Human-in-the-loop Rule Learning for Data Integration

Ju Fan
Renmin University of China
fanj@ruc.edu.cn

Guoliang Li
Tsinghua University
liguoliang@tsinghua.edu.cn

Abstract

Rule-based data integration approaches are widely adopted due to its better interpretability and effective interactive debugging. However, it is very challenging to generate high-quality rules for data integration tasks. Hand-crafted rules from domain experts are usually reliable, but they are not scalable: it is time and effort consuming to handcraft many rules with large coverage over the data. On the other hand, weak-supervision rules automatically generated from machines, such as distant supervision rules, can largely cover the items; however, they may be very noisy that provide many wrong results. To address the problem, we propose a human-in-the-loop rule learning approach with high coverage and high quality. The approach first generates a set of candidate rules, and proposes a machine-based method to learn a confidence for each rule using generative adversarial networks. Then, it devises a game-based crowdsourcing framework to refine the rules, and develops a budget-constraint crowdsourcing algorithm for rule refinement at affordable cost. Finally, it applies the rules to produce high-quality data integration results.

1 Introduction

Despite that machine learning (ML)-based solutions are widely used for data integration tasks due to high effectiveness, the rule-based approach is often preferred as it provides better interpretability that enables interactive debugging. According to a survey [3] over 54 information extraction vendors, 67% of the tools are rule-based while only 17% are ML-based. In addition, Walmart applies rule-based approaches in their product classification to allow domain experts to improve the system [21]. For example, entity extraction aims to extract entity from textual data. The rules like "C `including` e" can be used to extract entities, e.g., `Canon 40D` and `Nikon D80`, for an entity type, say `camera`. In entity matching, it is usually necessary to generate blocking rules to discriminate entity pairs that cannot match.

There are two widely-used methods to construct rules: *hand-crafted* rules from domain experts and *weak-supervision* rules automatically generated by machines. Hand-crafted rules ask experts 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. Therefore, weak-supervision rules automatically generated by machines are introduced [19, 18], e.g., distant supervision rules in information extraction. Weak-supervision rules can largely cover the items; however, they may be very unreliable that provide wrong labels.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

For effective rule generation, we first generate a set of candidate rules using machine algorithms and then use a human-in-the-loop approach to select high-quality rules. There are two important objectives to select *high-quality* rules. 1) *high coverage*: select the rules that cover as many items as possible to label the data. 2) *high quality*: select the rules that induce few wrong labels on their covered items. It is easy to obtain the coverage of the rules but it is hard to get the quality of the rules, as the ground-truth of items covered by the rules is unknown. To address this problem, we propose a human-in-the-loop framework.

We first utilize a machine-based rule evaluation phase to provide a *coarse evaluation* on rule quality (Section 3): how likely a rule can correctly label items. A naïve evaluation method solely based on the training data is far from sufficient, as the training data is usually very limited. To address the problem, we propose to infer labels for the data items not included in the training set, and take inferred labels as the basis for rule quality estimation. We fulfill the label inference by leveraging a recently proposed learning mechanism, generative adversarial networks (GAN) [10, 23]. The idea is to learn a label generative model that captures what *feature characteristics* of items can generate a particular label, using an adversarial game process. Moreover, we also devise a recurrent neural networks (RNN) based encoder-decoder neural network model to realize the GAN mechanism, and the model can alleviate the tedious burden of feature engineering.

However, rule evaluation based on inferred data labels may not be accurate because the inference is computed based on the static training data and may not be effectively adapted to others. Thus, the framework devises an active crowdsourcing approach (Section 4) for rule refinement to further select a subset of "high-quality" rules, by leveraging the human intelligence from crowdsourcing. A straightforward method asks the crowd to answer a *rule validation* task, which asks the crowd to check whether a rule with high coarsely-evaluated precision is valid or invalid. However, the crowd may give low-quality answers for a rule validation task, because a rule may cover many items and the workers cannot examine all the items covered by the rule. To alleviate this problem, we can ask the crowd to answer an *item checking* task, which asks the crowd to give the label of an item and utilizes the result to validate/invalidate rules that cover the item. However it is expensive to ask many item checking tasks. To address this problem, we devise a two-pronged crowdsourcing task scheme that effectively combines these two task types. We also develop a crowdsourcing task selection algorithm to iteratively select rule-validation and item-checking tasks until the crowdsourcing budget is used up.

Finally, in the third phase, we use the selected rules to annotate data with labels (Section 5.1). The challenge here is that an item can be covered by rules with diverse quality or even with conflicting labels. For example, in entity extraction, one rule may provide label 1 while another one labels -1. To address this problem, we develop an *aggregation* approach to consolidate the rules and provide high-quality labels for the items.

2 Overview of Human-in-the-loop Rule Learning for Data Integration

Most of the data integration tasks can be formalized as the problem that assigns a set of data items with one of the two possible labels, -1 and 1. We also call items with label 1 (-1) positive (negative) items. Figure 1 illustrates two data integration tasks, *entity matching* and *entity extraction*. A more detailed entity matching example is shown in Figure 2: It aims to identify any pair of product records (Figure 2(a)) is matched (i.e., being the same product entity) or not. Here, we regard each item as a possible record pair, as shown in Figure 2(b). Moreover, entity extraction task aims at identifying the entities (e.g., `Canon 40D` and `Nikon D80`) that belong to a specific class (e.g., `camera`) from the unstructured text. Formally, given a target class, we can regard this task as a labeling problem that assigns an entity with 1 if the entity is of the class, or -1 otherwise.

