Predicate Invention in Inductive Logic Programming

Duangtida Athakravi, Krysia Broda, and Alessandra Russo

Department of Computing, Imperial College London, U.K.
{da407,kb,a.russo}@doc.ic.ac.uk

Abstract

The ability to recognise new concepts and incorporate them into our knowledge is an essential part of learning. From new scientific concepts to the words that are used in everyday conversation, they all must have at some point in the past, been invented and their definition defined. In this position paper, we discuss how a general framework for predicate invention could be made, by reasoning about the problem at the meta-level using an appropriate notion of top theory in inductive logic programming.

1998 ACM Subject Classification I.2.6 Learning, D.1.6 Logic Programming

Keywords and phrases Predicate invention, Inductive logic programming, Machine learning

Digital Object Identifier 10.4230/OASIcs.ICCSW.2012.15

1 Predicate Invention

In Inductive Logic Programming (ILP) a hypothesis H (a set of rules) is learned from some background knowledge B and a set of observed positive and negative examples $E = E^+ \cup E^-$, using mode declarations as bias for the syntax of the rules. The learned hypothesis should be the most general one that will make the positive examples derivable once the hypothesis is added to the background knowledge ($B \cup H \models E^+$) and is consistent with the negative examples ($\forall e^- \in E^- : B \cup H \not\models e^-$).

Mode declarations contains the schema for the allowed literals in the rule, and can be of the form $modeh(s)$ or $modeb(s)$ for the head or body of the rule respectively. The schema s is a grounded literal with placemarkers of the form '$+type$', '$-type$', or '$\#type$', with $type$ corresponding to the type of the literal's argument. The symbols '$+$', '$-$', and '$\#$' indicates whether the argument should be a variable in the head of the rule or one from previous body literals in the rule (input variable), a new fresh variable (output variable), or a constant respectively. Thus, the mode declarations $modeh(fly(+bird))$ and $modeb(wings(+bird, \#property, -int))$ would allow a rule such as $fly(X) \leftarrow wings(X, has\_flight\_feathers, Y)$ to be constructed, where X is a bird and Y is an integer, with $has\_flight\_feathers$ being a property of the bird's wings.

Predicate Invention is when the hypothesis includes a predicate that was neither within the background knowledge nor the examples. There are two reasons why a new predicate may be invented:

1. *Reformulation*: To identify interesting concepts not directly related to the learning goal that could be used to restructure the program. For example, if the background knowledge contains the rules:
   $pigeon(X) \leftarrow beak(X), feathers(X), wings(X), fly(X)$.
   $penguin(X) \leftarrow beak(X), feathers(X), wings(X), \neg fly(X)$.
   These rules share many conditions which could be factored out by inventing a new predicate $bird/1$, with the definition of $bird(X) \leftarrow beak(X), feathers(X), wings(X)$.

2012 Imperial College Computing Student Workshop (ICCSW'12).
Editor: Andrew V. Jones; pp. 15–21

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The new predicate can then be used to replace all occurrences of the shared conditions within the background knowledge:
$pigeon(X) \leftarrow \underline{bird(X)}, fly(X).$
$penguin(X) \leftarrow \underline{bird(X)}, \neg fly(X).$
$\underline{bird(X)} \leftarrow beak(X), feathers(X), wings(X).$

2. *Bias Shift*: To specialise an overgeneral hypothesis and make it consistent with the examples. This is when the vocabulary available to the learner is not sufficient for constructing a consistent hypothesis. Therefore a new predicate is needed for specialising the overgeneral hypothesis such that it would no longer cover negative examples. Consider the following ILP problem:
$B = \{\ bird(alex).\ \ bird(bob).\ \}$ $E^+ = \{\ fly(alex).\ \}$ $E^- = \{\ fly(bob).\ \}$
$M = \{\ modeh(fly(+bird)).\ \}$
The current vocabulary is not strong enough to construct a consistent hypothesis. The mode declaration only allows the rule $fly(X)$, which is not sufficient for discriminating the negative example $fly(bob)$ from the positive one. Furthermore, no other conditions can be added to solve this problem given the current mode declaration. Thus, a new predicate p and the fact $p(alex)$ need to be added so that the consistent rule $fly(X) \leftarrow p(X)$ can be learned.

