The Journal of Open Source Software

YAD: A Learning-based Inductive Logic Programming Tool

Ali Kamandi[1] **and Hamed Karimi**[1]

1 School of Engineering Science, College of Engineering, University of Tehran, Tehran, Iran

DOI: 10.21105/joss.00892

Software

- Review
- Repository
- Archive

Submitted: 09 July 2018
Published: 18 October 2018

Summary

Inductive logic learning is considered as one of the most prominent approaches for multi-dimensional and multi-tabular learning by relying on first-order logic to describe train data in the form of positive and negative examples which can find a set of existing rules and relations and can propose them in first-order logic format (Muggleton, 1995), (Lavrač & Džeroski, 1994).

Generally, the inductive learning problem is defined as follows:

- **Input:**
  - Background knowledge, B, a set of Horn clauses.
  - Positive examples E^+, a set of Horn clauses (typically ground literals).
  - Negative examples E^-, a set of Horn clauses (typically ground literals).
- **Output:**
  - A hypothesis, H, a set of Horn clauses.
- **Problem Conditions:**
  - All the positive examples should be producible from combination of background knowledge and H, which it indicates the completeness criterion:

$$\forall e \in E^+ : B \wedge H \rightarrow e \quad \text{(Completeness)}$$

  - None of the negative examples should be producible, which it indicates the consistency criterion and also obviously keeps the Precision measure high:

$$\forall e \in E^- : B \wedge H \nrightarrow e \quad \text{(Consistency)}$$

It is remarkable to say that the symbols E^+, E^-, B, and H are proposed in first-order logic format.

Due to the expressiveness and high flexibility of first-order logic, the inductive learning approach has high capability in discovering the relations. Also, since the set of logic's rules is not restricted to a specific table or structure, it can be used in widespread areas and problems like finding relationships in social networks and discovering communication patterns in protein structures (bioinformatics problems).

Despite of all the advantages that learning based on inductive logic has; in practice, it becomes a problem to search and find the optimal solution which with respect to the vastness of search space, it has no polynomial and deterministic solution. So, in order to solve it, we need to the heuristic and randomized algorithms.

In recent years, many researches were performed in this area and numerous algorithms and tools were proposed such as FOIL (Quinlan & Cameron-Jones, 1993), (Quinlan, 1990), Golem, ProGolem (Muggleton, Santos, & Tamaddoni-Nezhad, 2010), Aleph (Srinivasan, 2007), and CIGOL. The algorithms provided so far for this problem, can be classified into two categories: top-down and bottom-up algorithms.

In top-down algorithms, some relations are guessed (they are constructed) based on heuristic criteria and then, their validity is tested on train data. While learning algorithm conditions are satisfied, that relation is accepted. In bottom-up algorithms, the work is started from positive examples and the algorithm endeavors to find a more general form of a fact. In the other words, the desired relation is generalized and then, a more generic form of it is created. This work is continued until the generated rule covers none of the negative examples.

The tool YAD, is an implementation of a new bottom-up inductive logic learning algorithm. Its main purpose is to reduce learning time. This is done in such a way that in every step, in order to choose some rules that can create more general rules with their own combinations, instead of quite random choice, choosing is performed from a set of relations (rules) that have a higher chance to generate the relevant rules.

Generally, this algorithm (Algorithm 1) has been presented as follows:

Algorithm 1. The proposed induction algorithm used in YAD ILP tool

1: **Input:** *Positive examples* E^+*, negative examples* E^-*, background knowledge* B*.*
2: **Output:** *Set of produced predicates (or rules)* H*.*
3: **Procedure** Induction Algorithm:
4: $H \longleftarrow \emptyset$;
5: Construct a hash map h, of all constants and containing clauses in B;
6: **foreach** clause $c \in E^+$ **do**
7: **if** c is not covered yet **then**
8: Make a predicate P as c:- ;
9: **foreach** predicate $P' \in Induce^*(P, B, h)$ **do**
10: $H \longleftarrow H \cup P'$;
11: **end foreach**
12: **end if**
13: **end foreach**
14: Remove duplicate predicates from H;
15: Remove weak predicates from H;
16: Remove predicates which produce negative examples from H;

** Constructing a generalized logical rule using inverse resolution on a predicate P with background knowledge B and a hash map h.*

This tool has been implemented in C# and a screenshot of its GUI is depicted in Figure 1. The input arguments of YAD are:

- The percentage of Train Examples (the remaining percentage will be Test Examples percentage for testing the generated generic logical rules on the data-set).
- The number of steps (typically, it is set to the default value 4) which is used to generate a general predicate in every step of induce function (inverse resolution

operations) in order to produce generic logical rules. So, for example, with four steps, the tool can produce general logical rules with at most four predicates in their right-hand side.

