^{\times}) &= \mathcal{J}^{\mathcal{R}_{\mathtt{q}},\mathcal{E}_{\mathtt{q}}} - \mathcal{J}^{\mathcal{R}_{\mathtt{q}},\mathcal{E}_{\mathtt{q}} \cup \{e_i^{\times}\}} \\ &= \sum_{e_l \in \mathcal{C}(\mathcal{R}_{\mathtt{q}})} \max_{r_j \in \mathcal{R}^l} \left\{ \hat{\lambda}_j(\mathcal{E}_{\mathtt{q}}) - \hat{\lambda}_j(\mathcal{E}_{\mathtt{q}} \cup \{e_i^{\times}\}) \right\}. \end{aligned} \tag{6}$$

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

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

$$\Delta f(\mathcal{E}|\mathcal{J}) = -\sum_{\mathcal{E}^{\times}} P(\mathcal{E}^{\times}) \cdot \mathcal{I}(\mathcal{E}^{\times}). \tag{7}$$

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

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. We omit the pseudo-code due to the space limit. Moreover, similar to RuleGen, $\Delta f(\mathcal{E}|\mathcal{J})$ can be incrementally computed without the exponential enumeration on $P(\mathcal{E}^{\times})$.

We discuss how to obtain refute probability $P(e_i^{\times})$ for entity matching and relation extraction in Section 6.

5. RULE PRECISION ESTIMATION

The challenge in estimating rule precision $\hat{\lambda}_j(\mathcal{E}_{\mathtt{q}})$ 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 Figure 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 precision evaluation.

To formalize the above 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 precision. Formally, let $p(\lambda|r^{\surd}, \mathcal{E}_{\mathtt{q}})$ denote the probability distribution of precision λ of rule r given the fact that r is validated by the crowd (denoted by $r^{\surd}$) and checked by a set $\mathcal{E}_{\mathtt{q}}$ of tuples. Then, following the Bayesian rule and assuming that rule validation and tuple checking results are conditionally independent given λ, we have

$$p(\lambda|r^{\surd}, \mathcal{E}_{\mathtt{q}}) = \frac{p(\mathcal{E}_{\mathtt{q}}|\lambda) \cdot p(\lambda|r^{\surd})}{p(\mathcal{E}_{\mathtt{q}}|r^{\surd})}, \tag{8}$$

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

Likelihood of tuple observations. Recall that, given a set $\mathcal{E}_{\mathtt{q}}$ of checked tuples, we use $\mathcal{E}_{\mathtt{q}}^{\times}$ and $\mathcal{E}_{\mathtt{q}}^{\surd}$ to respectively denote the subsets of $\mathcal{E}_{\mathtt{q}}$ refuted and passed by the crowd (see Section 4.2 for more details on tuple refuting). Clearly, we have $\mathcal{E}_{\mathtt{q}}^{\times} \cup \mathcal{E}_{\mathtt{q}}^{\surd} = \mathcal{E}_{\mathtt{q}}$ and $\mathcal{E}_{\mathtt{q}}^{\times} \cap \mathcal{E}_{\mathtt{q}}^{\surd} = \emptyset$. Then, given precision λ, we consider that $\mathcal{E}_{\mathtt{q}}$ follows a *binomial distribution* with λ as its parameter, i.e.,

$$p(\mathcal{E}_{\mathtt{q}}|\lambda) = \binom{|\mathcal{E}_{\mathtt{q}}|}{|\mathcal{E}^{\times}|} \cdot \lambda^{|\mathcal{E}_{\mathtt{q}}^{\surd}|}(1-\lambda)^{|\mathcal{E}_{\mathtt{q}}^{\times}|}, \tag{9}$$

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

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.,

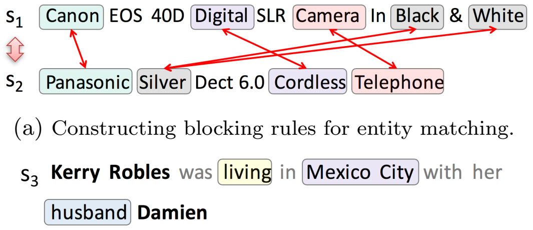
$$p(\lambda|r^{\surd}) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \lambda^{\alpha-1}(1-\lambda)^{\beta-1}, \tag{10}$$

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

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

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

which is also a beta distribution with the two parameters $\alpha + |\mathcal{E}_{\mathtt{q}}^{\surd}|$ and $\beta + |\mathcal{E}^{\times}|$. Then, we compute the estimate $\hat{\lambda}$ as



s1 Canon EOS 40D Digital SLR Camera In Black & White
s2 Panasonic Silver Dect 6.0 Cordless Telephone
s3 Kerry Robles was living in Mexico City with her
husband Damien

(a) Constructing blocking rules for entity matching.