There are three main difficulties [13] that need to be considered when inventing new predicates:

1. *When to invent new predicate*
There needs to be a criteria for deciding when a new predicate is necessary, as we would not like to add useless predicates to the background knowledge that would only hinder the learner in future tasks.
2. *How to invent new predicate*
What structure should the new predicate have and how can its definition be found? Would a recursive call to the ILP algorithm in use be sufficient, or would a separate algorithm be required for learning the new predicate's definition?
3. *How to control the search space of the new predicate*
The search space for the new predicate could potentially be infinitely large, and the mode declarations for the original learning problem may no longer applies for learning the new predicate. There needs to be a way of limiting or directing the search space for the new predicate.

In this position paper, we discuss how a general framework for predicate invention, which is able to accommodate both cases of of theory reformulation and bias shift, could be developed. This takes advantage of recently proposed meta-level abductive approach to inductive logic programming, whereby inductive tasks are transformed into equivalent abductive task, reducing the computation to the search of equivalent abductive explanations. After a brief summary of the current state of art of the field, we show how a predicate invention task can be formulated as a meta-level search problem, that can compute new predicates for compacting the given background knowledge as well as specialising hypothesis.

2 Related Work

Past systems have concentrated on one of either reformulation or bias shift when inventing new predicates. DIALOGS [5], SIRIES [14] and CHAMP [12] all invents new predicate for bias shift, while INDEX [4] invents new predicates for reformulating the theory. Although in Cigol [11] the main goal is to find new rules for reformulation, its new predicates are only

invented when negative examples are confirmed by the oracle, the same situation as bias shift. There are also methods such as Statistical Predicate Invention (SPI) [9] and matrix sorting [8] that do not follow the framework of ILP, but are also able to introduce new predicates into existing theories. These methods find new predicates by identifying trends within the theory then grouping objects together, and the new predicates are introduced to restructure the theory according to those trends and groupings.

While each system's exact method is different from one another, they use the same strategy for inventing a new predicate. They start by identifying the appropriate structure of the new predicate then use it as an input to a learning algorithm, often the main inductive learning algorithm, to learn the definition of the new predicate. For instance, CHAMP invents its predicate by finding the smallest set of arguments that would completely discriminate the positive and negative examples, while Cigol finds the predicate's arguments by identifying non-unifiable arguments when generalising its examples. Having identified the structure, both systems then use it in a recursive call to their main learning algorithm.

The way each system controls the search space for the new predicate is more varied. Both DIALOGS and SIRIES prioritise some hypothesis structures over others, giving lower priorities to those that are more complicated or with new predicates. CHAMP and Cigol prefer a hypothesis that will achieve the most compression of their theory. While INDEX, SPI and matrix sorting uses some scoring mechanism for the most appropriate representation of their data.

3 Predicate invention at meta-level

Our framework is general enough for both bias shift and reformulation. Abstracting the problem to the meta-level would be suitable for inventing new predicates, as past methods have shown that meta-knowledge is often needed for deciding the new predicate's structure regardless of its purpose. ILP systems already have some means for reasoning about the language of its hypotheses by using the *top theory*, a theory on the encoding of the hypothesis as allowed by the mode declaration. Furthermore, the top theory should be flexible enough for importing any heuristics for inventing predicates.

Firstly, we briefly describe an existing ILP system called ASPAL that performs standard ILP tasks using meta-level abduction. We use this system to automate our framework as ASP uses sophisticated mechanisms for optimising search.

3.1 ASPAL

ASPAL (ASP Abductive Learning) is the Answer Set Programming (ASP), declarative programming based on stable model semantics [7], implementation of TAL. TAL (Top-directed Abductive Learning) [1, 2] is a top-down nonmonotonic ILP algorithm implemented in Prolog, using abductive learning to find the correct hypothesis. It solves an inductive problem by converting it into an abductive one. The hypothesis of the problem is found using a top theory, the theory concerning the construction of the hypothesis according to the mode declarations of the inductive problem. By reasoning with the top theory, TAL and ASPAL abstract the problem to the meta-level of the hypothesis' encoding.

