Learning from Ordinal Data with Inductive Logic Programming in Description Logic

Nunung Nurul Qomariyah
AI Group, Computer Science
University of York, UK
nq516@york.ac.uk

Dimitar Kazakov
AI Group, Computer Science
University of York, UK
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.

1 Introduction

ILP is now a well-established research area where machine learning meets logic programming [18], which has shown the potential to address many real world problems [6]. Description Logics (DLs) are a family of languages representing domain knowledge as descriptions of *concepts*, *roles* and *individuals*. DLs 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. DLs are subsets of FOL, and the representation they use has a variable-free syntax. This is claimed to make the knowledge base statements easier to read than the corresponding FOL formulae [2]. Traditionally, ILP algorithms use the Closed World Assumption (CWA), while DLs operate under the Open World Assumption (OWA). Existing attempts to apply ILP to DL data have explored both possibilities, as shown in Section 3.1.

The chosen application area of Preference Learning (PL) aims to induce predictive preference models from empirical data, e.g. by assigning an absolute rank to the choices available [8]. Online vendors already employ methods, such as collaborative filtering, which allow to produce selective lists of the most popular products among users with similar tastes [16].

The automation of PL has now become essential in many e-commerce applications following the current call for customer personalisation. The method of pairwise comparisons, which we focus on, is relatively new in

In: Nicolas Lachiche, Christel Vrain (eds.): Late Breaking Papers of ILP 2017, Orléans, France, September 4-6, 2017, published at http://ceur-ws.org

preference learning, although there are exceptions [3, 10, 20, 19, 14]. Balakrishnan and Chopra [3] published a research study on pairwise preferences by using a Bayesian framework. Qian et al. [19] published a similar idea, proposing an approach to learn user preferences by using pairwise comparisons to explore each attribute in turn through "orthogonal queries" and then applying linear SVM to approximate the preferences. Similarly, Jensen et al. [10] apply a Gaussian Process regression model for comparisons between music tracks. Another similar work was performed by Rokach and Kisilevich [20] which uses lazy decision trees with pairwise comparisons at each node.

Most studies on pairwise preference learning use statistical machine learning approaches while in this paper, we propose a logic-based approach. There is one other attempt at using ILP for preference learning [14], which proposes a framework for learning weak constraints in Answer Set Programming (ASP). We build on previous work combining ILP and DL, namely, the DL-Learner [15], which 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 the form of a class description, which can contain conjunction, disjunction and existential quantification. DL-Learner itself develops previous work on systems, such as YinYang [9] and DL-FOIL [5]. Kietz's [12] and Konstantopoulos' [13] work is also relevant in this context. While DL-Learner can only learn class definitions, we aim to learn definitions of relations. Here we focus on the binary relation of order, and in particular, strict order, a relation that is transitive, anti-reflexive and anti-symmetric. 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. We evaluate our algorithm on two datasets, one on car preferences [1], the second on sushi preferences [11]. Both datasets provide pairwise preference choices of multiple users.

The rest of the paper is organized as follows: in Section 2 we explain the learning task in terms of both logic programming and DL to make the problem clearer to readers from either background. We then propose an algorithm based on ILP in Section 3 and provide the analysis of the evaluation in Section 4. Finally, we conclude our work and provide our plan for further work in Section 5.

2 Problem Representation

DLs represent the world in terms of *concepts*, *objects* and *roles*. Concepts can be seen as a formal definition of classes in the application domain, e.g. one can define a father as a man who has at least one child. Concepts have two functions: (1) to describe a collection of objects and (2) to describe the properties that a class should hold. Objects are the entities that belong to one concept or another. Roles represent binary relationships between objects, e.g. John is Anna's father. These can also be used to describe object properties, e.g. John's age is 42. In FOL terminology, objects correspond to unique constants (i.e. ground terms), concepts correspond to unary predicates, and roles correspond to binary predicates. All this information is stored in the form of a *knowledge base* which comprises of two components, an assertional part (*ABox*) and a terminological part (*TBox*).The TBox is a finite set of General Concept Inclusions (GCI) and role inclusions. It corresponds to the notion of a schema in a relational database. The assertional set in the knowledge base is called the ABox. It is used to state properties of individuals. In a relational database, this is the actual data, while in the FOL it corresponds to facts (*aka* ground terms).