(b) Constructing extraction rules for relation extraction.

Figure 5: Candidate rules construction.

the expectation of λ to minimize the squared error.

$$\hat{\lambda}(\mathcal{E}_{\text{q}}) = \mathbb{E}[\lambda | r^{\checkmark}, \mathcal{E}_{\text{q}}] = \frac{\alpha + |\mathcal{E}_{\text{q}}^{\checkmark}|}{\alpha + \beta + |\mathcal{E}_{\text{q}}|}. \qquad (11)$$

LEMMA 2. *Expectation of squared error, $\mathbb{E}[(\lambda - \hat{\lambda})^2]$ is minimized at the estimate $\hat{\lambda}$ computed by Equation (11) [2].*

EXAMPLE 3. *Consider prior* beta$(4, 1)$ *with $\alpha = 4$ and $\beta = 1$. we examine how "data observation" is used to adjust the prior. When crowdsourcing* 3 *tuples and receiving* 2 *passed and* 1 *refuted. Applying Equation (11), we estimate $\hat{\lambda} = 0.75$. Similarly, after crowdsourcing* 7 *tuples with* 4 *passed and* 3 *refuted answers, we estimate $\hat{\lambda}$ as* 0.67*. We can see that, although both of the cases have one more passed tuples than the refuted ones, we have a lower estimate, because more refuted tuples are observed. For better illustration, we plot a detailed figure in our technical report [48].*

Remarks. First, we want to emphasize that the set $\mathcal{E}_{\text{q}}$ of checked tuples will be incrementally updated as RULEREF selects more tuple checking tasks, as illustrated in Section 4.2. From Equation (11), we can clearly see how a refuted tuple $e_i^{\times}$ can decrease precision 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 precision 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. CANDIDATE RULES CONSTRUCTION

This section presents our methods to create candidate rules from the raw text data for *entity matching* (EM) (Section 6.1) and *relation extraction* (RE) (Section 6.2).

6.1 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 Figure 5(a), 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[1] 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 5(a) illustrates an example of using WMD to align keywords between two 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 Section 4.2. The intuition is that refute probability captures how likely the crowd will annotate 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 annotate the tuple as matched. The similarity-based idea is also used in other crowdsourced entity matching works [44, 4].

6.2 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 Figure 5(b). 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

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

Table 2: Statistics of Datasets and Crowd Answers

	Abt-Buy	Ama-Goo	Ebay	Spouse
# +1 tuples	1,090	1,273	2,057	424
# −1 tuples	227,715	179,525	107,847	5,493
# 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%

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. Note that we investigate the effect of such $P(e_i^\times)$ estimation in our technical report [48].

7. EXPERIMENTS

This section evaluates the performance of our approach. We evaluate different task selection strategies for rule generation, and compare our approach with the state-of-the-art methods. Note that, due to the space limit, we report additional experiments in our technical report [48].

7.1 Experiment Setup

Datasets. We consider two real-world applications, namely entity matching and relation extraction. For entity matching, we evaluate the approaches on three real datasets. Table 2 shows the statistics of the datasets. 1) Abt-Buy contains electronics product records from two websites, 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 Figure 1. Following the existing works in entity matching [43, 4], we prune the pairs with similarity smaller than a threshold 0.3 (we use WMD [19] to measure similarity), and obtain $1,090$ tuples with label 1 (matched) and $227,715$ tuples with label -1 (unmatched). 2) Ama-Goo contains software products from two websites, Amazon and Google. Similar to Abt-Buy, we obtain $1,273$ tuples with label 1 and $179,525$ tuples with label -1. 3) Ebay contains beauty products collected from website Ebay. Using the above method, we respectively obtain $2,057$ and $107,847$ tuples with labels 1 and -1. For these three datasets, we use the method in Section 6.1 to 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 if two person in a sentence have spouse relation. The Spouse dataset contains 2591 news articles[2]. 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 $5,493$ tuples with label -1 (entities do not have spouse relation). We construct candidate rules using the method in Section 6.2 and obtain 360 rules. Note the rules on this dataset can annotate both 1 (e.g., husband) and -1 (e.g., brother) labels.