$M = \{\ modeh(fly(+bird)).$
$\quad modeb(pigeon(+bird)).\}$

For example, using the mode declarations above, the corresponding top theory in ASPAL is as follows:

$$T_{ASPAL} = \{\ fly(X) \leftarrow bird(X), \$rule(r((fly, c, v))).$$
$$fly(X) \leftarrow bird(X), pigeon(X),$$
$$\$rule(r((fly, c, v), (pigeon, c, v(1)))).\ \}$$

The top theory matches the head of its clauses to examples of the ILP problem, testing conditions for those clauses and abducing the rule encoding $\$rule/1$ should no negative example satisfies the clause. Each tuple in $\$rule/1$ corresponds to a mode declaration (using fly or $pigeon$ for identification). The constants c and v are used to represents empty lists `[]` of constants and variables, while $v(1)$ represents the list `[1]` with a single index linking to the first variable in the rule. The constant list is used when the literal has constants in its arguments, while the variable list links variables in the rule together using their indexes. Using the top theory above with the background and examples:

$$B = \{\ bird(alex). \qquad E^+ = \{\ fly(alex).\ \}$$
$$bird(bob). \qquad E^- = \{\ fly(bob).\ \}$$
$$pigeon(alex).\ \}$$

The examples are used to construct an integrity constraint in the ASP program, such that all answer sets in the solution must include all positive examples and none of the negative ones. The first clause in T_{ASPAL} prevents the rule $fly(X)$ from being added to the hypothesis as the negative example $fly(bob)$ would also satisfy the clause. Thus $\$rule(r((fly, c, v)))$, the encoding for $fly(X)$, would not be included in the answer set. However, as the condition $pigeon(bob)$ is not satisfiable by the rule $fly(X) \leftarrow pigeon(X)$, its representation $\$rule(r((fly, c, v), (pigeon, c, v(1))))$ can be abduced.

ASPAL was used rather than TAL as the ASP solver is extremely efficient when solving a grounded program, ASPAL's current implementation avoids costly computation of the ASP grounder by using a preprocessor for constructing an ASP program with all possible grounded hypotheses. These advantages allow for many number of mode declarations to be used without high increase in computational time.

3.2 Predicate invention using meta-level abduction

In [10], a simple method for introducing new predicates through the mode declarations was shown by using *placeholders*. Placeholders are mode declarations of new predicates that were neither within the problem's example, its background knowledge, nor its mode declarations. For example, suppose we have the following problem:

$$B = \{\ alpha(a). \qquad E^+ = \{\ q(a, d), q(a, c).\ \}$$
$$alpha(b). \qquad E^- = \{\ q(c, d).\ \}$$
$$alpha(c). \qquad M = \{\ modeh(q(+alpha, +alpha)).\ \}$$
$$alpha(d).\ \}$$

To solve this using placeholders, we can add new mode declarations $modeh(new(\#alpha, \#alpha))$ and $modeb(new(+alpha, +alpha)$ to the problem, such that rules and facts such as $p(X, Y) \leftarrow new(X, X)$ and $new(a, a)$ can be learned. Running the problem in ASPAL takes only 0.01 seconds to solve. Should we want to add other seventeen alternative mode declarations for $modeb(new(+alpha, +alpha)$ (with negation and different combinations of constants, input and output variables), and limiting to maximum of one body literal per rule and two rules per hypothesis, ASPAL will solve the problem in 0.078 seconds and output 20 hypotheses. Many of the hypotheses subsumes each other, for instance:

$$H_1 = \{p(X, Y) \leftarrow new(X, Z). \quad new(a, a).\}$$
$$H_2 = \{p(X, Y) \leftarrow new(X, X). \quad new(a, a).\}$$
$$H_3 = \{p(X, Y) \leftarrow new(a, a). \quad new(a, a).\}$$

From the hypotheses above, simply selecting the shortest one is not sufficient as they all have the same length. Instead, we could lower the number of hypotheses by discarding

hypotheses that are subsumed by others. In the case above, H_2 and H_3 can be discarded as they are both subsumed by H_1, making H_1 the most general hypotheses out of the three hypotheses.