Rule-Based Data Integration Approach. The rule-based approach is often preferred as it provides better interpretability that enables interactive debugging. More formally, a rule is defined as a function $r_j : \mathcal{E} \rightarrow \{L, \texttt{nil}\}$ that maps item $e_i \in \mathcal{E}$ 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 item 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 l_i denote the true label of e_i,

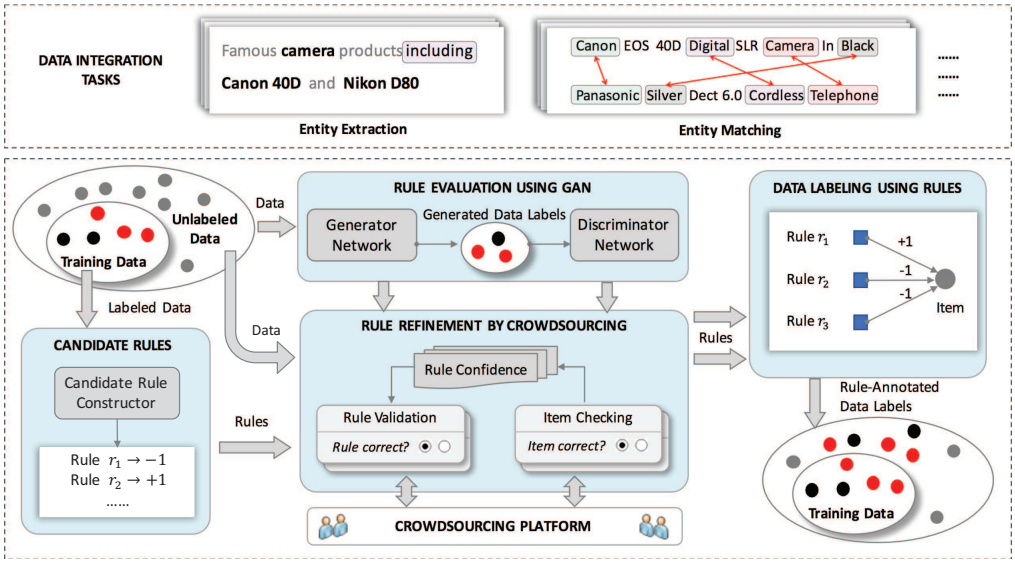


Figure 1: Framework of Human-in-the-loop Rule Learning.

There are two widely-used ways to generate rules for data integration: *hand-crafted* rules from domain experts and *weak-supervision* rules automatically generated by machines. 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. *Weak-supervision* rules automatically generated by machines are introduced [19, 18]. Weak-supervision rules can largely cover the items; however, some of them may be very unreliable that provide wrong labels.

Example 1 (Weak-Supervision Rule): For entity extraction, *distant supervision* [17] is commonly used to identify some textual patterns as *extraction rules* based on a small amount of known positive entities (label $+1$) and negative ones (label -1). For example, considering the text "famous camera products including Canon 40D" and a known "`camera`" entity `Canon 40D`, we may obtain a rule that "C `including` e" indicates that e is an entity of class C. For entity matching, *blocking rules* are also extensively used to eliminate the record pairs that could not be matched. For example, a camera from manufacture `Canon` cannot be matched with another one with `Nikon`, and thus we can use this rule to eliminate record pairs across `Canon` and `Nikon`.

Human-in-the-loop Rule Learning. Note that candidate rules, especially weak-supervision rules automatically generated by machines, could be unreliable. For instance, pattern "C `including` e" for entity extraction may not always indicate e is an entity of class C, and thus incur false positives. On the other hand, a bad blocking rule, such as using `color` to discriminate products, may introduce many false negatives.

To formalize rule quality, we introduce *rule confidence*, denoted by λ_j, of a rule r_j as the ratio of the items in $\mathcal{C}(r_j)$ correctly annotated with label L of r_j, i.e., $\lambda_j = \frac{\sum_{e_i \in \mathcal{C}(r_j)} \mathbb{1}_{\{l_i = L\}}}{|\mathcal{C}(r_j)|}$, where $\mathbb{1}_{\{l_i=L\}}$ is an indicator function that returns 1 if $l_i = L$ or 0 otherwise. Rule confidence is rather challenging to evaluate because most of the data items are unlabeled (denoted as gray circles in the figure).

To address this challenge, we introduce a human-in-the-loop framework, as shown in Figure 1. The framework takes a set of data items $E = \{e_1, e_2, \ldots, e_m\}$ as input, where a subset of data items is labeled with ground truth as the *training data*. In particular, we use black (red) circles to represent data items with label 1 (-1), and gray circles as unlabeled items. The framework also considers a set of candidate rules, which are constructed in either hand-crafted or weak supervision ways. To effectively label data items, it utilizes a rule learning approach that consists of the following three phases.

- *Phase I - Rule Evaluation using Generative Adversarial Networks (GAN):* This phase aims to utilize machine-based methods to provide a *coarse evaluation* of rule confidence. A straightforward way is to

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.

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

(b) items (record pairs).

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

(c) candidate blocking rules.

Figure 2: A running example for rule learning in entity matching task.

directly use the training data: a rule is of high confidence if it provides correct labels for most of the items in the training set, or vice versa. However, in most of the data integration tasks, the training set is usually far from sufficient to provide effective rule evaluation. To address this problem, our framework *infers* labels for the unlabeled items using a recently proposed learning approach, generative adversarial networks (GAN) [10, 23]. Intuitively, the objective is toward that one cannot easily distinguish whether an item is annotated with the true or inferred label. Then, fed with the inferred labels, it provides the coarse evaluation for rule quality. We will present the details of this phase in Section 3.