Note that ground-truth of each of the above datasets is already included in the original dataset.

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 referred to Figure 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*. Similarly, 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 (as shown in 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 10 tasks in an HIT, and spend 1 US cent for each HIT. We assign each HIT to 3 workers and combine their answers via majority voting. We use qualification test to only allow workers with at least 150 approved HITs and 95% approval rate.

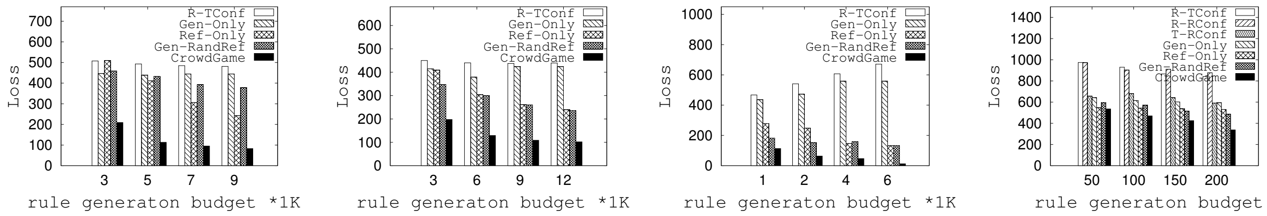
Parameter settings. First, γ is the weight balancing quality and coverage in our loss function in Equation (1). Due to the label skewness in entity matching as observed in Table 2, we set $\gamma = 0.001$ to prefer quality over coverage. In a similar way, we set $\gamma = 0.1$ for relation extraction. 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 Evaluation on Minimax Crowdsourcing

This section evaluates the minimax crowdsourcing objective and task selection algorithms. We compare different alternative task selection strategies in the framework of Algorithm 1. Gen-Only only utilizes rule validation tasks, and uses the prior as precision estimates (no tuple checking tasks). 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 task selection. Gen-RandRef considers both RuleGen and RuleRef. However, the RuleRef in this method uses a random strategy to select tuples for checking. CrowdGame is our game-based approach.

We also compare with simpler *conflict*-based heuristics. R-TConf selects the rules covering the largest number of "conflicting" tuples. 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. R-

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



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

Figure 6: Evaluating game-based crowdsourcing with different strategies.

`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 annotations from the rules covering the tuples.

Figure 6 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 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 precision 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` (Figure 6(a)). This is because a random refuter strategy may not be able to find the rules with the largest impact (Section 4.2), and thus performs weak to refute bad rules.

CrowdGame with our proposed task selection strategies 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.

7.3 Comparisons for Entity Matching (EM)

This section evaluates how CrowdGame boosts entity matching, and compares with state-of-the-art approaches.

Evaluation of CrowdGame on EM. We apply our two-phase framework to find record matches. Recall that Phase I uses crowd budget k for generating blocking rules, and Phase II applies the rules and crowdsources the record pairs not covered by the rules using tuple checking tasks (where transitive-based optimization technique [44] is applied).

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 ($\frac{2 \cdot precision \cdot recall}{precision + recall}$). On the other hand, we measure the total crowdsourcing cost in EM, including the rule generation crowd budget k in Phase I and the number of pair-based tasks in Phase II.

As shown in Table 3, increasing rule generation budget can improve both quality and cost. For instance, on the `Abt-Buy` dataset, with the increase of 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.

Approach Comparison. We compare CrowdGame with state-of-the-art approaches, where we set rule generation budget to 9,000, 12,000 and 6,000 on the three datasets respectively. We compare CrowdGame with the state-of-the-art 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 4, CrowdGame significantly reduces the total crowdsourcing cost over `Trans` and `ACD`, nearly by

