Towards User-Aware Rule Discovery

Chapter *in* Communications in Computer and Information Science · October 2017

DOI: 10.1007/978-3-319-68282-2_1

2 authors, including:

Paolo Papotti
EURECOM
101 PUBLICATIONS **1,506** CITATIONS

SEE PROFILE

Towards User-Aware Rule Discovery

Venkata Vamsikrishna Meduri and Paolo Papotti

Arizona State University
{vmeduri,ppapotti}@asu.edu

Abstract. Rule discovery is a challenging but inevitable process in several data centric applications. The main challenges arise from the huge search space that needs to be explored, and from the noise in the data, which makes the mining results hardly useful. While existing state-of-the-art systems pose the users at the beginning and the end of the mining process, we argue that this paradigm must be revised and new rule mining algorithms should be developed to let the domain experts interact during the discovery process. We discuss how new systems that embrace this approach overcome current limitations and ultimately result in shorter time and smaller user effort for rule discovery.

1 Introduction

Rule discovery from data is of utmost importance given the applicability of rules in several real world data-centric applications such as cleaning [13, 25, 32], fraud detection [4, 5], cybersecurity [22, 28], smart cities [15, 20], and database design [19, 21, 34]. While for many of these applications machine learning (ML) approaches have been designed, rules are still extremely popular in the industry [32]. In fact, rules are a favored choice to encode the background knowledge of the domain experts and ultimately take decisions over data. For example, financial services corporations manually create corpora of thousands of rules to identify fraudulent credit card transactions [24]. The main advantages of rule based approaches include the ability to work without training data, the possibility to debug them by non-experts, the potential to handle specialized infrequent patterns, and the semantically explainable and interpretable output [32].

In this work we focus on rules for data quality. We are interested in rules that go beyond traditional association rules [3], both in terms of complexity of the rule language and in terms of supported data models. Consider a scenario with credit card transactions by a customer, as shown in Figure 1. A domain expert states that if there are two transactions from the same card and the same merchant, the transaction IDs must be consistent with the timestamps, or the transactions should be manually reviewed. In other words, if a transaction has a higher id than another one that is registered later, there must be a problem.

In this example, the rule is triggered because record T_2 for transaction with ID "XX216" is registered before transaction "XX214" in T_1. The rule can be formally stated by using the formalization of *Denial Constraints* (DCs) [10] as

$$\forall T_\alpha, T_\beta \in R, \neg(T_\alpha.Merchant = T_\beta.Merchant \wedge T_\alpha.CreditCard = T_\beta.CreditCard \wedge T_\alpha.TransID > T_\beta.TransID \wedge T_\alpha.Time < T_\beta.Time)$$

where the universal variable T_α and T_β over the records are used to define predicates that, if true at the same time, trigger the rule.

	TransID	ItemId	Merchant	CreditCard	Time
T_1	XX214	17683	PayPal	XXX7038	10:35:02
T_2	XX216	43266	PayPal	XXX7038	10:34:43

Fig. 1. Credit card transactions.

However, discovering rules is a difficult exercise. Current approaches for rule discovery treat the algorithm as a black box, where the users are only engaged in the definition of the input parameters, such as the minimum support to consider a rule valid, and in the evaluation of the ultimate output. Unfortunately, these design choices make such approaches hard to use in real-world scenarios for three main reasons.

1. In the input definition step, several parameters strongly impact the final output, but are very hard to set upfront. Examples of such parameters include the percentage of tolerance to noise to discover approximate rules, or the way to select constants to be considered for rule discovery. These parameters are rarely known apriori, but tuning them with a trial-and-error approach is infeasible, given the large number of possible value combinations and the long running execution times for the mining, as discussed next.
2. Complexity in the mining of the rules comes from both the size of the schema and the size of the data. The schema complexity is exponential in the number of attributes, as all combinations of attributes must be tested [10]. For example, for the transactions example presented in Figure 1, the rule may also need to involve attribute ItemId. Moreover, if the language supports complex pairwise rules, such as denial constraints or de-duplication rules, the complexity is quadratic over the number of tuples.
3. In the output consumption, the number of rules that hold over the data is usually large, especially when constants and approximate rules are allowed, as they are often needed in practice. Moreover, when tolerance to noise in the data is required, semantically valid rules are mixed with incorrect rules because of dirty values in the data. This problem is alleviated by pruning mechanisms and ranking, but ultimately it leads to a large amount of time spent by the domain experts to identify the valid rules among the thousands that hold on the data.