- *Phase II - Rule Refinement by Crowdsourcing:* Rule evaluation in the previous phase may not be accurate because the inference is computed based on the static training data and may not be effectively adapted to other items. We devise an active crowdsourcing approach for rule refinement that selects a subset of "high-quality" rules from the candidates. A straightforward method asks the crowd to answer a *rule validation* task, which asks the crowd to check whether a rule with high coarsely-evaluated confidence is valid or invalid. However, the crowd may give low-quality answers for a rule validation task, because a rule may cover many items and the workers cannot examine all the items covered by the rule. To alleviate this problem, we can ask the crowd to answer an *item checking* task, which asks the crowd to give the label of an item and utilizes the result to validate/invalidate rules that cover the item. However it is expensive to ask many item checking tasks. To address this problem, we devise a two-pronged crowdsourcing task scheme that effectively combines these two task types. We also develop a crowdsourcing task selection algorithm to iteratively select rule-validation and item-checking tasks until the crowdsourcing budget is used up. We discuss the techniques in this phase in Section 4.

- *Phase III - Data Labeling using Rules:* This phase applies a rule-based labeling model with the learned rule confidence to obtain "high-quality" item labels. Note that some items may not be labeled by rules either because no rules cover the items, or because the labels provided by rules with low confidence. Thus, it calls for an *aggregation* approach to consolidate the rules. We discuss the approach in Section 5.1.

Figure 2(c) illustrates some "blocking rules" for entity matching. Each rule, represented by two keywords, expresses how we discriminate the product pairs covered by the rule. For example, $r_1 : (\texttt{sony}, \texttt{apple})$ expresses the following blocking rule: any item, 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.

Example 2 (Human-in-the-loop Rule Learning for Entity Matching): We illustrate how the proposed framework works for the entity matching task in Figure 2. In the first phase, based on a small training set $\{e_5, e_6, e_8\}$ with known labels, the framework utilizes a GAN-based approach to infer labels for the unlabeled items. The intuition is to learn a generative model that describes "how is a matched pair like" from the training set, and then use the model to label the items. In the second phase, the framework can either ask the crowd to directly validate a rule, e.g., whether $(\texttt{laptop}, \texttt{notebook})$ is effective for blocking, or utilize the crowd to check items covered by rules. Given a crowdsourcing budget, the framework unifies these two task types to further refine rule quality evaluation. Finally, the third phase determines whether to use the rules, so as to balance the following

tradeoff: using more rules may improve the coverage, while increase the risk of incurring error labels. Finally, the framework outputs the labeled items as the result.

3 Machine-Based Rule Evaluation

Rule confidence evaluation heavily relies on data labels: a rule is of high confidence if it provides correct labels for most of the items it covers, or vice versa. However, in most data integration tasks, the amount of labeled items in the training data is usually very small, and thus evaluation solely using the training data is far from sufficient. To address this problem, we propose to infer labels for the items not included in the training data, and take this as the basis for rule confidence estimation. We fulfill item label inference by developing a GAN-based approach. The objective is to learn a *label generator* that captures what *feature characteristics* of items can generate a particular label (e.g., label 1). For example, records s_2 and s_3 in Figure 2(a) share representative words, `Apple` and `Pro`, which can be taken as feature characteristics to generate positive labels. To obtain a good generator, our approach introduces *label discriminator*, and lets it play an adversarial game with the label generator. On the one hand, the generator tries to identify the items with a particular label that look like the ones in our training data, so as to deceive the discriminator. On the other hand, the discriminator tries to make a clear distinction on which items are from the training set and which are generated by the generator. Next, we first present the formalization of our GAN-based learning model, and then introduce RNN-based neural networks that fulfill the model.

GAN-based learning mechanism. We formalize the label generator as a probabilistic distribution over set E that generates positive items, denoted by $p_G(e_i|\theta)$, where θ is the model parameters. Moreover, we use $p_{\texttt{true}}(e_i)$ to represent the ground-truth distribution of positive items in $\mathcal{E}$. Among all items, the larger the $p_{\texttt{true}}(e_i)$ ($p_G(e_i|\theta)$) is, the more likely that e_i can be sampled as a true (generated) positive item. Naturally, the objective of model learning is to compute a good θ that approximates generator $p_G(e_i|\theta)$ to the ground-truth $p_{\texttt{true}}(e_i)$: the positive items that our generator generates are almost the same as the true positive items.

We utilize the generative adversarial networks to effectively learn parameters θ. We introduce a discriminator model $D(e_i)$ as a binary classifier that determines whether e_i is a true positive item. More specifically, $D(e_i)$ outputs a probability that e_i is a true positive item instead of a generative one. The generator and discriminator play an adversarial game as follows. On the one hand, the generator samples items from $\mathcal{E}$ according to the current distribution $p_G(e_i|\theta)$, and mixes the sampled items with the true positive ones in the training data. On the other hand, the discriminator tries to identify which ones in the mixed item set are true positive items. Formally, this can be formalized as a minimax game with a value function $V(G, D)$.

$$\min_G \max_D V(G, D) = \mathbb{E}_{e_i \sim p_{\texttt{true}}(e_i)}[\log D(e_i)] + \mathbb{E}_{e_i \sim p_G(e_i|\theta)}[1 - \log D(e_i)], \tag{5}$$

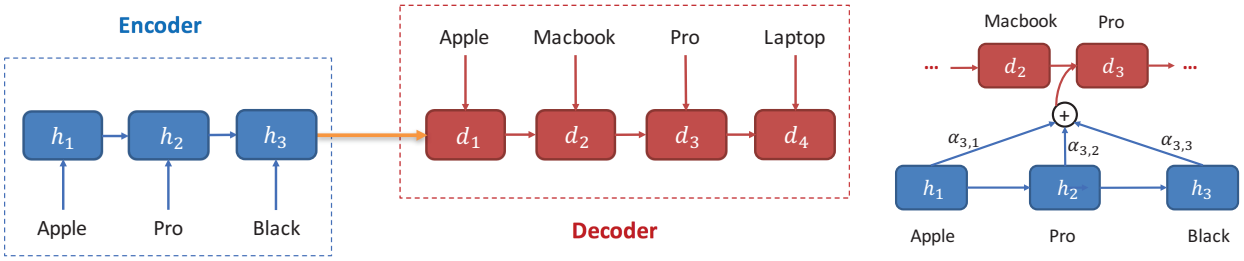
where G and D are the generator and the discriminator model respectively. We defer the discussion on how to derive them later, and focus on presenting how generator and discriminator optimize their objectives now.