For reformulating a theory, [3] has shown how an ILP system can be used to revise theories by transforming the revisable section of the background knowledge and learning revision operators. Each revisable rule $r_i \leftarrow c_{i,1}, \ldots, c_{i,n}$ can be transformed to:

$r_i \leftarrow try(i, 1, c_{i,1}), \ldots, try(i, n, c_{i,n}), ext(i, r_i).$
$try(i, 1, c_{i,1}) \leftarrow c_{i,1}, use(i, 1).$
$try(i, 1, c_{i,2}) \leftarrow not\ use(i, 2).$
$\ldots$
$use(i, j) \leftarrow not\ del(i, j).$

Each $try/3$ clause is used to test if a condition j in a rule i is not used in the rule. Due to the definition of $use/2$, the condition $c_{i,j}$ is removed from the theory when the corresponding $del(i, j)$ is learned. The literal $ext/2$ is used for learning additional conditions to be added to the body in the rule. For instance, $ext(i, r_i) \leftarrow p(a)$ indicates that the rule r_i should be extended with the literal $p(a)$.

We have applied this method to reformulate the clauses:

$grandfather(X, Y) \leftarrow male(X), parent(Z, Y), parent(X, Z).$
$grandmother(X, Y) \leftarrow female(X), parent(Z, Y), parent(X, Z).$

As well as the mode declarations needed for revising the clauses, additional mode declarations were included such that the rule $new(X, Y) \leftarrow parent(Z, Y), parent(X, Z)$ can be learnt. This is so the learner can find a solution that would reformulate the rules to:

$grandfather(X, Y) \leftarrow male(X), \underline{new(X, Y)}.$
$grandmother(X, Y) \leftarrow female(X), \underline{new(X, Y)}.$
$\underline{new(X, Y)} \leftarrow parent(Z, Y), parent(X, Z).$

While we were able to acquire the above solution, the learner outputs many more solutions, with most not decreasing the size of the theory. This is because the search is only guided by the examples given, not by how much each hypothesis could compact the theory. Thus, to the learner, the following solution would be as good as the previous one:

$grandfather(X, Y) \leftarrow male(X), parent(Z, Y), \underline{new(X, Z)}.$
$grandmother(X, Y) \leftarrow female(X), parent(Z, Y), \underline{new(X, Z)}.$
$\underline{new(X, Y)} \leftarrow parent(X, Y).$

While comparing the literal count could help us identify the best revision, another simple solution is by using the optimisation feature of iClingo [6], the ASP solver used by ASPAL. As $del/2$ instances indicated removal of clauses, by asking the solver the find the maximum number of $del/2$ instances possible, we can then use it to find only solutions that will most reduce the size of the background knowledge. Similarly, finding the minimum number of new clauses added to the background knowledge can also help to find the solutions that will least increase the size of the background knowledge.

In conclusion, as well as the general ILP task, our framework is also capable performing the following tasks with predicate invention:

1. A bias shift task $\langle E, B, M \rangle$, where E is a set of examples, B is the background knowledge, and M is the set of mode declarations. A set of rules, a hypothesis H_B, is a solution to the task $\langle E, B, M \rangle$ if H_B is compatible with M extended by a new predicate p, $H_B \cup B$ is consistent with E, and H_B the predicate p such that $p \notin B$, $p \notin E$, and $p \notin M$.
2. A reformulation task $\langle B_N, B_R, M \rangle$, where B_N is the non-revisable background knowledge, B_R is the revisable background knowledge, and M is the set of mode declarations. A solution to the task $\langle B_N, B_R, M \rangle$ is tuple $H_R = \langle H_N, H_O \rangle$, where H_N (possibly empty)

is a set of new rules, and H_O is a sequence of revision operations, these include adding conditions to existing rules and deleting conditions or rules in B_R. H_R is a valid solution to $\langle B_N, B_R, M\rangle$ if H_N is compatible with M extended by a new predicate p, and for $o_1, \ldots, o_n \in H_O$: $B_R \otimes \{o_1, \ldots, o_n\} \cup B_N \cup H_N$ to have the same answer sets with $B_N \cup B_R$ for their shared predicates, and H_R may contain the predicate p such that $p \notin B_N \cup B_R$, and $p \notin M$.