In addition to these shortcomings, we should consider the limits in terms of the expressive power in the existing solutions. Popular rules expressed with ETL or procedural languages employ *User Defined Functions* (UDFs) to specify

special comparisons among values or complex look up functions. These specialized functions lead to more powerful rules, but the discovery of the right function is hard, as it is domain and dataset specific. Consider Temporal Functional Dependencies (TFDs), which are FDs that hold only during a certain time period, e.g., "the same person cannot be in two different places at the same time" [1,31]. Discovering the appropriate functions (the absolute time difference between timestamps) from a given library of UDFs [30,33] and the correct temporal values (the correct "same time" duration) is hard. These hard discovery cases lead to new input parameters, longer execution times, and larger number of output rules, thus exacerbating the three problems listed above.

We believe that the most promising way to overcome the limits of traditional approaches is to rethink discovering algorithms to make them *user-aware*. This change of perspective should cover all the aspects mentioned above: a more natural input for the users, incremental efficient algorithms to enable interactive response time, and a simpler and more effective way to identify the useful rules. As an orthogonal dimension, the new systems should also embrace a language that can use libraries of user-defined functions, in order to discover more powerful rules. In order to prune the huge unwanted search space while retaining high expressive power, we can use human support in the search algorithm as early as possible, so that it can steer the search in the right direction.

In the following, we first describe the main challenges in creating such new systems (Section 2), and we then discuss new approaches that we believe are heading in the correct direction (Section 3).

2 Why Rule Discovery is Hard

In this Section, we first introduce formally a class of rules that will be used in the examples, and briefly give intuitions about other relevant classes. We then discuss rule discovery algorithms in general. Finally, we give the five main challenges in rule discovery.

2.1 Denial Constraints

Consider the example in Figure 2 with items from a chain of grocery stores in three states, "AZ", "CA" and "WA". Assume that only "Shoes" from the store in state "AZ" and "EarPhones" from the store in "WA" are labeled as "General"; in any other state both items can only be listed under the type "FootWear" and "Electronics", respectively. The highlighted value for the Type indicates an error in tuple r_3, as the entry "EarPhones" has been labeled "General" instead of "Electronics" in "AZ".

For a relation R, we use a notation for DCs of the form

$\varphi : \forall t_\alpha, t_\beta, t_\gamma, \ldots \in R, \neg(P_1 \wedge \ldots \wedge P_m)$

where P_i is of the form $v_1 \phi v_2$ or $v_1 \phi c$ with $v_1, v_2 \in t_x.A, x \in \{\alpha, \beta, \gamma, \ldots\}, A \in R$, $\phi \in \{=, <, >, \neq, \leqslant, \geqslant\}$, and c is a constant.

	ItemID	Location	Title	Description	Type
r_1	17683	AZ	Levis	Shoes	General
r_2	34987	CA	AllStar	Shoes	FootWear
r_3	14325	AZ	Samsung	EarPhones	General
r_4	82971	WA	Nokia	EarPhones	General
r_5	9286	CA	Toshiba	EarPhones	Electronics

Fig. 2. Items in a chain of grocery stores.

Assume that a mining algorithm comes up with several approximate rules for the relation in Figure 2. We use attribute abbreviations for readability.

$$R_1 : \forall T_\alpha \in R, \neg(T_\alpha.Desc. = \text{“}EarPhones\text{”} \wedge T_\alpha.Type \neq \text{“}Electronics\text{”})$$
$$R_2 : \forall T_\alpha \in R, \neg(T_\alpha.Loc. = \text{“}AZ\text{”} \wedge T_\alpha.Desc. = \text{“}EarPhones\text{”} \wedge T_\alpha.Type \neq \text{“}Electronics\text{”})$$
$$R_3 : \forall T_\alpha \in R, \neg(T_\alpha.Desc. = \text{“}Shoes\text{”} \wedge T_\alpha.Type \neq \text{“}General\text{”})$$
$$R_4 : \forall T_\alpha \in R, \neg(T_\alpha.Loc. = \text{“}AZ\text{”} \wedge T_\alpha.Desc. = \text{“}Shoes\text{”} \wedge T_\alpha.Type \neq \text{“}General\text{”})$$
$$R_5 : \forall T_\alpha, T_\beta \in R, \neg(T_\alpha.Desc. = T_\beta.Desc. \wedge T_\alpha.Type \neq T_\beta.Type)$$
$$R_6 : \forall T_\alpha, T_\beta \in R, \neg(T_\alpha.Loc. = T_\beta.Loc. \wedge T_\alpha.Desc. = T_\beta.Desc \wedge T_\alpha.Type \neq T_\beta.Type)$$

Only some of the approximate rules that have been automatically discovered are correct. Rule R_1 states that all “EarPhones” should be binned into the type “Electronics”, and identifies tuple r_4 as a violation while is not the case. Hence, R_1 is an incorrect rule. Rule R_2 is correct and identifies the error in r_3. However, it only enforces our domain knowledge for “EarPhones” and not for “Shoes”. Since “Shoes” in “AZ” can be misclassified to any other type than “General”, we need an additional rule, such as R_4, to detect all errors. Rule R_3 is correct for items sold in state “AZ”, but would identify correct values as errors in the other states. Rule R_5 represents a functional dependency stating that the Description determines the Type and it is incorrect, since tuples r_4 and r_5 are erroneously identified as errors. Rule R_6 states that Location and Description determine the Type. This rule is correct and more general than the union of R_2 and R_4 as it does not depend on constants. In fact, it can enforce all the domain constraints on future tuples not restricted to just “Shoes” or “EarPhones”.

