Learning from Ordinal Data with ILP in Description Logic

Nunung Nurul Qomariyah and Dimitar Kazakov
Computer Science, University of York, UK

Presented on the 27th International Conference on Inductive Logic Programming
Orléans, France, 4-6 September 2017
THE OUTLINE

1. Introduction
2. Problem representation
3. Proposed Algorithm
4. Evaluation
5. Conclusions and Further Work
INTRODUCTION

ILP algorithms using DL representation

– Have the potential to be applied to large volumes of linked open data

– To benefit from the tools available for such data
  • e.g. IDEs such as Protégé, DB system such as triplestore and Ontology reasoners
Previous work on ILP in DL includes:

- They are mostly aiming to learn about a concept descriptions
The application area: **Preference Learning (PL)**

- PL [6] aims to induce predictive user preference models from empirical data.
INTRODUCTION

The application area: **Recommender System**

– We have previously published *workshop paper* at the **ACM RecSys** 2017, Como, Italy
The objective of this work: to learn about transitive anti-reflexive relations:

- **Transitive:**
  - We use the examples provided by the user, along with their transitive closure, in their correct order (e.g. “car A is better than car B”) as positive examples.
  - Example of transitive closure:
    - User provides: “car A is better than car B”
    - User also provides: “car B is better than car C”
    - We add a closure: “car A is better than car C” as a positive example.

- **Anti-reflexive:**
  - We use the same examples in reverse order as negative examples. (e.g. “car B is better than car A”)
Hypothesis language

In Aleph

:- modeh(1,betterthan(+car,+car)).
:- modeb(1,hasbodytype(+car,#bodytype)).

As object property in RDF/XML:

<owl:ObjectProperty rdf:ID="betterthan"/>
5.1 Introduction

5.1.3.2 Background knowledge

The difference how we represent the attributes in Aleph is that they are written as predicate with two arguments, while in our algorithm, we treat them as classes and their individual member (e.g. a class of car with sedan body type; we specify any individual that has a body type of sedan is a member of that class). In Aleph, the background knowledge is written as below:

car(car1).
car(car2).
bodytype(sedan).
bodytype(suv).
hasbodytype(car1,sedan).
hasbodytype(car2,suv).

In our algorithm, the background knowledge is written in RDF/XML as below:

<owl:Class rdf:ID="sedan"/>
<owl:Class rdf:ID="suv"/>
<sedan rdf:ID="car1"></sedan>
<suv rdf:ID="car2"></suv>

5.1.3.3 Examples

In ILP, the examples are represented as ground facts with predicate $\text{betterthan}/2$, where the arguments are of type $\text{car}$. The positive examples is written as:

$\text{betterthan(car1,car2)}$, while
Background knowledge

In Aleph

\begin{align*}
\text{car(car1).} & \quad \text{car(car2).} \\
\text{bodytype(sedan).} & \quad \text{bodytype(suv).} \\
\text{hasbodytype(car1,} & \quad \text{hasbodytype(car2,} \\
\text{sedan).} & \quad \text{sedan).} \\
\text{suv).} & \quad \text{suv).}
\end{align*}

As class hierarchy in RDF/XML:

\begin{verbatim}
<owl:Class rdf:ID="sedan"/>
<owl:Class rdf:ID="suv"/>
<sedan rdf:ID="car1"/>
<suv rdf:ID="car2"/>
\end{verbatim}
5.1 Introduction

(a) Representation of background knowledge as class hierarchy
(b) Representation of examples as individuals
(c) Representation of hypothesis language as an object property

Figure 5.1: Problem representation in Protége

5.1.3.2 Background knowledge

The difference on how we represent the attributes in Aleph is that they are written as a predicate with two arguments, while in our algorithm, we treat them as classes and their individual members (e.g. a class of car with sedan body type; we specify any individual that has a body type of sedan is a member of that class). In Aleph, the background knowledge is written as below:


In our algorithm, the background knowledge is written in RDF/XML as below:

<owl:Class rdf:ID="sedan"/>
<owl:Class rdf:ID="suv"/>
<sedan rdf:ID="car1"></sedan>
<suv rdf:ID="car2"></suv>

5.1.3.3 Examples

In ILP, the examples are represented as ground facts with predicate betterthan/2, where the arguments are of type car. The positive examples is written as: betterthan(car1,car2), while...
Examples

In Aleph

\[ \text{betterthan}(\text{car1, car2}). \]

As relations between individuals in RDF/XML:

\[
\begin{align*}
\langle \text{sedan rdf:ID="car1"} \\
\quad \langle \text{betterthan} \ \langle \text{suv rdf:ID="car2"} \rangle \langle /\text{suv} \rangle \ \langle /\text{betterthan} \rangle \\
\quad \langle \text{/} \text{sedan} \rangle
\end{align*}
\]
Examples

In Protégé

<table>
<thead>
<tr>
<th>Description: car10</th>
<th>Property assertions: car10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types</td>
<td>Object property assertions</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Automatic</td>
<td></td>
</tr>
<tr>
<td>Car</td>
<td>Data property assertions</td>
</tr>
<tr>
<td>MediumCar</td>
<td></td>
</tr>
<tr>
<td>NonHybrid</td>
<td>Negative object property assertions</td>
</tr>
<tr>
<td>Suv</td>
<td></td>
</tr>
<tr>
<td>Same Individual As</td>
<td></td>
</tr>
<tr>
<td>Different Individuals</td>
<td></td>
</tr>
</tbody>
</table>

In ILP, the examples are represented as ground facts with predicate `betterthan/2`, where the arguments are of type `car`. The positive examples is written as: `betterthan(car1,car2),` while
We follow the four basic steps used in the Progol [7] / Aleph [8] greedy learning approach:

1. **Select a positive example.**
   • Each instance of the relation can be seen as a pair of object IDs.

2. **Build the bottom clause.**
   • The bottom clause is the conjunction of all non-disjoint class memberships for each object in the pair.

3. **Search.**
   • This step uses greedy best-first search to find a clause consistent with the data.

4. **Remove covered positive examples.**
   • Our algorithm is greedy, removing all covered examples once each highest-scoring clause is added to the current theory.
Search and refinement operator:

- Top down approach
- The bottom clause contains the conjunction of $n$ constraints (of type class membership) on the Domain side, and same number of constraints again on the Range side of the relation.
  - This will produce $n \times n$ possible pairs on the first level of generalisation.
The proposed algorithm involves learning from ordinal data using ILP in Description Logic. The bottom clause contains the conjunction of $n$ constraints (of type class membership) on the Domain side, and the same number of constraints again on the Range side of the relation. This will produce $n \times n$ possible pairs on the first level of generalisation. (We have chosen not to consider hypotheses only constraining one of the arguments.) We evaluate all combinations of constraints, except the ones that imply the same class membership of both arguments (i.e., one argument is better than the other because they both share the same property/class membership) and those that have already been considered. This is illustrated in Figure 1.

Fig. 1: Refinement Operator

We use a common ILP scoring function, $P \times (P - N)$, where $P$ is the number of positive examples covered, and $N$ is the number of negative examples covered. In the case that the solution has the same score as another alternative, Aleph will only return the first solution found. In our algorithm, we consider all the non-redundant hypotheses that are consistent with the examples (i.e., covered zero negative and more than 2 positive). The search will not stop until all the possible combinations have been considered. If we have not found yet a consistent hypothesis, we continue to refine the one with the highest non-negative score, which means that we add a pair of literals to constrain each of the two objects in the relation. We stop at 2 literals each for Domain and Range (this is the same as Aleph's default clause length of 5). Similarly to Aleph, we also consider any examples where we cannot find a consistent generalisation as exceptions. In this case, we add the bottom clause as the consistent rule.

