Learning from Ordinal Data with Inductive Logic Programming in Description Logic

Nunung Nurul Qomariyah and Dimitar Kazakov

Artificial Intelligence Group
Computer Science, University of York, UK
{nq516,dimitar.kazakov}@york.ac.uk

Abstract. Here we describe a Description Logic (DL) based Inductive Logic Programming (ILP) algorithm for learning relations of order. We test our algorithm on the task of learning user preferences from pairwise comparisons. The results have implications for the development of customised recommender systems for e-commerce, and more broadly, wherever DL-based representations of knowledge, such as OWL ontologies, are used. The use of DL makes for easy integration with such data, and produces hypotheses that are easy to interpret by novice users. The proposed algorithm outperforms SVM, Decision Trees and Aleph on data from two domains.

Keywords: Inductive logic programming, pairwise comparisons, preference learning, machine learning, description logics

1 Introduction

ILP is now a well-established research area where machine learning meets logic programming [1], which has shown the potential to address many real world problems [2]. Description Logics (DL) are a family of languages representing domain knowledge as concept descriptions. DL can be considered as fragments of First Order Logic (FOL). Because of their well-defined semantics and powerful inference tools, DL have been the representation of choice for ontologies, including applications, such as the Semantic Web. ILP algorithms using DL representation have the potential to be applied to large volumes of linked open data and to benefit from the tools available for such data, e.g. IDEs such as Protégé, *triplestores* providing efficient storage and retrieval, and *reasoners*, which can test the data for consistency and infer additional knowledge. The chosen application area of Preference Learning (PL) aims to induce predictive preference models from empirical data. The automation of PL has now become essential in many e-commerce applications following the current call for customer personalisation.

We build on previous work combining ILP and DL: DL-Learner [9] aims to find a correct class description in a specific ontology from a set of positive and negative examples of individuals. It builds a hypothesis in form of class description, which can contain conjunction, disjunction and existential quantification.

DL-Learner itself develops previous work on systems, such as YinYang [10] and DL-FOIL [11]. Kietz's [12] and Konstantopoulos's [13] work is also relevant in this context.

While DL-Learner can only learn class definitions, we aim to learn definitions of object properties. Here we focus on the binary relation of order, which is transitive and anti-reflexive. We evaluate our algorithm on two datasets, one on car preferences [3] and the second on sushi preferences [4]. Both datasets provide pairwise preference choices of multiple users. Our aim is to learn each user's individual preferences using a DL-based representation, and an algorithm inspired by the Progol/Aleph family of ILP tools.

2 Problem Representation

To learn from examples of transitive anti-symmetric relations, 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, and the same examples in reverse order as negative examples. All attributes of either item in the pair (e.g. car A has type Sedan) are available as background knowledge. This will be expressed in a class hierarchy in DL (e.g. car A is a member of subclass Sedan of class Car). The specific relation we want to learn, an object property reflecting a type of order, needs to be specified in the hypothesis language. A simple example using the Progol [5]/Aleph [6] convention appears below:

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

3 The Proposed Algorithm

We implement our algorithm in Java using the OWL API [7] library to handle DL. We follow the four basic steps used in the Progol/Aleph 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. We use a top down approach similar to the one in Progol/Aleph. The algorithm proceeds by considering an increasing number of properties constraining each of the two objects in the relation. As 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. (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. *X is better than Y because they both share the same property/class membership)* and those that have already been considered. This is illustrated in Figure 1.