2.2 Other Rule Types

Denial Constraints can express several common formalisms, such as Functional Dependencies [21,34] and Conditional Functional Dependencies [8,13]. They can express single tuple level rules (R_1–R_4) and table level rules, i.e., rules involving two or more tuples in the premise (R_5,R_6). Also they allow the use of variables and constants, and join across different relations. However, rule types are not limited to what we presented above. There are several other data quality rule types that are common in practice.

Regular-expression based rules specify constraints on textual patterns in a tuple [18] and lead to data transformations such as substring extraction, delimiter identification, and the specification of filters. For instance, a rule may state that only 5-digit numbers in the ItemID attribute of a tuple are allowed.

A very important problem where rules have proven their effectiveness is Entity Resolution [25,33]. The goal here is to perform de-duplication, i.e., to identify pairs and group of records that refer to the same real-world entity. An exemplary rule for the item data in Figure 2 may state that two items should be merged if they have very similar ItemID and the same Location and Title.

Another example of rule types are inference rules, which suggest when two entities respect a particular relationship in a Knowledge Base [16]. These rules help infer and add missing knowledge to the KB. An inference rule can state that two persons sharing a child are married, therefore a new fact can be inferred in the KB. Inference rules are different from association rules [3], which do not derive new facts, but are popular in relational databases to discover interesting relations between variables, e.g., "If a client bought tomatoes and hamburgers, then she is likely to also buy buns".

Finally, lookup rules use an external reliable source of information which can be exploited up to identify errors in the database. The external source can either be a Knowledge Base (KB) [11] or a table of master data [14]. The discovery process tries to map the columns in the relation of interest to the external, reliable source. The resulting rule can then be used to verify if the relation conforms to the external source. For example, in Figure 2, a rule would map the attributes Title and Type in the relation to two columns occurring in a master table, or to a $hasType(Title, Type)$ relationship in a KB. Every erroneous entry for Type, such as "Vegetables" for the Title "Levis", is identified as an error because it violates the mapping stated in the rule. A lookup rule for the example in Figure 2 and a master table M can be expressed in a DC as follows.

$$R_7 : \forall T_\alpha \in R, T_\beta \in M \neg(T_\alpha.Title = T_\beta.ItemTitle \wedge T_\alpha.Type \neq T_\beta.Class)$$

The rule states that if the Title of a tuple in the relation is equal to an ItemTitle value in the master data, then the value in the relation for Type should match the corresponding value specified by Class in the master data, otherwise there must be an error in the relation.

2.3 Discovering Rules

All the different kinds of rules share challenges in their discovery process. We first give an intuition of how these mining algorithms work in general, and we then discuss their challenges in real-world applications. We divide the algorithms into two main categories.

Lattice traversal algorithms. The search space for rule discovery can be seen as a power set lattice of all attribute combinations. Several algorithms try to come up with the right way to traverse such a lattice. Different approaches have been tested, such as level-wise bottom-up traversal strategy and depth-first

random walk. The commonality in these approaches is that they generate new rule candidates sequentially and immediately validate them individually (i.e., with a query over the data). The strategy to deal with the large search space is to prune it incrementally by inferring the validity of candidates that have not been checked yet. This test can be done exploiting the static analysis of the already discovered rules by using the minimality criterion, language axioms (e.g., augmentation and transitivity), and logical implication [17, 19, 21].

Difference- and agree-set algorithms. These algorithms model the search space in an alternative way by using difference and agree-sets. For pair-wise rules, the idea is to search for sets of attributes that agree on the values in the cross product of all tuples. Due to this change in the modelling, they do not try to successively check rules and aggressively prune the lattice search space. On the other hand, they look for attribute sets that agree on certain operators over the tuple values, since those can be in a dependency with other sets of attributes that also agree on some operator. Once obtained the agree-sets, the algorithms try derive the valid rules from them, either level-wise or by transforming them into a search tree that can traversed depth-first [10, 34].

Rule discovery algorithms are then commonly extended in two directions.

First, in the discovery problem a rule is considered correct if there are no violations when applied over the data. However, in real-world scenarios, there are two main reasons to relax this requirement:

Overfitting. Data is dynamic and as more data becomes available, overfitting constraints on current data set can be problematic.

Errors. The common assumption is that errors constitute small percentage of data, thus discovering rules that hold for most of the dataset is a common approach to overcome this issue.