Discriminator plays as a *maximizer*: given the current $p_G(e_i)$, it tries to train $D(e_i)$ by maximizing the probability that assigns correct label for both training and generative positive items, i.e.,

$$D^* = \arg\max_D \big(\mathbb{E}_{e_i \sim p_{\texttt{true}}(e_i)}[\log D(e_i)] + \mathbb{E}_{e_i \sim p_G(e_i|\theta)}[1 - \log D(e_i)]\big). \tag{6}$$

Generator plays as a *minimizer*: given the current $D(e_i)$, it tries to deceive the discriminator to believe that a generated e_i is a true positive item. To this end, it trains G by optimizing the following objective

$$\begin{aligned} G^* &= \arg\min_G \big(\mathbb{E}_{e_i \sim p_{\texttt{true}}(e_i)}[\log D(e_i)] + \mathbb{E}_{e_i \sim p_G(e_i|\theta)}[1 - \log D(e_i)]\big) \\ &= \arg\max_\theta \big(\mathbb{E}_{e_i \sim p_G(e_i|\theta)}[\log D(e_i)]\big) \end{aligned} \tag{7}$$



(a) An encoder-to-decoder mode (ENC-DEC). (b) ENC-DEC with attention.

Figure 3: A RNN-based encoder-decoder neural network model for entity matching.

To learn the parameters of generator and discriminator, we can iteratively use the optimization principle, such as stochastic gradient descent (SGD) used in the existing works [10, 23].

RNN-based encoder-decoder model. To apply the GAN mechanism, we need to derive both generative distribution $p_G(e_i|\theta)$ and the discriminator $D(e_i)$. For $p_G(e_i|\theta)$, we can use a method of first computing the likelihood $G(e_i, \theta)$ that e_i is a positive item given model θ, and then applying a softmax function, i.e.,

$$p_G(e_i|\theta) = \frac{\exp(G(e_i, \theta))}{\sum_{e' \in E} \exp(G(e', \theta))} \tag{8}$$

Then, we design neural networks that fulfill both $G(e_i, \theta)$ and $D(e_i)$. We introduce a recurrent neural networks (RNN) based encoder-decoder model [4], as illustrated in Figure 3. For ease of presentation, we use the entity matching task to illustrate how it works. Intuitively, given an item e_i with record pair (s_{i_1}, s_{i_2}), the model computes the probability that s_{i_2} can be "translated" from s_{i_1}. To this end, we model s_{i_1} as a sequence of words, $\mathbf{x} = \{x_1, x_2, \ldots, x_{|s_{i_1}|}\}$, where x_t is an embedding representation (i.e., numeric vector) of the t-th word in the sequence. Similarly, we model s_{i_2} as a sequence of words, $\mathbf{y} = \{y_1, y_2, \ldots, y_{|s_{i_2}|}\}$. For simplicity, we introduce the model for computing $G(e_i, \theta)$, and it can also be applied to compute $D(e_i)$.

We employ a Recurrent Neural Network (RNN) composed by Long Short-Term Memory (LSTM) units to realize the encoder, where each LSTM unit $\mathbf{h}_t$ encodes the *hidden state* of the t-th word in s_{i_1}, i.e., $\mathbf{h}_t = f_{\texttt{LSTM}}(x_t, \mathbf{h}_{t-1})$, where $f_{\texttt{LSTM}}$ denotes the output of the LSTM neural network (we omit its derivation due to the space limit. Please refer to [9] for more details).

The decoder takes the encoded hidden states $\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{|s_{i_1}|}\}$ as input, and computes $G(e_i, \theta)$ as the joint probability $G(e_i, \theta) = P(\mathbf{y}|\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{|s_{i_1}|})$ of "decoding" $\mathbf{y}$ from the hidden states. A simple way is to also employ an RNN composed by LSTM units (Figure 3(a)), where each unit is the hidden state of decoding word y_t in s_{i_2}. Then, we compute the joint probability by decomposing it into a sequence of conditional probabilities

$$P(\mathbf{y}|\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{|s_{i_1}|}) = \prod_{t=1}^{|s_{i_2}|} P(y_t|\mathbf{y}_{<t}), \tag{9}$$

where $\mathbf{y}_{<t}$ is the "context" words before y_t, i.e., $\mathbf{y}_{<t} = \{y_1, \ldots y_{t-1}\}$. To obtain $P(y_t|\mathbf{y}_{<t})$, we let $\mathbf{d}_0 = \mathbf{h}_{|s_{i_1}|}$, and then compute

$$P(y_t|\mathbf{y}_{<t}) = \texttt{Softmax}(\mathbf{d}_t); \quad \mathbf{d}_t = f_{\texttt{LSTM}}(y_t, \mathbf{d}_{t-1}) \tag{10}$$

where $\texttt{Softmax}$ is the softmax neural-network layer that maps hidden state $\mathbf{d}_t$ to a probabilistic distribution on the word vocabulary. We may also extend the decoding process with *attention* mechanism, as illustrated in Figure 3(b). The idea is to consider different importance of the encoded hidden states $\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{|s_{i_1}|}\}$ when decoding word y_t by introducing a weight $\alpha_{t,r}$, i.e.,

$$\tilde{\mathbf{d}}_t = \sum_{r=1}^{|s_{i_1}|} \mathbf{h}_r \cdot \alpha_{t,r} = \sum_{r=1}^{|s_{i_1}|} \mathbf{h}_r \frac{\exp(\mathbf{d}_t \cdot \mathbf{h}_r)}{\sum_{r'} \exp(\mathbf{d}_t \cdot \mathbf{h}_{r'})} \tag{11}$$

Based on this, the decoder computes the conditional probability as $P(y_t|\mathbf{y}_{<t}) = \texttt{Softmax}([\mathbf{d}_t; \tilde{\mathbf{d}}_t])$, where $[\mathbf{d}_t; \tilde{\mathbf{d}}_t]$ is a concatenation of hidden states $\mathbf{d}_t$ and $\tilde{\mathbf{d}}_t$.