The knowledge representation in DL can be expressed using the Resource Description Framework (RDF), which is one of the most powerful general-purpose knowledge representation languages. The syntax of RDF is based on triples in the form of subject-predicate-object expressions. An RDF triple is a statement expressing a relation ("predicate") between the object in its first argument, and the object or literal in its third argument (e.g. "Anna-has_father-John"). In this section, we use Turtle[1] syntax to describe RDF data[2]

The preference learning problem is defined as follows:

Given: a set of individuals, a class hierarchy, a mapping from individuals to classes, and a set of preferences represented as pairs of individuals, where the first individual is preferred over (strictly better than) the second,

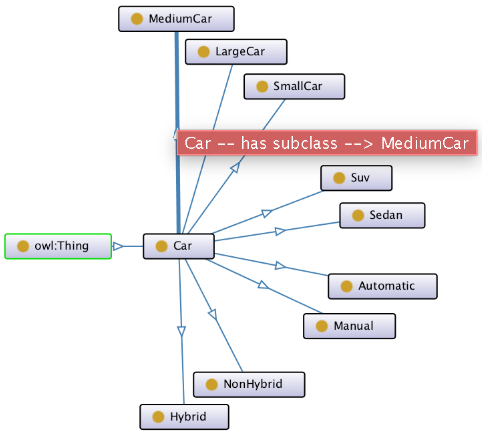
Find: a disjunction of axioms defining the domain and range of the relation `betterthan` (where each axiom is expressed as a conjunction of classes) that is complete and consistent with the given preferences.

This is a supervised learning problem as the user assigns one of two possible labels for each pair of items after

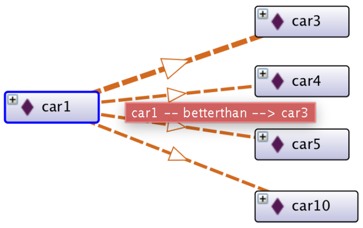
[1]https://www.w3.org/TR/turtle/
[2]Turtle stands for Terse RDF Triple Language.

considering their attributes. We then use these labels to search for a definition of the Domain and Range class memberships that render the relation true.

We propose an approach to learn relations and apply the resulting system to learn strict order (i.e. better than). The properties of strict order relation are *anti-symmetric*, which means that if item A is better than item B, then in any case item B cannot be better than item A, unless A=B. That special case is then excluded by the additional assumption of `betterthan` being anti-reflexive (i.e. X cannot be seen as better than itself). It is also *transitive*, which means whenever item A is better than item B, and item B is better than item C, then item A is better than item C. The ontology reasoner can always be used to extract all pairs of items satisfying the



(a) A simple car class hierarchy



(b) The user annotation is translated into the object property "betterthan".

Figure 1: Problem representation

preference relation as a result of applying its transitivity property. We add these inferred examples to the set of positive examples so that the complete closure is used by the learning algorithm. We also use the anti-symmetry property to produce all negative examples as a 'mirror image' of the positive (after completing their transitive closure), i.e., all pairs derived from a positive example through the swap of the first and second element in the pair.

The representation of a sample preference learning problem in DL is shown in Figure 1. The class hierarchy is given to the system as an input. We evaluate our algorithm using a simple class hierarchy as shown in Figure 1a in order to make the solutions comparable to the Aleph representation.

2.1 Hypothesis Language

The aim of ILP is to find a theory that is complete (it covers all the given positive examples) and consistent (it covers no negative examples). In our algorithm learning from RDF data, we build a hypothesis for an object property we want to learn, as in the case of the property `betterthan`. The fact that `betterthan` is an object property is described in Turtle as shown below:

```
myontology:betterthan rdf:type owl:ObjectProperty.
```

Each of the possible hypotheses about this relation is then described as a pair of class definitions, which specify the membership of the domain $\mathcal{D}$ and the range $\mathcal{R}$ of the relation. The same hypothesis language can be described in Aleph notation as the following mode declarations:

```
:- modeh(1,betterthan(+car,+car)).
:- modeb(1,hasbodytype(+car,#bodytype)).
:- modeb(1,hasfuelcons(+car,#fuelconsumption)).
:- modeb(1,hasbodytype(+car,#bodytype)).
:- modeb(1,hastransmission(+car,#transmission)).
:- modeb(1,hasenginesize(+car,#enginesize)).
```

2.2 Background Knowledge

Our algorithm represents background knowledge through classes and their membership, as shown in the example below (which uses Turtle syntax):

```
myontology:Sedan rdf:type owl:Class ;
               rdfs:subClassOf myontology:Car ;
               owl:disjointWith myontology:Suv .

myontology:car1 rdf:type owl:NamedIndividual ,
                       myontology:Car ,
                       myontology:Manual ,
                       myontology:NonHybrid ,
                       myontology:SmallCar ,
                       myontology:Suv .
```