What is usually done is to modify the discovery statement into an *approximate rule discovery problem*. The goal is still to find valid rules, but now a rule is considered of interest if the percentage of violations (i.e., tuples not satisfying the rules) is below a given threshold. For this new problem, the original mining algorithm is revised to take this extension into account.

A second important and common aspect is the extension of the search space to also handle constants. The reason is that a given rule may not hold on the entire dataset, thus conditional rules are useful. Adding a new predicate with a constant is a straightforward operation, but the number of constant predicates is linear w.r.t. the number of constants in the active domain, which is usually very large. The common approach here is to focus on discovering rules for the frequent constants in the dataset [12, 13].

As for the approximate version, the problem is revised to discover rules that involve constants with a frequency in the dataset above a given threshold. A common solution is to follow the "Apriori" approach to discover the frequent constants and then include only these constants in the predicates in the search space [10].

2.4 Challenges

We now use the example in Figure 2 to illustrate the five main issues in rule discovery.

Scalability. Discovering rules is an expensive process. Consider rule discovery systems for Knowledge Bases [16]. A KB consists of <subject, relationship, object> triples such that a relationship holds over the <subject, object> pair. Since KBs do not have a generic schema presented upfront, the discovery of a rule is done by the enumeration of all the instances of <subject, object> pairs conforming to any of the relationships in the KB. In order to efficiently measure the support for the rules while discovering them, the entire KB is loaded into the main memory. Also, literals (constants) are removed from KB to make it fit the memory. Even with such extensive pre-pruning, these systems suffer from memory concerns and are hence constrained to mine rules with at most 3 atoms.

Traditional database rule mining methods suffer from similar limitations [8, 13, 21, 26, 34]. Since any subset of attributes can be part of a rule, there is an exponential number of rules to be tested w.r.t. the number of attributes in the input relation [10, 19]. Also, rules that look over pairs of tuples with operators different from the equality, such as similarity comparisons for entity resolution [25, 30] or not equal operators in DCs, have a quadratic complexity over the number of tuples in the dataset. Main memory algorithms can address this issue, but have obvious constraints on the maximum amount of data that can be handled.

Sampling seems a natural candidate to alleviate the memory concerns when generating rules. However, picking a representative sample that captures the subtleties of all the existing patterns in the data is not feasible. The main reason is the necessity to discover rules involving constants. For example, there is only a single tuple, r_1, in Figure 2 for the fact that in "AZ", "Shoes" can be categorized as a "General" item. However, even if we sample three tuples r_2, r_3 and r_4 from the item list, thus with a larger sample size, we would not capture the specified pattern. The same issue applies with larger and more realistic datasets.

Another dimension in the complexity is the number of predicates that need to be tested. If we enable the use of a library of UDFs, each with its own configuration, such as a threshold for a similarity function in entity resolution [33], the search space becomes even larger and less tractable.

Noise. Noise is omnipresent in real datasets, and with percentages that can reach up to 26% of the data in applications such as data integration [1]. In addition, data can quickly turn stale. For instance, with concept drift, the composition of electronic goods like smartphones changes, thus there is a need of updating the classified category of their components [32]. In order to handle significant percentages of noise, rule discovery algorithms allow *approximate rules* that hold on most of the data. This is done by setting a threshold on the amount of admissible violations for a rule to be still considered valid. Approximate rules

are learnt from the patterns in the data occurring with a percentage of exceptions below the threshold. While this seems to be a valid solution, and it is used in several approaches, there are two important complications.

- Since the amount of errors in data is usually unknown, identifying a suitable threshold to overcome the noise is a trial-and-error process, where several thresholds are tried until an appropriate value is identified. In our running example, we would have to put a 20% threshold to discover R_2 (r_3 is one tuple among five).
- Even after setting a threshold, it is not guaranteed that the rules mined out of the frequent patterns are semantically valid. In fact, large thresholds lead to rules that are incorrect. For example, if we set the noise threshold to 20% in Figure 2, a DC discovery algorithm would mine rule R_5
  $\forall T_\alpha, T_\beta \in R, \neg(T_\alpha.Desc. = T_\beta.Desc. \wedge T_\alpha.Type \neq T_\beta.Type)$
  This is because we treat as error the evidence from tuple r_1 that in "AZ", "Shoes" are allowed to be categorized as "Grocery" items. An appropriate rule would have been
  $\forall T_\alpha, T_\beta \in R, \neg(T_\alpha.Location = T_\beta.Location \wedge T_\alpha.Description = T_\beta.Description \wedge T_\alpha.Type \neq T_\beta.Type)$
  but this is not inferred because the incorrect rule is more general, and for implication the correct rule is not part of the output.