Example 3 (Machine-based rule evaluation using GAN and ENC-DEC): Figure 3 provides examples of the encoder-decoder model. As shown in Figure 3(a), the model uses LSTM units to encode "Apple Pro Black" into hidden states $\mathbf{h}_1$ to $\mathbf{h}_3$, and then uses another set of LSTM units to compute the probability of decoding "Apple Macbook Pro Laptop" from the hidden states. Figure 3(b) illustrates the attention-based model. For instance, for decoding word `Pro`, it considers weights $\alpha_{3,1}$ to $\alpha_{3,3}$. Intuitively, weight $\alpha_{3,3}$ would be smaller than $\alpha_{3,2}$, as `Pro` is not very relevant to word `Black`. We use this encoder-decoder model for computing $G(e_i, \theta)$ and $D(e_i)$, and then apply the GAN mechanism on the training set (e.g., the items with blue color in Figure 2(b)). Intuitively, given a true positive item $e_5 = (s_2, s_3)$, we may use GAN to generate another positive item $e_7 = (s_2, s_5)$. Finally, based on the generated labels, we compute rule confidence for machine-based rule evaluation.

4 Rule Refinement by Game-Based Crowdsourcing

The generative model $G(e_i, \theta)$ learned in the previous section can be used as an estimate of the probability that e_i is a positive item. Then, given a rule r_j with label L, we can estimate its confidence, denoted by $\hat{\lambda}_j$ as

$$\hat{\lambda}_j = \begin{cases} \frac{\sum_{e_i \in \mathcal{C}(\mathcal{R})} G(e_i,\theta)}{|\mathcal{C}(\mathcal{R})|}, & L = 1 \\ 1 - \frac{\sum_{e_i \in \mathcal{C}(\mathcal{R})} G(e_i,\theta)}{|\mathcal{C}(\mathcal{R})|}, & L = -1 \end{cases} \tag{12}$$

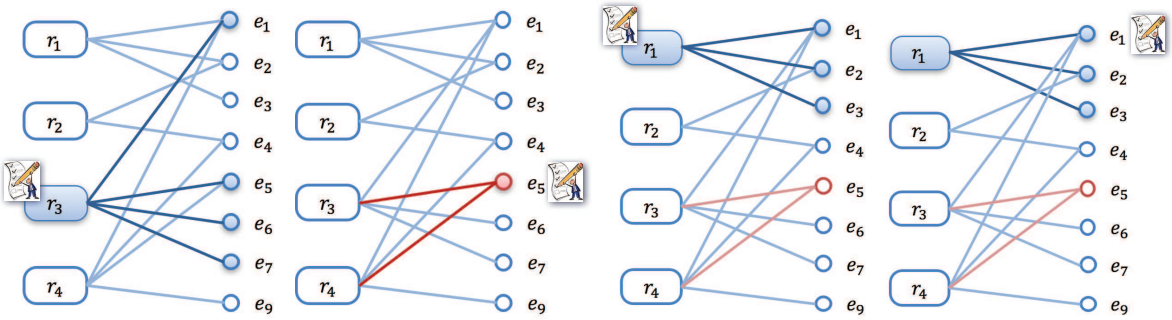
Based on Equation (12), a simple way for rule refinement is to select rules with larger confidence estimates. However, despite of using GAN and RNN, the estimation may not be accurate enough, because the training set is static and thus the inferred labels may not be adapted to other items. Thus, this section presents our active crowdsourcing approach named CROWDGAME for more effective rule refinement.

4.1 Overview of CrowdGame

The objective of CROWDGAME is to leverage the human intelligence in crowdsourcing for selecting high-confidence rules. A naïve crowdsourcing approach is to refine the estimation in Equation (12) by sampling some items covered by a rule, checking them through crowdsourcing, and updating the estimation. However, as the crowdsourcing budget, i.e., the affordable number of tasks, is usually much smaller than item set size, such "aimless" item checking without focusing on specific rules may not be effective.

Two-pronged task scheme. To address the problem, we devise a two-pronged crowdsourcing task scheme that first leverages the crowd to directly validate a rule and then applies item checking tasks on validated rule. To this end, we introduce another type of task, *rule validation*. For example, a rule validation task asks the crowd to validate whether rule r_1 (`sony`, `apple`) in Figure 2(c) is good at discriminating products. 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. Thus, the fine-grained item checking tasks are also indispensable.

Crowdsourcing task selection. Due to the fixed amount of crowdsourcing budget, there is usually a tradeoff between the rule-validation tasks and item-checking tasks. On the one hand, assigning more budget on rule validation will lead to fewer item checking tasks, resulting in less accurate evaluation on rules. On the other hand, assigning more budget on item checking, although being more confident on the validated rules, may miss the chance for identifying more good rules.



(a) RULEGEN (1st round). (b) RULEREF (1st round). (c) RULEGEN (2nd round). (d) RULEREF (2nd round).

Figure 4: Illustration of game-based crowdsourcing.