Table 3: 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	$3,000$	0.764	0.083	0.927	0.916	0.921	$58,739$	$61,739$
	$5,000$	0.867	0.037	0.942	0.928	0.935	$34,189$	$39,189$
	$7,000$	0.898	0.033	0.960	0.955	0.957	$24,269$	$31,269$
	$9,000$	0.924	0.028	0.969	0.957	0.963	$17,381$	$26,381$
Ama-Goo	$3,000$	0.528	0.003	0.925	0.996	0.959	$89,671$	$92,671$
	$6,000$	0.697	0.002	0.947	0.998	0.972	$57,685$	$63,685$
	$9,000$	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	$1,000$	0.504	0.005	0.995	0.969	0.982	$33,321$	$34,321$
	$2,000$	0.785	0.004	0.995	0.985	0.990	$18,761$	$20,761$
	$4,000$	0.902	0.004	0.999	0.988	0.993	$5,292$	$9,292$
	$6,000$	0.966	0.003	1.000	0.996	0.998	$1,410$	$7,410$

Table 5: 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	$2,227$	$2,277$
100	0.687	0.027	0.537	0.545	0.645	0.591	$1,686$	$1,786$
150	0.719	0.026	0.453	0.585	0.640	0.611	$1,511$	$1,661$
200	0.695	0.027	0.149	0.810	0.635	0.712	$1,643$	$1,843$

Table 4: Comparison with EM Methods.

Dataset	Method	$F1$ of EM	Total Crowd Cost
Abt-Buy	Trans	0.864	203,715
	PartOrder	0	**1,063**
	ACD	0.887	216,025
	Snorkel	0.909	26,381
	CrowdGame	**0.963**	26,381
Ama-Goo	Trans	0.896	158,525
	PartOrder	0.486	**763**
	ACD	0.919	167,958
	Snorkel	0.923	48,115
	CrowdGame	**0.982**	48,115
Ebay	Trans	0.971	50,163
	PartOrder	0.553	**170**
	ACD	**0.998**	57,637
	Snorkel	0.857	7,410
	CrowdGame	**0.998**	7,410

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]). This setting asks the crowd to annotate tuples (e.g., record pairs in entity matching), and represents each crowd worker as well as her answers as a *labeling function*. Fed with the labeling functions, Snorkel outputs final annotation results. For fair comparison, we use exactly the same total crowdsourcing cost (Phases I and II) of CrowdGame as the crowdsourcing budget for Snorkel, e.g., $26,381$ on the Abt-Buy dataset, which makes sure that the two approaches rely the same crowd efforts. 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, selection by similarity will increase the chance of finding +1 tuples in the crowdsourcing set. As shown in Table 4, the experimental results show that, under the same crowdsourcing cost, CrowdGame outperforms Snorkel 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 annotate a large amount of tuples with high precision.

7.4 Comparison for Relation Extraction

This section evaluates the performance of CrowdGame for relation extraction on the Spouse dataset. We construct candidate rules using the method in Section 6. 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. Table 5 shows the performance of CrowdGame. With the increase of 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

Table 6: 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**

identify and refute more low-precision rules and thus significantly reduces false positives for relation extraction.

We compare CrowdGame with 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" (annotating +1), "ex-husband" (annotating +1), and "father" (annotating −1), and 2) a set of 6126 entity pairs with spouse relation extracted from an external knowledge base, DBPedia[3]. Second, we take both these manual rules and crowd annotations as labeling functions. Note that we use the similarity-based method (same to the EM scenario) to select tuples for crowdsourcing for 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 $1,843$ for both Snorkel and CrowdGame. As observed from Table 6, Snorkel with only manual rules achieves inferior quality. The reason is that the manual rules are based some generally summarized keywords and external knowledge, which are not specifically designed for the Spouse dataset. Moreover, further considering crowd annotations, 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.

8. RELATED WORK

Crowdsourced data annotation. 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 [25, 9, 51, 40, 51, 50], crowd DB systems [12, 26, 31, 11, 22, 39], etc. This paper pays special attention on *crowdsourced data annotation*, which acquires relatively low cost labeled data in a short time using crowdsourcing, with focus on reviewing such works in entity matching and relation extraction. Crowdsourced entity matching (aka. crowdsourced entity resolution) [21, 4, 18, 41, 13, 6, 47, 43, 46, 42, 8] has been extensively studied recently. These existing works have studied many aspects in the field, including task generation [43], transitivity-based inference [44, 4, 42], partial-order inference [4], and task selection [41]. However, most of them only annotate 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 [13, 6]. 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 [24, 1]. However, similar to entity matching, most of the works focus on tuple-level annotation.