The same background knowledge can be spelt out in Aleph as follows:

```
car(car1).        %type predicate
bodytype(sedan).  %type predicate
hasbodytype(car1,suv).        %bk
hasfuelcons(car1,nonhybrid).  %bk
hastransmission(car1,manual). %bk
hasenginesize(car1, small).   %bk
```

2.3 Examples

In our algorithm, the set of positive examples is defined by the user's preferences, and the following pre-processing step that computes the transitive closure of these preferences, which are then negated to produce all negative examples. The following Turtle code expresses that car1 is better than car3, car4, car5, and car10:

```
 myontology:car1 myontology:betterthan myontology:car3 ,
                                     myontology:car4 ,
                                     myontology:car5 ,
                                     myontology:car10 .
```

In traditional ILP syntax, the same examples are represented as ground facts of the predicate `betterthan/2`, where the arguments are of type `car`. So, the positive examples in Aleph are written as: `betterthan(car1,car3)`, and the negative, obtained by their reversal, as `:-betterthan(car3,car1)`.

3 Proposed Algorithm

We propose an algorithm called APARELL (Active[3] Pairwise Relation Learner) which learns the relation of strict order in DL. We implement our algorithm in Java using the OWL API[4] [7] and RDF4J API [5] to handle DL. The algorithm is shown in Algorithm 1. 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 constructed from the conjunction of all non-disjoint classes of which each individual in the pair is a member.
3. **Search**. This step is to find all generalisations of the bottom clause of the same, lowest possible, length that are consistent with the data.

[3]The active learning algorithm feature has not been described or used in this work.
[4]http://owlapi.sourceforge.net/
[5]http://rdf4j.org/

4. **Remove covered positive examples**. Our algorithm is greedy in its treatment of positive training examples. We remove all covered positive examples once all clauses found by the search step are added to the current theory.

3.1 Search and Refinement Operator

We use a top-down approach similar to the one in Progol/Aleph starting from the shortest clause possible. Note that for the specific class of problems, namely, learning strict order, the most general, unconstrained definition stating that for any pair of objects, the first is better than the second, is true in exactly 50% of the cases for every pair $\langle x, y\rangle, x \neq y$, as where this is true, the relation will not hold for the pair $\langle y, x\rangle$. It will also always be false for pairs of type $\langle x, x\rangle$. This is why the algorithm proceeds by considering an increasing, and strictly positive number of properties (literals) constraining either object in the relation. 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. 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)* or those that have already been considered. An example of the refinement operator steps starting from a positive example `car1` is better than `car3` is illustrated in Figure 2 (please see Section 4.1 for a description of the dataset of car preferences).

We express the coverage of a clause through P and N, where P – 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 new clauses that cover at least two positive examples and none of the negatives. (This is done to ensure consistency, and that at least a minimum level of generalisation takes place.) The search will not stop until all the possible combinations at each level have been considered. The resulting theory is a disjunction of clauses. Therefore, any test example covered by one of them is classified as positive.