We address the challenge by employing two groups of crowd workers from a crowdsourcing platform: one group answers rule validation tasks to play a role of *rule generator* (RULEGEN), while the other group answers item checking tasks to play a role of *rule refuter* (RULEREF). Then, we unify these two worker groups by letting them play a game. On the one hand, RULEGEN tries to elect some rules $\mathcal{R}$ from the candidates $\mathcal{R}^{\mathsf{C}}$ for crowdsourcing via rule validation tasks, such that the elected rules can not only cover more items but also incur less errors. It is easy to examine the coverage of a rule. For the second factor, we take the machine-based rule estimation in Equation (12) to evaluate whether a rule may incur less errors. On the other hand, RULEREF tries to refute its opponent RULEGEN by checking some items that provide enough evidence to "reject" more rules in $\mathcal{R}$. Note that, although also using a game-based strategy, the above process is essentially different from GAN, as it focuses on task selection instead of model (parameters) learning.

Example 4 (Crowdsourced Rule Refinement): Figure 4 shows how CROWDGAME works under a crowdsourcing budget with 4 affordable tasks, which is like a round-based board game between two players. For simplicity, we apply an extreme refuting strategy that one counter-example is enough to refute all rules covering the item, and we consider all rules have the same estimated confidence obtained from Equation (12). In the first round, RULEGEN selects r_3 for rule validation, as it has a large item coverage. However, its opponent RULEREF finds a "counter-example" e_5 using an item checking task. Based on this, RULEREF refutes both r_3 and r_4 and rejects their covered items to maximize the loss. Next in the second round, RULEGEN selects another crowd-validated rule r_1, while RULEREF crowdsources e_1, knows e_1 is correctly labeled, and finds no "evidence" to refute r_1. As the budget is used up, we find $\mathcal{R}_2 = \{r_1\}$ as the refined rule set.

4.2 Crowdsourcing Task Selection

Formalization of rule set loss. Intuitively, We want to select rules with two objectives. 1) high coverage: we would like to select the rules that cover as many items as possible. 2) high confidence: we also prefer the rules that induce few wrong labels on their covered items. For simplicity, we focus on the case that there is only one kind of label L provided by the rules, e.g., $L = -1$ for blocking rules in entity matching.

We formalize the objectives by defining the *loss* of a rule set $\Phi(\mathcal{R})$ as the following combination of the number of uncovered items $|\mathcal{E}| - |\mathcal{C}(\mathcal{R})|$ and the expected number of errors $\sum_{e_i \in \mathcal{C}(\mathcal{R})} P(l_i \neq L)$: $\Phi(\mathcal{R}) = \gamma(|\mathcal{E}| - |\mathcal{C}(\mathcal{R})|) + (1-\gamma) \sum_{e_i \in \mathcal{C}(\mathcal{R})} P(l_i \neq L)$, where γ is a parameter between $[0, 1]$ to balance the preference among coverage and quality. For example, let us consider two rule sets $\mathcal{R}_1 = \{r_1\}$ and $\mathcal{R}_2 = \{r_1, r_3\}$. $\mathcal{R}_1$ covers three items without any errors, while $\mathcal{R}_2$ covers more (6 items) with wrongly labeled items (e_5 and e_7). Note that entity matching prefers quality over coverage on the blocking rules, and thus one needs to set a small parameter γ, say $\gamma = 0.1$ to achieve this preference. Obviously, under this setting, we have $\Phi(\mathcal{R}_1) < \Phi(\mathcal{R}_2)$.

It is non-trivial to estimate $P(l_i \neq L)$. To solve the problem, we use rule confidence to quantify the probability $P(l_i \neq L)$. Moreover, we extend the notation $\hat{\lambda}_j$ to $\hat{\lambda}_j(\mathcal{E}_{\mathsf{q}})$, which denotes an estimator of λ_j for rule r_j based on a set $\mathcal{E}_{\mathsf{q}}$ of items checked by the crowd, and $\hat{\Lambda}^{\mathcal{R}}(\mathcal{E}_{\mathsf{q}}) = \{\hat{\lambda}_j(\mathcal{E}_{\mathsf{q}})\}$ is the set of estimators,

each of which is used for evaluating an individual rule $r_j \in \mathcal{R}$. We will discuss how to estimate $\hat{\lambda}_j(\mathcal{E}_\mathtt{q})$ later. Let $\mathcal{R}^i \subseteq \mathcal{R}$ denote the set of rules in $\mathcal{R}$ covering item 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 $P(l_i = L)$ as $P(a_i)$ if the context is clear. Then, we introduce $\Phi(\mathcal{R}|\mathcal{E}_\mathtt{q})$ to denote the *estimated* loss based on a set $\mathcal{E}_\mathtt{q}$ of items checked by the crowd, i.e.,

$$\begin{aligned}\Phi(\mathcal{R}|\mathcal{E}_\mathtt{q}) &= \gamma(|\mathcal{E}| - |\mathcal{C}(\mathcal{R})|) + (1-\gamma) \sum_{e_i \in \mathcal{C}(\mathcal{R})} 1 - P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_\mathtt{q})) \\ &= \gamma|\mathcal{E}| - (1-\gamma) \sum_{e_i \in \mathcal{C}(\mathcal{R})} \{P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_\mathtt{q})) - \frac{1-2\gamma}{1-\gamma}\} \end{aligned} \tag{13}$$

The key in Equation 13) is $P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_\mathtt{q}))$, which captures our *confidence* about whether $l_i = L$ (e_i is correctly labeled) given the observations that e_i is covered by rule $\mathcal{R}_i$ with confidence $\Lambda^{\mathcal{R}^i}(\mathcal{E}_\mathtt{q})$. We discuss how to compute $P(a_i|\hat{\Lambda}^{\mathcal{R}^i}(\mathcal{E}_\mathtt{q}))$ in our technical report [24].