Large Output. Consider again the example in Figure 2. A possible rule would state that ItemID equals to "9286" indicates "EarPhones". If we go beyond comparisons based on equalities, we could incorrectly infer that items with a description of "EarPhones" and ItemID greater than "9286" are classified with type "General". Experiments show that temporal FDs are most effective if the duration constants are discovered at the entity level by defining a rule with different constants for each entity [1]. For instance, Obama travels more often than an average person, and therefore has a smaller duration in the "same time" example discussed above. The same observation motivated Conditional Functional Dependencies, which extend FDs with constants. It is easy to see that there is a plethora of rules pivoting on constants, and the good ones are hidden among the many that do not hold semantically. The traditional way to handle this big search space is to rely on the most popular constants [1, 10]. These constants occur in enough tuples to gather the evidence to derive a rule. This support threshold to mine rules containing only popular constants is a crucial parameter in the input definition to find the sweet spot between acceptable execution times and the discovery of useful rules involving constants.

Enabling constants leads to a large number of rules in the output of the mining systems. To facilitate the users, implication tests for pruning and ranking techniques are popular solutions. However, ranking rules is hard, as useful, correct rules may have very low support, i.e., since they cover only very few tuples for rare events, therefore they may end up at the bottom of the ranking. Other systems resort even to crowdsourcing as a post-processing step to evaluate the rules [11, 32]. However, the results are commonly in the order of thousands of

rules, thus hard to skim, especially when most of the discovered rules are not useful because of the issues raised by the approximation that handles the noise. Ultimately, selecting the correct rules among thousands of results is a daunting task that requires experts both in the rule language at hand (to understand the precise semantics of the rule: "what does it mean?") and in the data domain (to validate the semantic correctness of the rule: "is it always true?").

Hard Configuration. As it should be clear from the discussion on the role of noise and constants, rule discovery algorithms require several non-obvious parameters to be set. In general, there is no way to know in advance what is the amount of noise that should be tolerated in the discovery of the rules. Also, high noise threshold can hide important rules, so there is no one unique value that suddenly leads to the discovery of all the semantically correct rules. The same challenge applies for the threshold for the constant values and several other parameters that are language specific. For example, in algorithms for the discovery of TFDs the granularity of the time buckets is required (minutes, hours, or days?) [1], or for inference rules mining, the maximum number of hops to be traversed in the KB must be set [16].

Since the search space of possible rules is exponential in the number of attributes, some systems even require an initial suggestion of the rule structure from the user to begin with. For instance, in [33], the user is asked to provide the DNF specification of the rule grammar for Entity Resolution that specifies the attributes that need to be considered to classify a pair of tuples as referring to the same entity or not. In addition to that, in case the rules rely on functions with accompanying thresholds, it is a difficult task for the user to specify those values (how similar should two IDs be to be considered a match?).

Need for Heterogeneous Rule Types. Several types of rules are needed for any application, but there is no a single system that discovers all kinds of rules [2]. There are primarily two different types of rules - syntactic and semantic. The error that we see in tuple r_3 of Figure 2 can be fixed by a semantic rule, such as a DC that states that "EarPhones" can be tagged as a "General" item in all states but "WA" (R_2). But a regular expression that restrains the text patterns in the table in Figure 2 would capture if an entry for Location is expressed as "Washington", instead of "WA". Likewise, if another syntactic rule states that ItemID can only be a 5-digit number, tuple r_5 can be treated as a violation of that rule because of the 4-digit entry for $ItemID$. Specific tools such as `Trifacta` [18] and `OpenRefine` [2] discover and enforce syntactic rules as regular expressions on the textual patterns of the attribute values. The same discussion applies for lookup rules. There can be data errors that are not captured syntactically nor by a DC, but require to verify the data with some reference information, such as in R_7 (Section 2.2), but these rules usually require different discovery algorithms (e.g., [11, 14]).

It is clear that in general more than one rule type needs to be defined for a given application. But this implies that multiple tools need to be configured and multiple outputs must be manually verified by the users.

3 Opportunities and Directions

To overcome the challenges in Section 2, we envision a rule discovery system that puts the users at the center of the mining process. The main idea is that the human-in-the-loop proactively participates in the discovery by interacting with the mining algorithms, instead of limiting the interaction to the specification of parameters and the post-pruning of rules emitted by the black box. Following are the main directions of research that we recognize in recent work for the new generation of rule discovery systems.

Continuous Involvement. Given the challenges discussed in the previous section, we argue that there is a clear opportunity of having the human expert involved along with the system in the rule generation process. This means that the users do not need to set up the parameters upfront, and do not need to evaluate long lists of rules at the output. However, the first step to move toward this vision is to achieve interactive response times in the mining steps. There have been several efforts to reduce the overall mining time by exploiting distributed algorithms [7,19]. These solutions exploit parallelization techniques to distribute the most expensive operations, such as joins, by using native primitives under the Map-Reduce paradigm. However, we argue that another direction should be explored to enable a novel, more effective approach to the rule discovery problem.