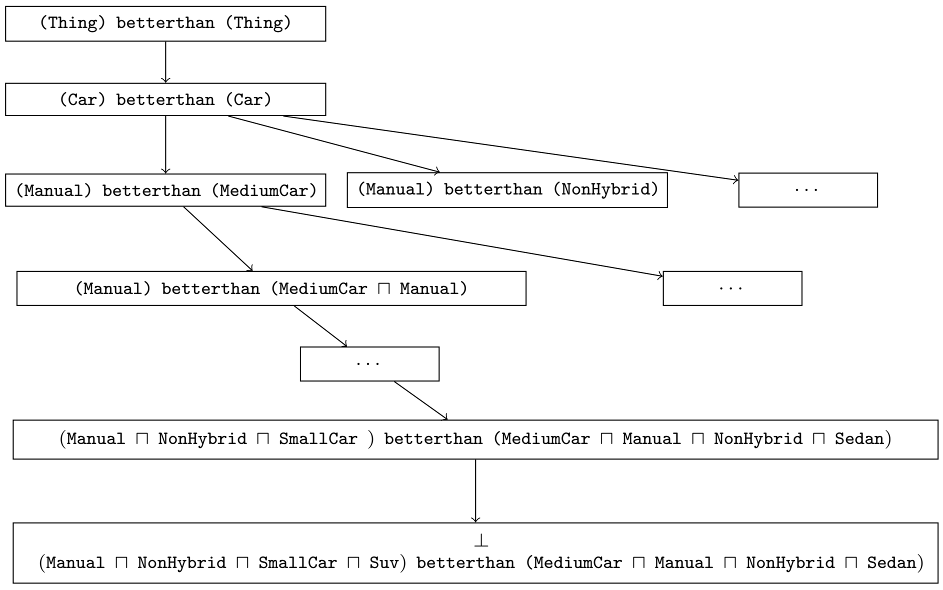

(Thing) betterthan (Thing)
(Car) betterthan (Car)
(Manual) betterthan (MediumCar)
(Manual) betterthan (NonHybrid)
...
(Manual) betterthan (MediumCar ⊓ Manual)
...
...
(Manual ⊓ NonHybrid ⊓ SmallCar) betterthan (MediumCar ⊓ Manual ⊓ NonHybrid ⊓ Sedan)
⊥
(Manual ⊓ NonHybrid ⊓ SmallCar ⊓ Suv) betterthan (MediumCar ⊓ Manual ⊓ NonHybrid ⊓ Sedan)

Figure 2: Refinement operator search graph

Our algorithm retains *all* consistent clauses generalised from a given positive example, rather than just one of them, as Aleph would have done. (At the same time, both algorithms discard the positive example from the training set after this generalisation step in a greedy learning manner.) This is the most important difference between the two algorithms and a possible explanation for any observed difference in their performance.

If a consistent clause has not been found yet, the algorithm continues to refine the current candidate by adding one literal to constrain either object in the relation. Similarly to Aleph, when APARELL cannot find a consistent generalisation, it adds the bottom clause itself to the theory.

We implement our algorithm using the Closed World Assumption (CWA). For the problem of learning strict order, it makes virtually no difference whether our system operates under the CWA or OWA. Under the OWA, we learn two hypotheses: one as mentioned before, the other with the positive and negative examples swapped.

Algorithm 1 APARELL

```
1: Input : background knowledge $B$,
        a set of positive examples $E^+ = \{\langle e_1, e_2\rangle, \dots, \langle e_n, e_m\rangle\}$,
        a set of negative examples $E^- = \{\langle e_2, e_1\rangle, \dots, \langle e_m, e_n\rangle\}$,
        maximum number of literals ($depth\ limit$) $l$
        a relation name $r = betterthan$
2: Output : A theory $T$ represented as a set of clauses, each defining a pair of concepts specifying the relation's
   domain and range
3: set $T = \emptyset$ /* initialization*/
4: set $x_1 = \emptyset$ /* the list of constraints on Domain */
5: set $x_2 = \emptyset$ /* the list of constraints on Range */
6: set $flag$ = false /* whether a consistent generalisation of the bottom clause has been found */
7: while $E^+$ is not empty do
8:     set $clause\_size = 2$ /* initial value for $|x_1| + |x_2|$ */
9:     set $x_1 = \emptyset$, $x_2 = \emptyset$, $flag$=false /* reset in every loop*/
10:    select $e^+ \in E^+$ to be generalised, and define $\langle e_1, e_2\rangle = e^+$
11:    /* build the bottom clause $\langle x_1, x_2\rangle$ for $e^+$ as follows: */
12:    set $x_1 = \{C_1, \dots, C_m\}$ $\forall C_i$ such that $e_1 \in C_i$
13:    set $x_2 = \{D_1, \dots, D_n\}$ $\forall D_i$ such that $e_2 \in D_i$
14:    while $flag$==false and $clause\_size < l$ do
           /* search (top-down) through all generalisations of the bottom clause */
15:        for all $x'_1 \subseteq x_1, x'_2 \subseteq x_2, |x'_1| + |x'_2| == clause\_size$ do
16:            if $x'_1$ $betterthan$ $x'_2$ is consistent and more general than $e^+$ then
17:                add the clause to $T$
18:                set $flag$ = true
19:            end if
20:        end for
21:        if $flag$==false and $clause\_size < l$ then
22:            set $clause\_size = clause\_size + 1$ /* increment clause size */
23:        else if $flag$==false and $clause\_size == l$ then
24:            add $e^+$ to $T$ and remove $e^+$ from $E^+$
25:        end if
26:    end while
27:    if flag==true then
28:        remove covered positive examples from $E^+$
29:    end if
30: end while
31: Return $T$
```

For the given domain (of learning strict order), the second hypothesis (i.e. the one that being swapped between the negative and positive examples) is almost the exact negation of the first (modulo the choice of a training sample). The resulting coverage is therefore almost identical to the one of the CWA hypothesis.