- Try Count (typically, it is set to the default value 5) is the number of endeavors that the tool performs to create and generate a general logical rule.
- Negative Threshold (typically, it is set to a low amount e.g. the default value 5) is the percentage of the whole negative examples that every produced generic logical rule can produce negative examples equal or lower than the selected percentage, otherwise, the produced logical rule is useless and it is not considered as an output generic logical rule.

Also, the use cases and functions of this tool are as follows (respectively in use):

1. **Open:** Opening a file (typically, a text file) consists of Background Knowledge lines (the lines should start with 'B' with a white-space after), Positive Examples lines (the lines should start with '+' with a white-space after), and Negative Examples lines (the lines should start with '-' with a white-space after) in first-order logic format as logical rules.
2. **Induce:** Producing general logical rules by inducing (using inverse resolution) on background knowledge.
3. **Prune:** Computing the measures such as Precision, Recall, Accuracy, and F-Measure for evaluation and comparison.
4. **Result Filtering:** Colorizing and filtering the results, i.e., Train and Test Examples in order to determine their coverage.

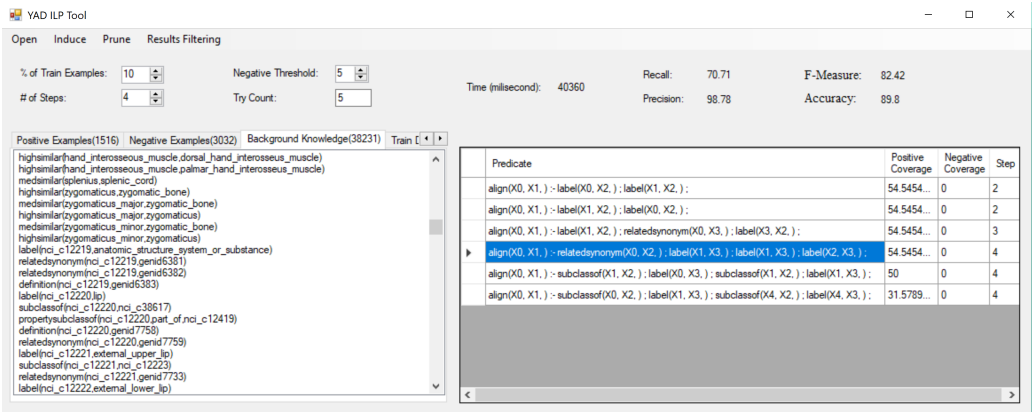

Figure 1. A screenshot of YAD

Software Architecture

The architecture of this ILP tool is shown in Figure 2. As it is obvious in this figure, the system consists of the following components:

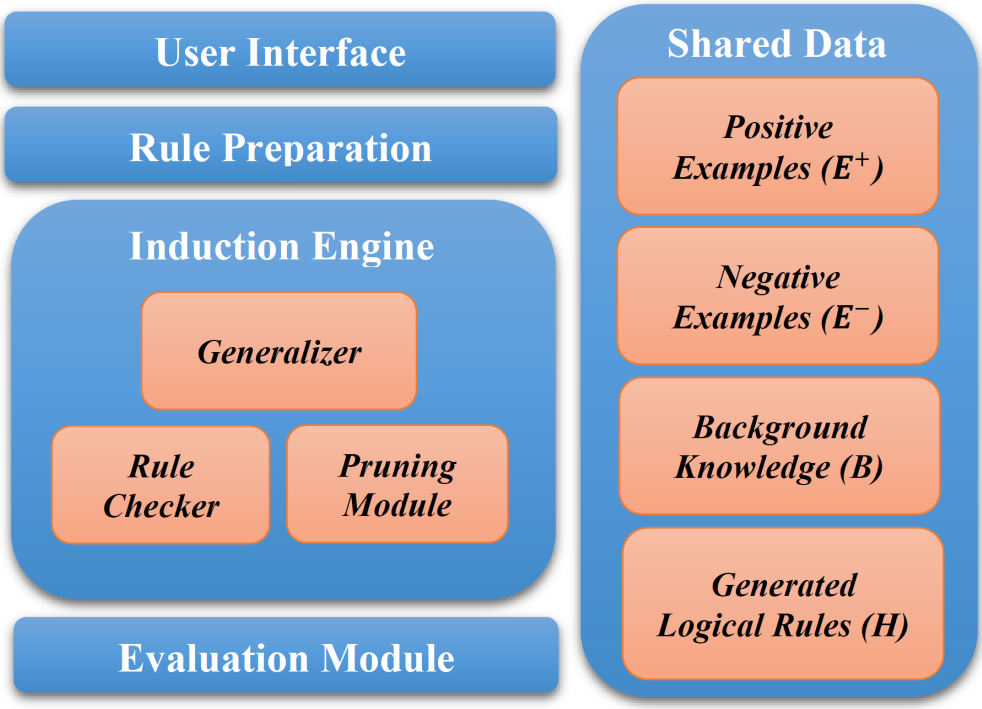

Figure 2. YAD ILP tool architecture

User Interface: It provides some necessary facilities and UI in order to present the input data and configure the model parameters for user. These user-configurable parameters were mentioned before in previous sections.

Rule Preparation: This section has some tasks to read the input file, parse the information, and produce background knowledge and positive and negative examples set in appropriate form of the first-order logic format. All of the examples (positive and negative examples) are located in the format of a set of objects. Also, the division of the input data into the train and test sets is accomplished by this module.

Induction Engine: This section forms the main part of the tool, which consists of three subsections:

- ***Generalizer:*** This subsection has a task to produce more general logical rules from positive examples. In the other words, this module receives a positive example and tries to generalize it as much as possible.
- ***Rule Checker:*** The logical rules that are generated by generalizer, are controlled by this module to determine whether they satisfy the problem conditions (covering the positive examples and not covering the negative examples) or not. So, if the conditions of the desired logical rule are not covered, then it will not be accepted.
- ***Pruning Module:*** This module has a task to prevent from the over-fitting of the model, otherwise, we probably have many logical rules that each of them satisfies only one or two positive examples, i.e. they do not have an adequate generality. Pruning operates in such a way that not only we can achieve the best results in terms of precision, but also the number of logical rules will be minimized.

Evaluation Module: It has a task to execute the logical rules produced by induction module on the test data and evaluates the results. For this reason, the various measurements such as Precision, Recall, Accuracy, and F-Measure are computed and reported.

Shared Data: The sections we have mentioned before, have access to a set of data which is the input and output data of ILP algorithm such as background knowledge set, positive and negative examples, and the set of produced logical rules by the algorithm.

Applications

This tool can be used in solving many various problems which need to discover some relationships between different phenomena. We previously have used this ILP tool in a prominent area of new web science called Semantic Web for Ontology Alignment (Ontology Mapping) problem using inductive logic programming (Karimi & Kamandi, 2018). We used anatomical data-sets to evaluate our new proposed approach for ontology alignment as a key problem in semantic web and its infrastructure using ILP by YAD. This ILP tool could generate some efficient generic logical rules to find some valid alignments between anatomical concepts and therefore, we could averagely achieve a high F-Measure with respect to acceptable percentage of Precision and Recall, among similar matching tools in 2017.

Also, the ILP tool can be used in many different problems such as bioinformatics and protein bonds analysis as well as informatics chemistry to discover chemical bonds. In addition, it can be used in social networks analysis and to discover the communications between individuals, rules, and concepts. Generally, the application of this tool is in networks analysis (networks of individuals, molecules, proteins, and etc.).

References

Karimi, H., & Kamandi, A. (2018). Ontology alignment using inductive logic programming. In *2018 4th international conference on web research (icwr)* (pp. 118–127). IEEE. doi:10.1109/ICWR.2018.8387247

Lavrač, N., & Džeroski, S. (1994). *Inductive logic programming : techniques and applications* (p. 293). E. Horwood.

Muggleton, S. (1995). Inverse entailment and progol. *New Generation Computing*, *13*(3-4), 245–286. doi:10.1007/BF03037227

Muggleton, S., Santos, J., & Tamaddoni-Nezhad, A. (2010). ProGolem: A System Based on Relative Minimal Generalisation. In (pp. 131–148). doi:10.1007/978-3-642-13840-9_13

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, *5*(3), 239–266. doi:10.1007/BF00117105

Quinlan, J. R., & Cameron-Jones, R. M. (1993). FOIL: A midterm report. In (pp. 1–20). Springer, Berlin, Heidelberg. doi:10.1007/3-540-56602-3_124

Srinivasan, A. (2007). *The Aleph Manual*.