The solution we envision for this problem is to drop the one-shot algorithms that discover all the possible rules in a dataset. Instead, we should interleave the pivotal steps of the rule mining algorithm with user interactions. Of course, understanding when and how to ask for user feedback is a crucial requirement. Recent works have started to look at this problem in the context of user updates [17,18]. The systems discover the possible rules underlying a given update and validate the most promising tentative rules with the users. This early feedback is useful to prune large portions of the search space and guide the algorithms towards the correct rules. Besides that, pivoting on the user for feedback can also address the issue of *noise*, even when the examples underlying meaningful rules have very small support in the table, such as tuple r_1 in Figure 2. In this example, a single tuple is below the noise threshold and can hence be mistaken for noise. However such an example can be championed by the data expert is (s)he thinks that the corresponding rule can contribute to high coverage and recall.

Let us clarify this idea of contribution in the context of data cleaning. For data cleaning rules, the challenge lies in identifying the rule that maximizes the number of covered dirty tuples in the database, while minimizing the number of questions asked to the users. Given a search space of the possible rules, the algorithms try to quickly identify rules to be presented to the user for validation that are both as general as possible (to maximize impact) and most likely correct

(to quickly identify valid rules). This is in opposition to the enumeration of the entire space of traditional algorithms. Results show that with as little as two questions for a user update, general rules can be discovered [17]. This is done from a user update, a simple action that does not require setting input parameters. By validating a small number of "promising" rules with the user (i.e., rules that find the good compromise between coverage and likelihood of being correct), the system is not exposing the long list of rules to evaluate at the end of a time consuming mining. Thus the algorithmic effort in presenting the right set of questions to the human-in-the-loop coupled with her feedback tackle the issue of the *large output*.

For example, consider again Figure 2. If the user updates the incorrect value "General" to "Electronics", this would create a search space with Type as the right hand side of the rule, as this is where the user made the update. Now the search space is still exponential over the number of attributes, but with the user validating or refusing a possible rule, the algorithm quickly converges to the search space area containing the correct rule. Suppose the first rule exposed to the user is

$$\forall T_\alpha \in R, \neg(T_\alpha.Desc. = \text{“}EarPhones\text{”} \wedge T_\alpha.Type \neq \text{“}Electronics\text{”})$$

and the user does not validate it. This is a clear sign that a more specific rule is needed, if we want to use Description. So another question can be asked for rule

$$\forall T_\alpha \in R, \neg(T_\alpha.Loc. = \text{“}AZ\text{”} \wedge T_\alpha.Desc. = \text{“}EarPhones\text{”} \wedge T_\alpha.Type \neq \text{“}Electronics\text{”})$$

which this time is validated. A rule involving constants is obtained for a dirty dataset without setting any input parameter.

However, two important limitations hinder the impact of such solutions. First, the current languages support simple 1-tuple rules, thus not exploiting more powerful rule languages, such as Denial Constraints over multiple tuples. Second, these systems are designed to handle one update at a time, or a sequence of updates with the same semantics. However, given a batch of user updates over the data, such as historical data for data cleaning, it is rarely the case that all updates have been made with a single rule in mind. On the contrary, it is likely that each, or subsets, of the updates have a different underlying rule guiding the users towards the update. This is a challenging problem for which new algorithms are needed.

	ItemID	Location	Title	Description	Type
r_1	-	AZ	Levis.D	Shoes	General
r'_1	-	Arizona	L.Denim	Shoes	FootWear
r_2	34987	California	AllStar	Shoes	FootWear
r'_2	-	CA	Converse.AS	Shoes	FootWear

Fig. 3. De-duplicating items in a grocery store.

User Defined Functions. As discussed above, more expressive rules are needed in real world datasets for complex problems, such as de-duplication (a.k.a., entity resolution) [6,23,30]. For instance, if we treat r_1 and r'_1 as a tuple pair in Figure 3, we may want to be able to assess whether the records in the pair are duplicates of the same item.

An approach to the de-duplication problem is to feed ML algorithms with positive and negative examples, and build a model for a classifier. Interestingly, a set of rules can outperform an ML based approach for this task [30, 33]. In particular, provided a grammar to shape the form of the rules and a library of similarity functions (available also to the ML classifier), the appropriate functions and their thresholds can be automatically discovered given sets of positive and negative pairs. In our example in Figure 3, a DNF grammar can be

$$sim(ItemID) \vee (sim(Title) \wedge sim(Type)) \vee (sim(Title) \wedge sim(Loc.) \wedge sim(Desc.))$$

The rules satisfying this grammar state that a tuple pair can be labeled identical if the tuples are similar w.r.t. ItemID alone (expressed by *sim* function), or over Title and Type together, or upon Title, Location, and Description. However, the ItemID values are often missing from the table in Figure 3, which makes other DNF clauses in the grammar more useful in this example. Given a library of similarity functions $F_1, \ldots, F_n$ and a set of threshold values $T_1, \ldots, T_m$, the computation of similarity function $sim(attr)$ in the DNF grammar is done by checking every $F_i(r_1[attr], r'_1[attr]) > T_j$, with $i \in 1, \ldots, n$ and $j \in 1, \ldots, m$. The test checks if the outcome of applying the similarity function F_i upon the attribute *attr* exceeds one or more threshold values. The system then picks the appropriate F_i and T_j for all attributes participating in the DNF grammar by pruning redundant threshold values and similarity functions with greedy and hill climbing algorithms.