3.2 Search in a Complex Class Hierarchy

In the case of a more complex class hierarchy that consists of more than one level, we consider the hypothesis search over all the inferred class memberships of each individual. For example, if the class hierarchy and its membership in Figure 3 are given to the system, the algorithm progresses as follows:

1. **Select a positive example.** As an example, we choose car 1 is better than car 3 to be generalised. Please see Table 1 for the properties of the cars.

2. **Build the bottom clause**. In this step, we use the reasoner to infer all the classes which include car 1 and car 3 as their direct and indirect membership. The bottom class of the above example is shown as follows:

   ```
   (Car and EconomyCar and FamilyCar and Manual and
            NonHybrid and SmallCar and Suv) betterthan
            (Car and Manual and MediumCar and NonHybrid and Sedan)
   ```

   The above bottom clause is only used to guide the refinement search, as
   `SmallCar` $\sqsubseteq$ `EconomyCar` $\sqsubseteq$ `Car`. We do not consider clauses referring to the membership of the intersection between a class and its superclass, as this will be the same as the membership of the class itself. For example, the membership of `SmallCar` $\sqcap$ `EconomyCar` is {`car1,car8`}, which is the same as the membership of `SmallCar` itself.

3. **Search**. The search begins from the shortest clause length considered (2 literals) as shown below:

   ```
   Car betterthan Manual
   Car betterthan MediumCar
   Car betterthan NonHybrid
   Car betterthan Sedan
   EconomyCar betterthan Car
   EconomyCar betterthan Manual
   ...
   ```

   In the case that we have not found any consistent hypothesis of length 2, this parameter is increased to 3, etc.:

   ```
   EconomyCar and FamilyCar betterthan Car
   EconomyCar and FamilyCar betterthan Manual
   ...
   ```

   It should be noted that the algorithm skips the evaluation of the same value comparisons like: `Car betterthan Car` to speed up the search.

4. **Remove covered positive examples**. We count the coverage for each hypothesis we build and remove the covered positive examples from the training dataset.

3.3 Current State

Our algorithm can handle a multi-level class hierarchy, but, in order to limit the search, we only allow conjunctions of literals in the clauses, effectively limiting the language to EL description logic. With this limitation, we can still produce accurate models for our test cases, and a result in a form that is easy to interpret. The most expensive process is checking the class membership (by using the ontology reasoner) for every possible hypothesis. This is

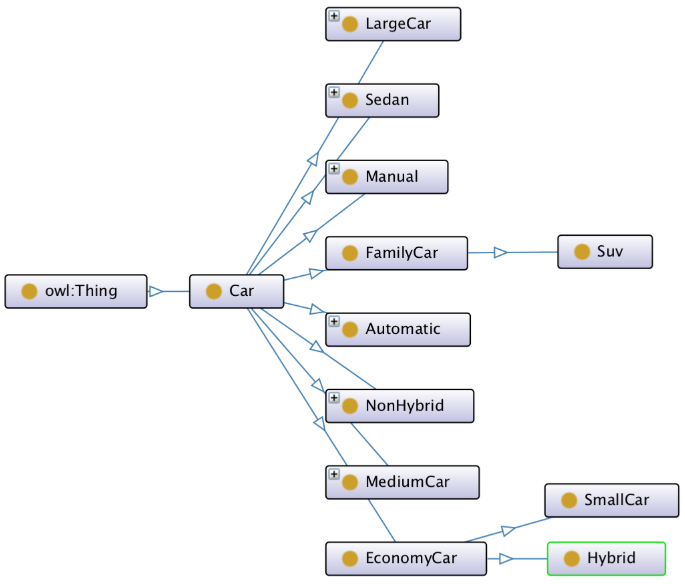


Figure 3: An example of more than one level of car class hierarchy

used for scoring the hypothesis coverage. The search in our algorithm follows breadth-first search with limited depth (specified as the l parameter).

Another property of our proposed algorithm is that, unlike Aleph, we do not provide a single consistent clause covering a given positive example, but add to the resulting theory all such clauses of the minimal possible length. While this can produce a more accurate result, the learning process takes longer than in Aleph. For example, given 45 training examples in which each item has 4 attributes, our algorithm takes 409 ms to finish while Aleph performs faster in just 88 ms.

