Affordable Discovery of Positive and Negative Rules in Knowledge-Bases

us

ABSTRACT

We present KRD, a system for the discovery of declarative rules over knowledge-bases (KBs). KRD does not limit its search space to rules that rely on "positive" relationships between entities, such as "if two persons have the same parent, they are siblings", as in traditional mining of constraints for KBs. On the contrary, it extends the search space to discover also negative rules, i.e., patters that lead to contradictions in the data, such as "if two person are married, one cannot be the child of the other". While the former class is fundamental to infer new relationships in the KB, the latter class is crucial for error detection in data cleaning, or for the creation of negative examples when bootstrapping learning algorithms.

The main technical challenges addressed in this paper consist in enlarging the expressive power of the considered rules to include comparison among constants, including disequalities, and in designing a disk-based discovery algorithm, effectively dropping the assumption that the KB has to fit in memory to have acceptable performance. To guarantee that the entire search space is explored, we formalize the mining problem as an incremental graph exploration. Our novel search strategy is coupled with a number of optimization techniques to further prune the search space and efficiently maintain the graph. Finally, in contrast with traditional ranking of rules based on a measure of support, we propose a new approach inspired by set cover to identify the subset of useful rules to be exposed to the user. We have conducted extensive experiments using both real-world and synthetic datasets to show that KRD outperform previous proposals in terms of efficiency and that it discovers more effective rules for the application at hand.

1. INTRODUCTION

2. PRELIMINARIES AND DEFINITIONS

Talk about KBs.

2.1 Language

Horn Rule, with the restriction of having each variables appearing twice. Extension of predicates with inequalities.

A Horn Rule r has the form $A_1 \wedge A_2 \wedge \cdots A_n \Rightarrow r(a, b)$, where $A_1 \wedge A_2 \wedge \cdots \wedge A_n = r_{body}$ is the *body* of the rule.

2.2 Coverage

Given a pair of entities (x, y) from the KB and a Horn Rule r, we say that r_{body} *covers* (x, y) if $(x, y) \models r_{body}$. In other words, given a Horn Rule $r = r_{body} \Rightarrow r(a, b)$, r_{body} covers a pair of entities (x, y) iff r_{body} can be instantiated over the KB by substituting a with x and b with y. Given a set of pair of entities $E = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$ and a rule r, we denote by $C_r(E)$ the *coverage* of r_{body} over E as the set of elements in E covered by r, $C_r(E) = \{(x, y) \in E | (x, y) \models r_{body}\}$.

Given the body r_{body} of a Horn Rule r, we denote by r^*_{body} the *unbounded body* of r. The unbounded body of a rule is the set of all atoms in r_{body} that contains either variable a or variable b, and where the other variable of the atom is substituted with another unique variable. As an example, given $r_{body} = rel_1(a, v_0) \wedge rel_2(b, v_0)$, $r^*_{body} = rel_1(a, v_1) \wedge rel_2(b, v_2)$. Paolo: I suggest to have $rel_3(a, b)$ to avoid the confusion raised by cartesian product Given a set of pair of entities $E = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$ and a rule r, we denote by $U_r(E)$ the *unbounded coverage* of r^*_{body} over E as the set of elements in E covered by r^*_{body}, $U_r(E) = \{(x, y) \in E | (x, y) \models r^*_{body}\}$.

Example 1. *Given the rule* $r = \texttt{hasChild}(a, v_0) \wedge \texttt{hasChild}(b, v_0)$ *and a KB K, we denote by E the set of all possible pairs of entities in K. The coverage of r over E ($C_r(E)$) is the set of all pairs of entities (x, y) where both x and y are in relation* `hasChild` *with the same entity v_0, while the unbounded coverage of r over E ($U_r(E)$) is the set of all pairs of entities (x, y) where x is in relation* `hasChild` *with an entity v_1 and y is in relation* `hasChild` *with an entity v_2, and not necessarily $v_1 = v_2$.*

2.3 Scoring Function

Introduce red-blue set cover as a possible solution, and say why it is not appropriate.