This is a case of the opportunity of using a library of UDFs to discover more expressive rules, without exposing the internals of the mining to the domain experts. The system is thus not *hard to configure* as all the human helper needs to provide is an easy-to-define input in the form of labeled training data and has to examine the results matches and mismatches, which can be used to refine the rules, rather than the output rules themselves. As in the case of the updates, users only have to deal with examples over the data, thus there is no required expertise in logic nor in procedural code. However, in cases where the training sets are too small or not representative, the above approach would fail. It is easy to see *active learning* as a tool to help classify ambiguous test data by using human support. Bootstrapping hard-to-classify test points into the training data will strengthen the rule mining algorithms. Tuple r_1 in Figure 2 is a hard-to-classify example as it is mistaken for an error by most rule discovery systems. A human may know that stores in "AZ" should classify "Shoes" as "General" items, thus she would label this tuple as a candidate into the training data. However, integrating active learning to the discovery process is not easy because new mining algorithms should be designed for identifying what are the most beneficial examples for the internal model. While this has shown potential in

ad-hoc solutions [29], it is not obvious how to make it more general for the discovery of arbitrary rules beyond entity resolution.

Tool Ensembles. Given the necessity of using a plethora of *heterogeneous rules* to fix multiple types of errors, an ensemble of tools performs better than a single tool. Recent results have shown that combining multiple kinds of rules is mandatory to obtain high recall in the task at hand, and best results are obtained when combining rules with statistical methods [2, 27]. For example, a recent ensemble for error detection in [2] consists of tools for outlier detection together with both syntactic and semantic rules. The authors here focus on combining the cleaning suggestions from these tools over the data, as they rely on manually tuned tools and *hand written rules*.

However, we know that discovering correct rules is a challenging problem, and (manually) doing it over dirty data for multiple tools is indeed an expensive operation. To tackle this problem, we believe that the idea of assembling different rules should be lifted to the idea of combining multiple discovery algorithms. Instead of having the user manually checking rules for each tool supporting a single language (say, syntactic or semantic), the ensemble over the data enables a unified approach to the heterogeneity problem. Given multiple algorithms to discover rules on a given dataset, we can filter the rules emitted by the ensemble to apply only those that have a mutual consensus about identifying a tuple or an attribute for the task at hand. This naive approach is similar to majority voting – if a large fraction of algorithms agree, then they are trustable –but more sophisticated techniques can be developed. For example, the majority voting can be parameterized by using a *min-K* approach, where K indicates the minimum number of tools that produce rules that agree over the data. Another approach resorts to ordering the diverse rules from the ensemble by their estimated precision, for example computed upon a sampled dataset for which the ground truth is available. This enables a data expert to validate the outcome emitted by each tool in the ranked order, while implicitly giving feedback on all the rule discovery algorithms. In fact, we can label the rules as meaningful or not depending on the validation of the rule outcome by the expert. This ensures that a manual validation step can greatly help the rule selection.

4 Conclusion

Discovering rules that are semantically meaningful is important in many applications, but it is a challenging task. In this paper, we propose to open the black box of rule discovery systems to the end user by emphasizing the need to employ early human feedback into the rule mining process. There are techniques that also aim at opening the blackbox of Machine Learning for Information Extraction (IE) [9]. However, ML approaches are mostly non-interpretable.

Our vision goes beyond the state-of-the-art rule mining frameworks that use the human help only to set parameters and to select valid rules from the discovered ones. We argue that such design decisions fail to effectively help the

users in producing meaningful rules. This goal can be better achieved by enabling human suggestions during the algorithmic phase of rule discovery.

In this context, we discussed how recent trends in rule discovery systems show great potential for new research that finally put the user at the center of the mining process. We advocate for new solutions with the goal of graciously involving the human in the mining, with very limited input to bootstrap and immediate interaction to guide the mining towards the right direction. The three main directions that we advocate are (i) a direct involvement of the user in the traversal of the search space, (ii) the support for libraries of user defined functions to discover more expressive rules, and (iii) an ensemble of rule discovery algorithms to handle the diversity of languages available and steer effectively the human interaction.