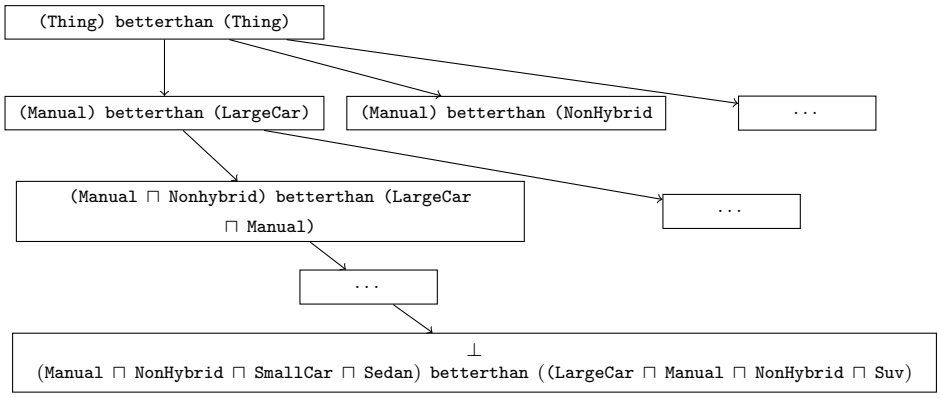


Fig. 1: Refinement Operator

We use a common ILP scoring function, $P \times (P - N)$, where P is the number of positive examples covered, and N – 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 $\mathcal{ALC}$ (attributive language with complement) [14], the basic DL language which has

the least expressivity. $\mathcal{ALC}$ allows one to construct complex concepts from simpler ones using various language constructs. The capabilities include direct or indirect expression, e.g. concept disjointness, domain and range of roles, including the empty role.

The most expensive process is the membership checking part for every possible hypothesis. This is used for scoring the hypothesis. For every single hypothesis the reasoner needs to check the coverage of each hypothesis. One possible way to reduce the complexity is by minimising the search tree and checking the redundancy without reducing the accuracy.

5 Evaluation

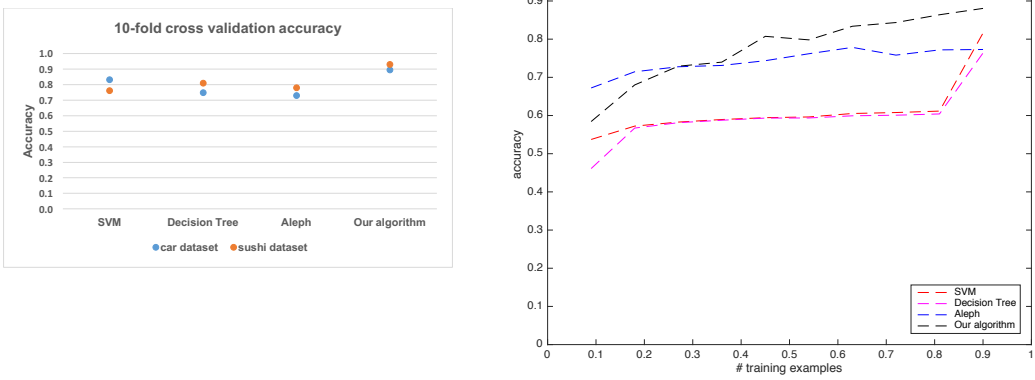
Dataset. We use two publicly available preference datasets [3] [4]. Both the sushi and the car datasets have 10 items to rank which leads to 45 preference pairs per user. We take 60 users from each dataset and perform 10-fold cross validation for each user's individual preferences. The car dataset has 4 attributes: body type, transmission, fuel consumption and engine size, while the sushi dataset has 7 attributes: style, major, minor, heaviness, how frequently consumed by a user, price and how frequently sold. Despite the difference in the number of attributes in the two datasets, we found that the maximum clause length of 4 (in Aleph and in our algorithm) is sufficient to produce consistent hypotheses.

Evaluation method. The goal of this evaluation is 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. SVM is a very common statistical classification algorithm that used in many domains. Similar work of pairwise preference learning was performed by Qian et. al. [8] show that SVM can also be used to learn in this domain. Both DT and Aleph can be included in the evaluation since both of them are logic based learner, where the first is in propositional logic and the latter is in First Order Logics.

We learn each individual preferences and test them using 10-fold cross validation. The result is shown in Table 1 and Figure 2a. According to the ANOVA test, the result shows that there is a significant difference amongst the algorithms with the *p-value* 2.0949×10^{-21} for the car dataset and 7.3234×10^{-36} for the sushi dataset.

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

	SVM	DT	Aleph	Our algorithm
car dataset	0.8317±0.12	0.7470±0.10	0.7292± 0.08	**0.8936**±0.05
sushi dataset	0.7604±0.09	0.8094±0.06	0.7789±0.06	**0.9302**±0.03



(a) 10-cross validation accuracy (b) Accuracy by varying number of training examples

Fig. 2: Evaluation results

We also perform several experiments with the algorithms by varying the proportion of training examples and test it on 10% of examples. For a more robust result, we validate each cycle with 10-fold cross validation. The result of this experiments is shown in Figure 2b. We show that our algorithm still work better even with the smaller number of training examples.