4 Evaluation

4.1 Datasets

We use two publicly available preference datasets [1, 11]. 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

Table 1: Car Dataset - Item Descriptions

Car ID	Bodytype	Transmission	Fuel	Engine size
1	suv	manual	non-hybrid	small
2	sedan	automatic	hybrid	large
3	sedan	manual	non-hybrid	medium
4	sedan	manual	non-hybrid	large
5	suv	manual	non-hybrid	medium
6	suv	automatic	hybrid	medium
7	sedan	automatic	hybrid	medium
8	suv	automatic	hybrid	small
9	sedan	automatic	non-hybrid	medium
10	suv	automatic	non-hybrid	medium

validation for each user's individual preferences. The car dataset has 4 attributes: body type, transmission, fuel consumption and engine size [1], as shown in Table 1. 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.

The second dataset used in the experiments is about sushi preferences[6] which has been published by Kamishima [11]. The dataset contains user individual preferences about 10 different sushi types. The users

[6]http://www.kamishima.net/sushi/

were asked to sort the sushis according to their preferences in an ascending order. For the experiment, we generate the pairs order from each set of individual preferences. Similar to the previously mentioned car dataset, 45 pairs of sushi preferences are produced for 10 types of sushi (which represents all possible pairs). In this dataset, there were 5000 users involved in the survey. The sushi preference dataset is quite large when compared to the car preference dataset as it has 7 attributes describing each type of sushi. With a dataset of such size, we can test the performance of our algorithm and evaluate how it grows.

4.2 Evaluation Method

The goal of this evaluation is to assess the accuracy of the predictive power of our algorithm to solve the preference learning problem. To do this, we set up four experiments as below:

1. Assess the accuracy of our algorithm compared to the three baseline algorithms, SVM, Aleph and DT, on 60 users in car dataset and sushi dataset. Here, we also perform an ANOVA test and a post-hoc test to assess which algorithms significantly differ.

2. Assess the accuracy and the performance of our algorithm on a larger dataset by conducting an experiment on 5000 users of the sushi dataset.

3. Assess the potential of our algorithm to learn relations from a more complex class hierarchy by using the car dataset.

4. Assess the accuracy of our algorithm on training examples of different size, and compare it to the three baseline algorithms.

Accuracy on 60 users.

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, which is used in many domains. Previous work of pairwise preference learning was performed by Qian et al. [19] showing that SVM can also be used to learn in this domain. Both DT and Aleph are learners expressing their models in logic, where the former uses propositional logic and the latter – (a subset of) First Order Logic (namely, Horn clauses).

We build a simple class hierarchy as explained in Section 2 for each dataset. We learn from each set of individual preferences and evaluate the model by using 10-fold cross validation. We repeat the same test for all users then calculate the average accuracy. The result is shown in Table 2. In this experiment, the search stopped at a maximum length of 4 literals (this is the same as Aleph's default clause length of 4). According to the ANOVA test with $\alpha = 0.05$, the result shows that there is a significant difference amongst the algorithms with a *p-value* of 1.14×10^{-21} for the car dataset and 2.97×10^{-3} for the sushi dataset. ANOVA is conceptually similar to multiple two-sample t-tests, but is more conservative (results in less type I error). After performing the ANOVA test, we find which algorithms are significantly different by using Fisher's Least Significant Difference (LSD). The result of this post-hoc test is shown in Table 3. Please note that the result of this 10-fold cross validation may have introduced a bias in terms of absolute performance levels due to the fact that the use of transitivity chains (closures) may have created an overlap between training and test data. However, the same training and test data have been used to compare all algorithms, so the results for their comparison remain unaffected (are not biased).

Table 2: Mean and standard deviation of 10-fold cross validation test

	SVM	DT	Aleph	Our algorithm
car dataset	0.8264±0.12	0.7470±0.10	0.7292±0.08	**0.8456**±0.06
sushi dataset	0.7604±0.09	0.7869±0.07	0.7789±0.06	**0.8138**±0.07

Accuracy on 5000 users.

In this experiment, our algorithm is still showing the highest accuracy amongst the other three baseline algorithms. The average accuracy on individual preferences from 5000 sushi dataset users is shown in Table 4. In the first experiment, we have already evaluated the mean difference amongst the algorithms using ANOVA and a post-hoc test. In here, the *p*-value of ANOVA (α=0.05) test is 0. This is a common case in large datasets. Performing statistical test to analyze the mean differences in a large dataset can be problematical as *p*-value

Table 3: Post-Hoc Fisher's Least Significant Difference (LSD)

Algorithm 1	Algorithm 2	Sushi dataset		Car dataset	
		$p - value$	means diff.	$p - value$	means diff.
Aleph	DT	0.561	same	0.292	same
Aleph	Our algorithm	0.011	different	0	different
Aleph	SVM	0.177	same	0	different
Our algorithm	DT	0.049	different	0	different
Our algorithm	SVM	0	different	0.257	same
DT	SVM	0.054	same	0	different

tends to drop quickly to zero. From the result shown in Table 4, the low standard deviation rate can be a good indication that the accuracy of our algorithm is excellent in any case.