References

1. Z. Abedjan, C. G. Akcora, M. Ouzzani, P. Papotti, and M. Stonebraker. Temporal rules discovery for web data cleaning. *Proc. VLDB Endow.*, 9(4):336–347, Dec. 2015.
2. Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *Proc. VLDB Endow.*, 9(12):993–1004, Aug. 2016.
3. R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, June 1993.
4. T. P. Bhatla, V. Prabhu, and A. Dua. Understanding credit card frauds. In *Cards Business Review 1.6*, 2003.
5. R. Brause, T. Langsdorf, and M. Hepp. Neural data mining for credit card fraud detection. In *ICTAI*, 1999.
6. B. Chardin, E. Coquery, M. Pailloux, and J.-M. Petit. RQL: A Query Language for Rule Discovery in Databases. *Theoretical Computer Science*, Nov. 2016.
7. Y. Chen, S. Goldberg, D. Z. Wang, and S. S. Johri. Ontological pathfinding: Mining first-order knowledge from large knowledge bases. In *SIGMOD*, pages 835–846. ACM, 2016.
8. F. Chiang and R. J. Miller. Discovering data quality rules. *PVLDB*, 1(1):1166–1177, 2008.
9. L. Chiticariu, Y. Li, and F. Reiss. Transparent machine learning for information extraction. In *EMNLP (tutorial)*, 2015.
10. X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *Proc. VLDB Endow.*, 6(13):1498–1509, Aug. 2013.
11. X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. KATARA: a data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD*, 2015.
12. C. T. Dieng, T.-Y. Jen, D. Laurent, and N. Spyratos. Mining frequent conjunctive queries using functional and inclusion dependencies. *The VLDB Journal*, 22(2):125–150, 2013.
13. W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering conditional functional dependencies. *IEEE TKDE*, 23(5):683–698, 2011.
14. W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *The VLDB Journal*, 21(2):213–238, 2012.

15. T. Furche, G. Gottlob, L. Libkin, G. Orsi, and N. W. Paton. Data wrangling for big data: Challenges and opportunities. In *EDBT*, pages 473–478, 2016.
16. L. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek. Fast rule mining in ontological knowledge bases with AMIE+. *The VLDB Journal*, 24(6):707–730, Dec. 2015.
17. J. He, E. Veltri, D. Santoro, G. Li, G. Mecca, P. Papotti, and N. Tang. Interactive and deterministic data cleaning. In *SIGMOD*, 2016.
18. J. Heer, J. Hellerstein, and S. Kandel. Predictive interaction for data transformation. In *CIDR*, 2015.
19. A. Heise, J.-A. Quiané-Ruiz, Z. Abedjan, A. Jentzsch, and F. Naumann. Scalable discovery of unique column combinations. *Proc. VLDB Endow.*, 7(4):301–312, Dec. 2013.
20. B. Hu, T. Patkos, A. Chibani, and Y. Amirat. Rule-based context assessment in smart cities. In *Web Reasoning and Rule Systems: RR*, pages 221–224, 2012.
21. Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.
22. K. Julisch and M. Dacier. Mining intrusion detection alarms for actionable knowledge. In *KDD*, pages 366–375, 2002.
23. Z. Khayyat, I. F. Ilyas, A. Jindal, S. Madden, M. Ouzzani, J.-A. Quiane-Ruiz, P. Papotti, N. Tang, and S. Yin. BigDansing: A system for big data cleansing. In *SIGMOD*, 2015.
24. T. Milo, S. Novgorodov, and W.-C. Tan. RUDOLF: Interactive rule refinement system for fraud detection. *Proc. VLDB Endow.*, 9(13):1465–1468, Sept. 2016.
25. F. Naumann and M. Herschel. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.
26. T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J. Rudolph, M. Schönberg, J. Zwiener, and F. Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *PVLDB*, 8(10):1082–1093, 2015.
27. N. Prokoshyna, J. Szlichta, F. Chiang, R. J. Miller, and D. Srivastava. Combining quantitative and logical data cleaning. *Proc. VLDB Endow.*, 9(4):300–311, Dec. 2015.
28. M. Roesch. SNORT - Lightweight intrusion detection for networks. In *LISA*, pages 229–238, 1999.
29. S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *SIGKDD*, pages 269–278, 2002.
30. R. Singh, V. Meduri, A. K. Elmagarmid, S. Madden, P. Papotti, J. Quiané-Ruiz, A. Solar-Lezama, and N. Tang. Generating concise entity matching rules. In *SIGMOD*, pages 1635–1638, 2017.
31. S. Song, L. Chen, and H. Cheng. Efficient determination of distance thresholds for differential dependencies. *IEEE Trans. Knowl. Data Eng.*, 26(9):2179–2192, 2014.
32. 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.
33. J. Wang, G. Li, J. X. Yu, and J. Feng. Entity matching: How similar is similar. *Proc. VLDB Endow.*, 4(10):622–633, July 2011.
34. C. M. Wyss, C. Giannella, and E. L. Robertson. FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances. In *DaWaK*, pages 101–110, 2001.