Sample solutions found. Our algorithm can produce more readable results for a novice user compared to Aleph. An example of consistent hypothesis found by our algorithm is shown below:
`Automatic` $\sqcap$ `Hybrid betterthan MediumCar` $\sqcap$ `Suv`
While Aleph produces rules, such as:

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

6 Conclusion and Further Work

In this paper, we have shown that the implementation of ILP in DL can be useful to learn a user's preferences from pairwise comparisons. We are currently working to address the following limitations of our algorithm:

- We only consider one level class hierarchy in the ontology for simplicity. In the real world, the class hierarchy can be more complex.
- Currently, our algorithm uses the Closed World Assumption, which makes it easier to find a consistent hypothesis. This is not in line with the fact that most DL-based knowledge databases and their reasoners operate under the Open World Assumption.

– In term of accuracy, we show that our algorithm outperformed the other algorithms, but this is a time consuming process. In order to produce a complete and consistent hypothesis, our algorithm takes much longer than the three baseline algorithms. The algorithm optimisation method is quite simple. We need to work on the improvement of this aspect without reducing the accuracy performance.

References

1. Muggleton, S., De Raedt, L., Poole, D., Bratko, I., Flach, P., Inoue, K. and Srinivasan, A.: ILP Turns 20. In: Machine Learning, vol. 86 no. 1, pp. 3–23. Springer, Heidelberg (2012)
2. Gulwani, S., Hernandez-Orallo, J., Kitzelmann, E., Muggleton, S.H., Schmid, U. and Zorn, B.: Inductive Programming Meets the Real World. In: Communications of the ACM, vol. 58 no. 11, pp. 90–99. ACM, New York (2015)
3. Abbasnejad, E. and Sanner, S. and Bonilla, E. V. and Poupart, P.: Learning Community-based Preferences via Dirichlet Process Mixtures of Gaussian Processes. In: Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI) (2013)
4. Kamishima, T.: Nantonac Collaborative Filtering: Recommendation Based on Order Responses. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 583–588. ACM, New York (2003)
5. Muggleton, S.: Inverse Entailment and Progol. In: New Generation Computing, vol. 13 no. 3-4, pp. 245–286 (1995)
6. Srinivasan, A.: The Aleph Manual. In: Technical Report. Computing Laboratory, Oxford University (2000), `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/`
7. Horridge, M. and Bechhofer, S.: The OWL API: A java api for owl ontologies. In: Semantic Web, vol. 2 no. 1, pp. 11–21, (2011)
8. Qian, L., Gao, J. and Jagadish, H. V.: Learning User Preferences by Adaptive Pairwise Comparison. In: Proceedings of the VLDB Endowment, vol. 8 no. 11, pp. 1322–1333. VLDB Endowment (2015)
9. Lehmann, J.: DL-Learner: Learning Concepts in Description Logics. In: The Journal of Machine Learning Research, vol. 10, pp. 2639–2642. (2009)
10. Iannone, L., Palmisano, I., Fanizzi, N.: An Algorithm Based on Counterfactuals for Concept Learning in The Semantic Web. In: Applied Intelligence, vol. 26 no. 2, pp. 139–159. Springer, Heidelberg (2007)
11. Fanizzi, N., d'Amato, C., and Esposito, F.: DL-FOIL Concept Learning in Description Logics. In: Proceedings of Inductive Logic Programming, ser. LNCS, vol. 5194, pp. 107–121. Springer, Heidelberg (2008)
12. Kietz, J.: Learnability of Description Logic Programs. In: Proceedings of Inductive Logic Programming, ser. LNCS, vol. 2583, pp. 117–132. Springer, Heidelberg (2002)
13. Konstantopoulos, S. and Charalambidis, A.: Formulating Description Logic Learning as An Inductive Logic Programming Task. In: Proceedings of IEEE World Congress on Computational Intelligence, pp. 1–7 (2010)
14. Schmidt-Schauß, M. and Smolka, G.: Attributive concept descriptions with complements. In: Artificial Intelligence, vol. 48, no. 1, pp.1–26 (1991)