Minimax optimization task selection. Now, we are ready to formalize a minimax objective in the aforementioned RULEGEN-RULEREF game. Let $\mathcal{R}_\mathtt{q}$ and $\mathcal{E}_\mathtt{q}$ respectively denote the sets of rules and items, 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

$$\mathcal{O}^{\mathcal{R}^*_\mathtt{q},\mathcal{E}^*_\mathtt{q}} = \min_{\mathcal{R}_\mathtt{q}} \max_{\mathcal{E}_\mathtt{q}} \Phi(\mathcal{R}_\mathtt{q}|\mathcal{E}_\mathtt{q}) \iff \max_{\mathcal{R}_\mathtt{q}} \min_{\mathcal{E}_\mathtt{q}} \sum_{e_i \in \mathcal{C}(\mathcal{R}_\mathtt{q})} \{P(a_i|\hat{\Lambda}^{\mathcal{R}_i}(\mathcal{E}_\mathtt{q})) - \frac{1-2\gamma}{1-\gamma}\} \tag{14}$$

such that task numbers $|\mathcal{R}_\mathtt{q}| + |\mathcal{E}_\mathtt{q}| \leq k$.

Based on Equation (14), we can better illustrate the intuition of CROWDGAME. Overall, CROWDGAME aims to find the optimal task sets $\mathcal{R}^*_\mathtt{q}$ and $\mathcal{E}^*_\mathtt{q}$ from the choices with constraint $|\mathcal{R}_\mathtt{q}| + |\mathcal{E}_\mathtt{q}| \leq k$. RULEGEN prefers to validate rules with large coverage and high confidence to minimize the loss. On the contrary, RULEREF aims to check items which are helpful to identify low-confidence rules that cover many items, so as to effectively maximize the loss. These two players iteratively select tasks until all crowdsourcing budget is used up.

Our minimax objective is fulfilled by asking crowdsourcing tasks, as both $\mathcal{R}_\mathtt{q}$ and $\mathcal{E}_\mathtt{q}$ defined in Equation (14) depend on the crowd answers. We develop an *iterative crowdsourcing* algorithm. The algorithms takes as input a candidate rule set $\mathcal{R}^\mathtt{C}$, an item set $\mathcal{E}$, and a crowdsourcing budget k. It produces the generated rule set $\mathcal{R}_\mathtt{q}$ as output. Overall, the algorithm runs in iterations until k tasks have been crowdsourced, where each iteration consists of a RULEGEN step and a RULEREF step. RULEGEN selects rule-validation tasks. We apply a commonly-used *batch crowdsourcing mode*: We select a b-sized rule set $\mathcal{R}^{(t)}_\mathtt{q}$ that achieves the maximization in Equation (14) at the most. We can prove that the problem of selecting such a rule set is NP-hard, and design an approximate algorithm with theoretical bound to solve it [24]. Similarly, RULEREF selects a batch of b item checking tasks $\mathcal{E}^{(t)}_\mathtt{q}$, so as to achieve the minimization.

Rule Confidence Updating Model. To achieve minimax task selection, we need to estimate rule confidence. Initially, we can use the machine-based estimation in Equation (12) as *prior*. Then, we can use item checking crowdsourcing tasks to continuously update the estimation as more items are checked. The key challenge is how to effectively integrate the crowd's evaluation from rule validation and item checking tasks into the prior. To address this problem, we utilize the *Bayesian estimation* technique [2]. The basic idea is that we use the crowd results as "data observation" to adjust the prior, so as to obtain a *posterior* of rule confidence.

Example 5 (Crowdsourcing Task Selection): Consider the example in Figure 4 with $k = 4$ and $b = 1$. In the 1st iteration, RULEGEN and RULEREF respectively select r_3 and e_5 for crowdsourcing. Based on the crowdsourcing answers, the algorithm updates confidence estimates and continues to the 2rd iteration as shown in Figures 4(c) and 4(d). After these two iterations, the budget is used up, and the algorithm returns $\mathcal{R}_\mathtt{q} = \{r_1\}$.

5 Discussions

5.1 Data Labeling using Rules

This section discusses how we annotate an item e_i given a set $\mathcal{R}_i$ of rules with the estimated confidence $\Lambda^{\mathcal{R}^i} = \{\hat{\lambda}_j\}$ that covers e_i. We first present how to extend crowdsourced rule refinement to more general labeling rules, and then discuss some methods for rule aggregation.

Extension of labeling rules. We discuss a more general case that some rules in the candidates $\mathcal{R}^{\mathtt{C}}$ annotate label -1 (called negative rules for simplicity) while others annotate 1 (called positive rules). Consider our entity extraction example that annotates 1 if an entity belongs to a specific class or -1 otherwise. In this case, an item, e.g., an entity-class pair $(\texttt{Canon 40D}, \texttt{camera})$ could be covered by conflicting rules (textual patterns), e.g., a L_2 rule "`camera` *such as* `Canon 40D`" and a L_1 rule "`cameras` *not including* `Canon 40D`". We devise a simple extension by taking the refinement of positive and negative rules *independently*. More specifically, we split the crowdsourcing budget into two parts: one for positive rules and the other for negative ones. Then, we apply the techniques described in Section 4 to separately learn positive/negative rules using crowdsourcing.

Rule label aggregation methods. After the above rule refinement, an item e_i is now covered by conflicting rules, i.e., a set $\mathcal{R}_i^+$ of positive rules and a set $\mathcal{R}_i^-$ of negative rules. We determine label of e_i based on these two rule set, following the two steps below.