Weak-supervision labeling rules. There are many works in the machine learning community to annotate large training sets using weak-supervision labeling rules. A well-known example is distant supervision [16, 34, 29, 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] annotate 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, 49]. Some approaches are recently proposed to consolidate noisy or even contradictory rules [36, 32]. 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 annotation: 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 precision, which results in performance superiority shown in our experiments.

Generative Adversarial Networks. The recent Generative Adversarial Networks (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 annotation, while GAN focuses on parameter learnings. Second, GAN uses algorithms such as stochastic gradient descent to optimize the parameters. In contrast, 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

We have studied the data annotation problem. Different from previous tuple-level annotation methods, we introduced *labeling rules* to reduce annotation cost while preserving high quality. We devised a crowdsourcing approach to generate high-quality rules with high coverage and precision. We first constructed a set of candidate rules and then solicited crowdsourcing to select high-quality rules. We utilized crowdsourcing to estimate quality of a rule by combining `rule validation` and `tuple checking`. We developed a game-based 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.

Acknowledgment. This work was supported by the 973 Program of China (2015CB358700, 2014CB340403), NSF of China (61632016, 61602488, U1711261, 61472198, 61521002, 61661166012, 61502503), and the Research Funds of Renmin University of China (18XNLG18, 18XNLG21).

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

10. REFERENCES

[1] A. Abad, M. Nabi, and A. Moschitti. Self-crowdsourcing training for relation extraction. In *ACL*, pages 518–523, 2017.

[2] C. M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.

[3] K. Bowman and L. Shenton. Parameter estimation for the beta distribution. *Journal of Statistical Computation and Simulation*, 43(3-4):217–228, 1992.

[4] C. Chai, G. Li, J. Li, D. Deng, and J. Feng. Cost-effective crowdsourced entity resolution: A partial-order approach. In *SIGMOD*, pages 969–984, 2016.

[5] K. W. Church and P. Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

[6] S. Das, P. S. G. C., A. Doan, J. F. Naughton, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, and Y. Park. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *SIGMOD*, pages 1431–1446, 2017.

[7] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[8] J. Fan and G. Li. Human-in-the-loop rule learning for data integration. *IEEE Data Eng. Bull.*, 41(2):104–115, 2018.

[9] J. Fan, G. Li, B. C. Ooi, K. Tan, and J. Feng. icrowd: An adaptive crowdsourcing framework. In *SIGMOD*, pages 1015–1030, 2015.

[10] J. Fan, M. Lu, B. C. Ooi, W. Tan, and M. Zhang. A hybrid machine-crowdsourcing system for matching web tables. In *ICDE 2014*, pages 976–987, 2014.

[11] J. Fan, M. Zhang, S. Kok, M. Lu, and B. C. Ooi. Crowdop: Query optimization for declarative crowdsourcing systems. *IEEE Trans. Knowl. Data Eng.*, 27(8):2078–2092, 2015.

[12] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *SIGMOD*, pages 61–72, 2011.

[13] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. W. Shavlik, and X. Zhu. Corleone: Hands-off crowdsourcing for entity matching. In *SIGMOD*, pages 601–612, 2014.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

[15] D. Haas, J. Wang, E. Wu, and M. J. Franklin. Clamshell: Speeding up crowds for low-latency data labeling. *PVLDB*, 9(4):372–383, 2015.

[16] R. Hoffmann, C. Zhang, X. Ling, L. Zettlemoyer, and D. S. Weld. Knowledge-based weak supervision for information extraction of overlapping relations. In *ACL*, pages 541–550. Association for Computational Linguistics, 2011.

[17] M. Joglekar, H. Garcia-Molina, and A. Parameswaran. Comprehensive and reliable crowd assessment algorithms. pages 195–206, 2014.

[18] A. R. Khan and H. Garcia-Molina. Attribute-based crowd entity resolution. In *CIKM*, pages 549–558, 2016.

[19] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger. From word embeddings to document distances. In *ICML 2015*, pages 957–966, 2015.

[20] Y. LeCun, Y. Bengio, and G. E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[21] G. Li. Human-in-the-loop data integration. *PVLDB*, 10(12):2006–2017, 2017.

[22] G. Li, C. Chai, J. Fan, X. Weng, J. Li, Y. Zheng, Y. Li, X. Yu, X. Zhang, and H. Yuan. CDB: optimizing queries with crowd-based selections and joins. In *SIGMOD*, pages 1463–1478, 2017.