Table 4: Accuracy on 5000 users

	SVM	DT	Aleph	Our algorithm
mean	0.7599	0.8004	0.7867	0.8150
stdev	0.0912	0.0612	0.0697	0.0604

Accuracy on more complex class hierarchy.

As an additional experiment, we evaluate the car dataset with the complex class hierarchy (see Figure 3) using literal depth limit=4. The accuracy of our algorithm is performed using 10-fold cross validation. The result of the experiment with the complex class hierarchy is a mean of 0.8409 and standard deviation of 0.1405. The accuracy result shows no difference between using a simple class hierarchy and a complex class hierarchy.

Accuracy on different training examples size.

We 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 these experiments is shown in Figure 4. We show that our algorithm still works better even with the smallest number of training examples.

Algorithm performance.

We run another test to see the algorithm performance by different clause length settings (see Table 5). The evaluation is performed on both datasets by using a 90:10% training-test split. We also record the algorithm execution times of 60 users $\times$ 90% of examples (2,382 total examples in car dataset and 2400 total examples in sushi dataset, please note that some of the positive examples are removed during the search). The algorithm was executed on Java 8 Eclipse IDE with 8 GB Memory 1867 MHz DDR3 and 2.9 GHz Processor Intel Core i5 machine. The result shows there is no significant accuracy improvement after 4 literals. Surprisingly, the algorithm runs very slow at 2 literals for the car dataset (where the achieved accuracy is the lowest for all tested clause lengths). The reason is in Step 4 i.e. the removal of covered positive examples (see Section 3). If we cannot find any consistent hypothesis when generalising a positive example, we add an exception and only remove one example. But when the algorithm can find consistent hypotheses, it removes more than one example which results in much fewer positive examples thus speeding up the search. This anomaly does not occur in the sushi dataset due to the larger number of attributes so that the possibility to find a consistent hypothesis for clause length 2 is higher.

Table 5: Performance on different clause length settings

Clause length	Car dataset		Sushi dataset	
	Accuracy	Running time	Accuracy	Running time
2 literals	0.7433 $\pm$0.15	17,824 ms	0.8135$\pm$0.13	12,189 ms
3 literals	0.8533 $\pm$ 0.14	11,338 ms	0.8117$\pm$0.13	19,040 ms
4 literals	0.8434 $\pm$0.12	13,050 ms	0.8266$\pm$0.13	36,922 ms
5 literals	0.8217 $\pm$0.15	14,223 ms	0.8150$\pm$ 0.14	60,943 ms

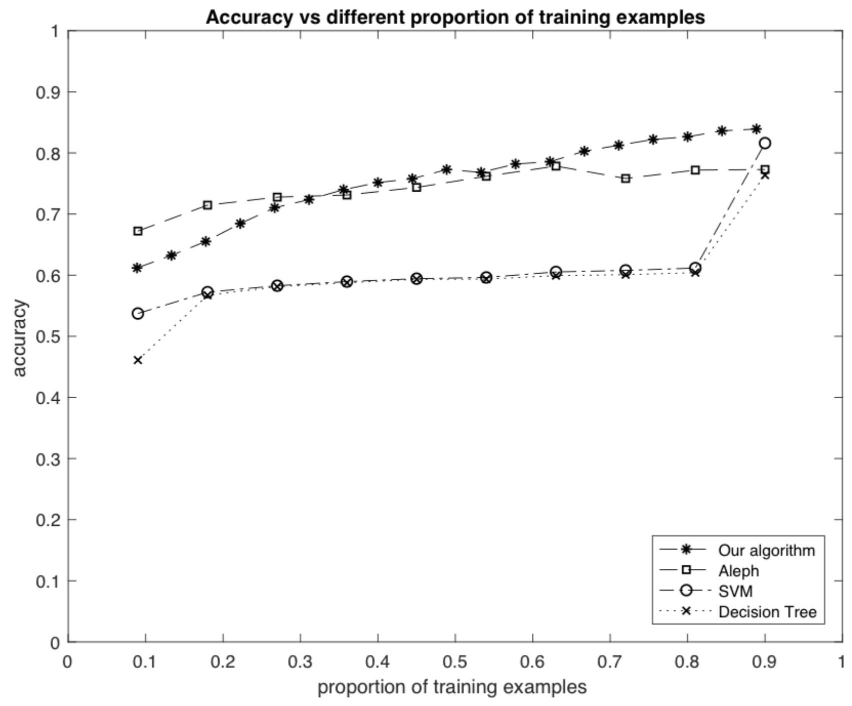


Figure 4: Accuracy by varying number of training examples

Sample solutions found.