1) Low-confidence rule elimination: We prune rules with $\hat{\lambda}_j < \frac{1-2\gamma}{1-\gamma}$ as they are useless as shown in Equation (14). For example, considering a setting that $\gamma = 0.1$, we would not consider rules with $\hat{\lambda}_j \leq 0.89$, and they would increase the overall loss. As a result, if all rules covering item e_i satisfying the pruning criterion, we leave e_i unlabeled, as no candidate rule is capable of labeling the item.

2) Confidence-aware label aggregation: After the elimination step, if $\mathcal{R}_i^+$ or $\mathcal{R}_i^-$ of item e_i is not empty, an label aggregation method should be applied. A simple method is to simply follow the label of the rule with the maximum estimated confidence, i.e., $r^* = \arg_{r_j \in \mathcal{R}_i^+ \cup \mathcal{R}_i^-} \max \{\hat{\lambda}_j\}$. A more sophistication aggregation method is to apply weighted majority voting [6], where rule confidence is taken the weight in voting. We may also apply a discriminative deep learning model (e.g., multi-layer perceptron, MLP) that takes rules in $\mathcal{R}_i^+$ and $\mathcal{R}_i^-$ as input and trains the aggregation weights, like the existing works [19, 18].

5.2 Candidate Rule Generation for Different Data Integration Tasks

Our human-in-the-loop approach can be easily applied to different data integration tasks. The key of the application is to devise task-specific methods for generating candidate rules. Next, we discuss some representative data integration tasks as follows.

Entity matching task. For entity matching, we want to generate candidate *blocking rules* annotating label -1 to record pairs. Although blocking rules are extensively studied (see a survey [5]), 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 2(c), keyword pairs, such as $(\texttt{Canon}, \texttt{Panasonic})$ and $(\texttt{Camera}, \texttt{Telephone})$, 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.

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 intuition, we utilize word embedding based techniques [15, 16], which are good at capturing semantic similarity among words. We first 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

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

pairs from each record pair (s_a, s_b) using the Word Mover's Distance (WMD) [14]. WMD optimally aligns 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 [20] for details). Figure 1 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.

Relation extraction task. 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 a sentence "Kerry Robles was living in Mexico City with her husband Damien". We can use 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 the *distant supervision* [17], 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 entity pair (`Kerry Robles`, `Damien`), we identify words around these entities, e.g., `living`, `Mexico City` and `husband` between them (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.

Schema matching task. Schema matching aims to discover the semantic correspondences between the columns of two tables [7]. For example, consider a table A with address information `building` and `street`, and another table B with columns `bld` and `st`. Schema matching aims to find column correspondences, (A.`building`, B.`bld`) and (A.`street`, B.`st`). In this task, we can generate candidate rules using both schema- and instance-level information. For schema-level, we can learn string transformation rules from some examples of matched column names, such as (`street`, `st`) using techniques in [1]. For instance-level, we can devise different similarity functions, such as Jaccard, edit distance, and etc., over entity sets of any two columns, e.g., if the Jaccard similarity between the sets is larger than a threshold (e.g., 0.8), the two columns are matched.

Data repairing task. There are many data repairing rules, such as editing rules [8], fixing rules [22], Sherlock rules [13], similarity-based rules [11] and knowledge-based rules [12]. However these rules are data dependent, and they usually work well for some datasets. To produce high-quality, high-coverage, and general rules, we can mix these rules together and use our techniques to effectively select high-quality rules.

6 Conclusion and Future Works

In this paper, we present how to learn high-quality rules for data integration. We first discuss how to generate a set of candidate rules, and then propose a GAN-based method to learn a confidence for each rule. Next we devise a game-based crowdsourcing framework to refine the rules. Finally, we discuss how to apply the rules to address data integration problems.

Next we discuss some research challenges in rule learning for data integration. First, there are many machine learning techniques, e.g., deep learning and reinforcement learning, but they are hard to interpret. So a challenge is how to use machine learning techniques to generate explainable rules. Second, most rules are data dependent and it is hard to transfer the rules from one dataset to another. Thus a challenge is to devise a transfer-learning based method that can apply the knowledge of rule learning on one dataset to another dataset. Third, most rules rely on human knowledge, e.g., synonyms and common senses. A challenge is to build a common-sense base and utilize the common senses to help data integration.

References

[1] A. Arasu, S. Chaudhuri, and R. Kaushik. Learning string transformations from examples. *PVLDB*, 2(1):514–525, 2009.

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

[3] L. Chiticariu, Y. Li, and F. R. Reiss. Rule-based information extraction is dead! long live rule-based information extraction systems! In *ACL*, pages 827–832, 2013.

[4] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, pages 1724–1734, 2014.

[5] 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.

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

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

[8] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *VLDB J.*, 21(2):213–238, 2012.

[9] F. A. Gers, J. Schmidhuber, and F. A. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000.

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

[11] S. Hao, N. Tang, G. Li, J. He, N. Ta, and J. Feng. A novel cost-based model for data repairing. *IEEE Trans. Knowl. Data Eng.*, 29(4):727–742, 2017.

[12] S. Hao, N. Tang, G. Li, and J. Li. Cleaning relations using knowledge bases. In *ICDE*, pages 933–944, 2017.

[13] M. Interlandi and N. Tang. Proof positive and negative in data cleaning. In *ICDE*, 2015.

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

[15] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[16] 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.

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

[18] 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.

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

[20] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *ICCV*, pages 59–66, 1998.

[21] P. Suganthan, C. Sun, K. Gayatri, H. Zhang, F. Yang, N. Rampalli, S. Prasad, E. Arcaute, G. Krishnan, R. Deep, V. Raghavendra, and A. Doan. Why big data industrial systems need rules and what we can do about it. In *SIGMOD*, pages 265–276, 2015.

[22] J. Wang and N. Tang. Towards dependable data repairing with fixing rules. In *SIGMOD*, 2014.

[23] 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, 2017.

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