[23] G. Li, J. Wang, Y. Zheng, and M. J. Franklin. Crowdsourced data management: A survey. *IEEE Trans. Knowl. Data Eng.*, 28(9):2296–2319, 2016.

[24] A. Liu, S. Soderland, J. Bragg, C. H. Lin, X. Ling, and D. S. Weld. Effective crowd annotation for relation extraction. In *NAACL HLT*, pages 897–906, 2016.

[25] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang. CDAS: A crowdsourcing data analytics system. *PVLDB*, 5(10):1040–1051, 2012.

[26] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Demonstration of qurk: a query processor for humanoperators. In *SIGMOD 2011*, pages 1315–1318, 2011.

[27] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[28] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[29] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL 2009*, pages 1003–1011, 2009.

[30] F. Parisi, F. Strino, B. Nadler, and Y. Kluger. Ranking and combining multiple predictors without labeled data. *Proceedings of the National Academy of Sciences of the United States of America*, 111(4):1253–8, 2014.

[31] H. Park, R. Pang, A. G. Parameswaran, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: A system for declarative crowdsourcing. *PVLDB*, 5(12):1990–1993, 2012.

[32] A. Ratner, S. H. Bach, H. R. Ehrenberg, J. A. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *PVLDB*, 11(3):269–282, 2017.

[33] A. J. Ratner, C. D. Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. In *NIPS 2016*, pages 3567–3575, 2016.

[34] B. Roth and D. Klakow. Combining generative and discriminative model scores for distant supervision. In *EMNLP*, pages 24–29, 2013.

[35] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *International Conference on Computer Vision*, page 59, 1998.

[36] V. S. Sheng, F. Provost, and P. G. Ipeirotis. Get another label? improving data quality and data

mining using multiple, noisy labelers. In *SIGKDD*, pages 614–622. ACM, 2008.

[37] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. *CoRR*, abs/1707.02968, 2017.

[38] S. Takamatsu, I. Sato, and H. Nakagawa. Reducing wrong labels in distant supervision for relation extraction. In *Meeting of the Association for Computational Linguistics: Long Papers*, pages 721–729, 2012.

[39] Y. Tong, L. Chen, Z. Zhou, H. V. Jagadish, L. Shou, and W. Lv. Slade: A smart large-scale task decomposer in crowdsourcing. *IEEE Transactions on Knowledge and Data Engineering*, 30(8):1588–1601, 2018.

[40] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen. Online mobile micro-task allocation in spatial crowdsourcing. In *ICDE*, pages 49–60, 2016.

[41] V. Verroios, H. Garcia-Molina, and Y. Papakonstantinou. Waldo: An adaptive human interface for crowd entity resolution. In *SIGMOD*, pages 1133–1148, 2017.

[42] N. Vesdapunt, K. Bellare, and N. N. Dalvi. Crowdsourcing algorithms for entity resolution. *PVLDB*, 2014.

[43] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 2012.

[44] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, pages 229–240, 2013.

[45] J. Wang, L. Yu, W. Zhang, Y. Gong, Y. Xu, B. Wang, P. Zhang, and D. Zhang. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *SIGIR*, pages 515–524. ACM, 2017.

[46] S. Wang, X. Xiao, and C. Lee. Crowd-based deduplication: An adaptive approach. In *SIGMOD*, pages 1263–1277, 2015.

[47] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, 2013.

[48] J. Yang, J. Fan, Z. Wei, G. Li, T. Liu, and X. Du. Cost-effective data annotation using game-based crowdsourcing. In *Technical Report*, 2018. `http://iir.ruc.edu.cn/~fanj/papers/crowdgame-tr.pdf`.

[49] Y. Zhang, X. Chen, D. Zhou, and M. I. Jordan. Spectral methods meet em: a provably optimal algorithm for crowdsourcing. In *International Conference on Neural Information Processing Systems*, pages 1260–1268, 2014.

[50] Y. Zheng, G. Li, Y. Li, C. Shan, and R. Cheng. Truth inference in crowdsourcing: Is the problem solved? *PVLDB*, 10(5):541–552, 2017.

[51] Y. Zheng, J. Wang, G. Li, R. Cheng, and J. Feng. QASCA: A quality-aware task assignment system for crowdsourcing applications. In *SIGMOD*, pages 1031–1046, 2015.