Given a KB K, a set of pair of entities G from K (generation set), a set of pair of entities V from K (validation set) where $G \cap V = \varnothing$, and a set of Horn Rules $R = \{r_1, r_2, \cdots, r_n\}$, we define the *cumulative score* $s(R, G, V)$

as the function:

$$\alpha\cdot\left(1-\frac{|\bigcup_{r_i\in R} C_{r_i}(G)|}{|G|}\right)+\beta\cdot\sum_{r_i\in R}\frac{|C_{r_i}(V)|}{|U_{r_i}(V)|}+\gamma\cdot\left(1-\frac{|\bigcup_{r_i\in R} U_{r_i}(V)|}{|V|}\right)$$

Paolo: if we want to minimize it, it should be defined as a cost function

2.4 Problem Definition

Given a KB K, a generation set G and a validation set V, where both G and V are sets of pair of entities from K, an optimal set of rules R^* is the set of rules that minimizes the cumulative score:

$$R^* = \min_R\{s(R, G, V)\}$$

- **Optimal Solution.** Generate universe of all possible rules U and solve an integer linear programming problem. We have a integer variable x_r for each rule in U, 1 denotes the rule is in the optimal output, 0 the rule is not. The objective function of the problem is the minimization function defined above;
- **Greedy Algorithm.** Best if we can prove that the optimal problem is NP hard, but even if we cannot we can always argue that computing the optimal solution is expensive since we have to generate all possible rules. In the greedy approach, at each iteration, we choose the rule that gives the smallest contribution to the cumulative score (since the object is to minimize). Greedy algorithm will not produce the optimal solution in some circumstances, but it does not require the generation of all the rules.

2.5 Alternative Problem Definition

Given a KB K with G and V, and a set of rules R with the union of their bodies B, a solution for the *exact discovery problem* is a subset R' of R s.t.

$$R_{opt} = \underset{|R'|}{\text{argmin}}(R'|(B'(K) = G) \wedge (B'(K) \cap V = \varnothing))$$

Discussion: this solution may not exist. This may lead to one query for each triple in G. We need more flexibility to account for errors in G and V, this leads to better rules. We therefore introduce a cost function c as defined above. Paolo: here goes the description of the three intuition behind the three components, why they are needed, etc. Show with example introduce in intro

We can now state the approximate version of the problem.

Given a KB K with G and V, a set of rules R with the union of their bodies B, and an c cost function for K, a solution for the *approximate discovery problem* is a subset R' of R s.t.

$$R_{opt} = \underset{c(R')}{\text{argmin}}(R'|(B'(K) \supseteq G))$$

We can map this problem to the well-known weighted set cover problem, which is proven to be an NP-Complete problem [**?**], where the universe is G and the sets are all the possible queries over K.

Since the set of all possible queries for K is too big to enumerate, we solve the online variant of the above problem. Paolo: if offline problem is NP, online is at least NP; to be verified

3. RULES DISCOVERY

Talk about translation from Horn Rules to paths on the graph.

3.1 Literals and Constants

Generation of artificial edges to include inequalities. Substitions of variables with constants if same value appears for each example.

3.2 Input Examples Generation

Define how we compute generation and validation set: how we generate positive and negative examples.

4. A* GREEDY ALGORITHM

Justify the use of a greedy algorithm: generating all the rules is very expensive. Better if we can prove that the minimization function algorithm is NP hard.

4.1 Optimality

Define property on why the A* algorithm produces the greedy solution. Maybe study when the greedy solution become optimal? (If all rules identify disjoint set of input example, then greedy solution is optimal)

5. EXPERIMENTS

5.1 Negative Rules Evaluation

Evaluation of negative rules.

5.2 Comparison Evaluation

Comparison against AMIE and evaluation of positive rules.

5.3 Machine Learning Application

DeepDive.

6. RELATED WORK

7. CONCLUSION