As mentioned in [2], by using DL representation, which is variable-free syntax, our algorithm can produce a set of solutions which are more readable compared to the First Order Logic representation. An example of a consistent hypothesis found by our algorithm is shown below:

```
(Manual) betterthan (Sedan)
(Automatic and Hybrid) betterthan (MediumCar and Suv)
```

The above rules simply say that: “Manual car is better than sedan car OR automatic hybrid car is better than medium size SUV car.” In comparison, Aleph produces rules, such as:

```
betterthan(A,B) :- hasenginesize(B,large), hasenginesize(A,small).
betterthan(A,B) :- hasfuelcons(B,nonhybrid), hasbodytype(B,sedan).
```

The first Aleph’s rule states that: “car A is better than car B if car B has a large engine size and car A has a small engine size.” In the second rule, it says:“car A is better than car B if car B is a non hybrid car and sedan.” In other words, if the car has characteristics as a non-hybrid sedan car, it will not be better than any other car.

5 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 preference from pairwise comparisons. In fact, the proposed algorithm has proven statistically significantly better than all tested alternatives in all but one case. The exception in question is when compared to SVM on the car dataset, where our algorithm achieves a seemingly higher mean accuracy, but the result is not statistically significant (in other words, it is a draw).

We are currently working to address some of the limitations of the algorithm. In terms of accuracy, we show that our algorithm outperformed the other algorithms. However, to produce the final theory, our algorithm takes much longer than each of the three baseline algorithms. Currently, the only way to restrict the search complexity is by limiting its depth. We need to work on the improvement of this aspect without reducing the accuracy performance. The possible alternatives include the use of heuristic functions to predict the cost of expanding a node (i.e. informed search). We are also working on expanding the refinement operator by allowing the use of different operators (e.g. union or negation) with the aim of improving the accuracy.

References

[1] Abbasnejad, E., Sanner, S., 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)

[2] Baader, F., Horrocks, I. and Sattler, U.: Description Logics. In: Handbook of Knowledge Representation, pp. 135–179. Springer, Heidelberg (2009)

[3] Balakrishnan, S. and Chopra, S.: Two of a kind or the ratings game? Adaptive pairwise preferences and latent factor models. In: Proceedings of IEEE 10th International Conference on Data Mining (ICDM), pp. 725–730 (2010)

[4] Bradley, R. A., Terry, M. E.: The rank analysis of incomplete block designs. I. The method of paired comparisons. In: Biometrika, vol 39, no. 3-4, pp. 324–345 (1952)

[5] 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)

[6] 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)

[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] Hüllermeier, E., Fürnkranz, J., Cheng, W., and Brinker, K.:Label ranking by learning pairwise preferences. In: Artificial Intelligence, vol. 172, no. 16, pp. 1897–1916 (2008)

[9] 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)

[10] Jensen, B., Saez Gallego, J. and Larsen, J.: A predictive model of music preference using pairwise comparisons. In: Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1977–1980 (2012)

[11] 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)

[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] Law, M., Russo, A. and Broda, K.: Learning weak constraints in answer set programming. In: Theory and Practice of Logic Programming, vol. 15 no. 4-5, pp. 511–525 (2015)

[15] Lehmann, J.: DL-Learner: Learning Concepts in Description Logics. In: The Journal of Machine Learning Research, vol. 10, pp. 2639–2642. (2009)

[16] Linden, G. D. , Jacobi, J. A., and Benson, E. A.: Collaborative recommendations using item-to-item similarity mappings. In: US Patent 6,266,649 (2001)

[17] Muggleton, S.: Inverse Entailment and Progol. In: New Generation Computing, vol. 13 no. 3-4, pp. 245–286 (1995)

[18] 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)

[19] Qian, L., Gao, J. and Jagadish, H.: Learning user preferences by adaptive pairwise comparison. In: Proceedings of the VLDB Endowment, vol. 8 no. 11, pp. 1322–1333 (2015)

[20] Rokach, L. and Kisilevich, S.: Initial profile generation in recommender systems using pairwise comparison. In: IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, vol. 42 no. 6, pp. 1854–1859 (2012)

[21] Srinivasan, A.: The Aleph Manual. In: Technical Report. Computing Laboratory, Oxford University (2000), `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/`