4 Algorithm complexity

We implement our algorithm in one of the DL family of languages, namely ALC (attributive language with complement) [14], the basic DL language which has...
Search and refinement operator:

- **Refinement:**
  - If we have not found yet a consistent hypothesis, we continue to refine the one with the highest non-negative score,
  - We add a pair of literals to constrain each of the two objects in the relation.
  - We stop at 2 literals each for Domain and Range (this is the same as Aleph’s default clause length of 5).

- **Exception:**
  - Similarly to Aleph, we also consider any examples where we cannot find a consistent generalisation as exceptions. In this case, we add the bottom clause as the consistent rule.
Search and refinement operator:

- We evaluate all combinations of constraints, **except**:
  - the ones that imply the same class membership of both arguments
    - e.g. car A has body type sedan and car B has body type sedan
  - those that have already been considered.
Scoring function:

- We use a common ILP scoring function, $P \times (P - N)$,
  - $P = $ positive examples covered, $N = $ negative examples covered.

- In the case that the solution has the same score as another alternative, we consider all the non-redundant hypotheses that are consistent with the examples
  - (i.e. covered zero negative and more than 2 positive).

- The search will not stop until all the possible combinations have been considered.
EVALUATION

Dataset

We use two publicly available datasets, car preferences [8] and sushi preferences [9] with the statistics below:

- Number of items: 10 items
- Number of pairs: 45 preference pairs (2-combination of 10)
- Number of participants: 60 users
- Number of attributes:
  - car dataset has 4 attributes (body type, transmission, fuel consumption and engine size)
  - sushi dataset has 7 attributes (style, major, minor, heaviness, how frequently consumed by a user, price and how frequently sold)
Evaluation method

- The goal:
  - to assess the accuracy of the predictive power of each algorithm to solve the preference learning problem.

- We compare our algorithm with three other machine learning algorithms:
  - SVM, the Matlab CART Decision Tree (DT) learner, and Aleph
EVALUATION

Results

Experiment 1: We learn each individual preferences and test them using 10-fold cross validation.

Table 1: Mean and standard deviation of 10-fold cross validation test

<table>
<thead>
<tr>
<th>Dataset</th>
<th>SVM</th>
<th>DT</th>
<th>Aleph</th>
<th>Our algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>car dataset</td>
<td>0.8317±0.12</td>
<td>0.7470±0.10</td>
<td>0.7292±0.08</td>
<td><strong>0.8936±0.05</strong></td>
</tr>
<tr>
<td>sushi dataset</td>
<td>0.7604±0.09</td>
<td>0.8094±0.06</td>
<td>0.7789±0.06</td>
<td><strong>0.9302±0.03</strong></td>
</tr>
</tbody>
</table>

ANOVA test result:
- $p-value = 2.0949 \times 10^{21}$ for the car dataset
- $p-value = 7.3234 \times 10^{36}$ for the sushi dataset.
- There is a significant difference amongst the algorithms
EVALUATION

Results

Experiment 2:
We set different proportion of training examples and test it on 10% of test data. For a more robust result, we validate each cycle with 10-fold cross validation.
Sample solutions

- By using DL representation, we can produce more readable results for a novice user

  Automatic □ Hybrid betterthan MediumCar □ Suv

- Aleph produces rules, such as:

  betterthan(A,B) :-hasfuelcons(B,nonhybrid), hasbodytype(B,suv).
CONCLUSIONS

- We have shown that the implementation of ILP in DL can be useful to learn a user’s preference from pairwise comparisons.
- Currently, our algorithm uses the Closed World Assumption, which makes it easier to find a consistent hypothesis and test the coverage.
- In term of accuracy, we have shown that our proposed algorithm outperformed the other algorithms even with the smaller number of training examples.
FUTURE WORK

We are currently working to address the following limitations of our algorithm:

– Working with more complex class hierarchy (includes negations, unions and quantifiers)
– Working under the Open World Assumption (OWA)
– The improvement of the system performance and scalability
  • e.g. implementing triplestore

• We plan to implement our algorithm in Recommender System application and invite the real users to evaluate using larger real dataset from Autotrader (7361 cars)
Thank you

Any question or feedback?