3. By combining the previous two tasks, predicates can also be invented for correcting erroneous knowledge. Expanding the theory revision task to give a task $\langle E, B_N, B_R, M\rangle$, where E is a set of examples, B_N is the non-revisable background knowledge, B_R is the revisable background knowledge, M is the set of mode declarations, and $B_N \cup B_R$ is inconsistent with E. A solution to the task $\langle E, B_N, B_R, M\rangle$ is a tuple $H_T = \langle H_N, H_O\rangle$, where H_N is a set of new rules and H_O is a sequence of revision operations. H_T is a valid solution to $\langle E, B_N, B_R, M\rangle$ if H_N is compatible with M extended by a new predicate p, and for $o_1, \ldots, o_n \in H_O$: $B_R \otimes \{o_1, \ldots, o_n\} \cup B_N \cup H_N$ is consistent with E, and H_T contains a new predicate p such that $p \notin B_N \cup B_R$, $p \notin E$, and $p \notin M$.

4 Future Work

The objective of our research is a general ILP framework with capacity for an efficient way of inventing new predicates, able to invent new predicates for both reformulation and bias shift.

The simple approach of generating all placeholders seems to be able to handle all issues we outlined in Section 1: (i) placeholders can be added to the ASP program when the original learning problem fails to produce a hypothesis, (ii) use all possible placeholders, starting from one argument and increasing the number of its arguments until solutions are found, and rely on the learning algorithm of ASPAL to compute the hypotheses, (iii) search can be controlled by limiting the number of rules within the hypothesis and increasing it until the minimum number of rules is found. However, we still need to test it on larger problems to ensure that high number of mode declarations does not lead to a great increase in computational time or the number of solutions. If so, then we will need to find a way to limit the number of mode declarations that are considered for each time new predicates are needed.

For theory reformulation, we still need to determine when should the theory be reformulated, and how to control the search. A way to control the search is by associating each revision operator with a measure of its effect on the program size, so that the learner can use it to discard hypothesis that do not reduce the theory's size.

ASPAL already has some preliminary work done on hypothesis refinement, which could be used after the learner has gone through the search space from the given mode declarations. We plan to complete this hypothesis refinement framework of ASPAL, as this will help with determining when to invent new predicates for bias shift. It will allow us to invent new predicates only when demanded, similar to systems such as CHAMP and SIRIES.

Acknowledgements We would like to thank Domenico Corapi for his help and discussion on TAL and ASPAL.

References

1 Domenico Corapi. *Nonmonotonic Inductive Logic Programming as Abductive Search.* PhD thesis, Imperial College London, 2011.

2 Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive logic programming as abductive search. In Manuel Hermenegildo and Torsten Schaub, editors, *Technical Commu-*

nications of the 26th International Conference on Logic Programming, volume 2010, pages 54–63, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

3 Domenico Corapi, Alessandra Russo, Marina De Vos, Julian A. Padget, and Ken Satoh. Normative design using inductive learning. *TPLP*, 11(4-5):783–799, 2011.

4 Peter A Flach. Predicate Invention in Inductive Data Engineering. In P Brazdil, editor, *Machine Learning ECML93 European Conference on Machine Learning Proceedings*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 83–94. Springer-Verlag, 1993.

5 Pierre Flener. Inductive logic program synthesis with DIALOGS. In Stephen Muggleton, editor, *Inductive Logic Programming*, volume 1314 of *Lecture Notes in Computer Science*, pages 175–198. Springer Berlin / Heidelberg, 1997.

6 M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. Engineering an incremental ASP solver. In M. Garcia de la Banda and E. Pontelli, editors, *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*, volume 5366 of *Lecture Notes in Computer Science*, pages 190–205. Springer-Verlag, 2008.

7 Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. pages 1070–1080. MIT Press, 1988.

8 Charles Kemp, Joshua B Tenenbaum, Sourabh Niyogi, and Thomas L Griffiths. A probabilistic model of theory formation. *Cognition*, 114(2):165–196, 2010.

9 Stanley Kok and Pedro Domingos. Statistical predicate invention. In *Proceedings of the 24th International Conference on Machine Learning (2007)*, volume pages, pages 433–440. ACM Press, 2007.

10 Gregor Leban, Jure Zabkar, and Ivan Bratko. An Experiment in Robot Discovery with ILP. In *ILP 08 Proceedings of the 18th international conference on Inductive Logic Programming*, pages 77–90, 2008.

11 Stephen Muggleton and Wray L Buntine. Machine Invention of First Order Predicates by Inverting Resolution. In *Machine Learning*, pages 339–352, 1988.

12 B K M N M Shimura. Discrimination-Based Constructive Induction of Logic Programs. In *AAAI92 proceedings Tenth National Conference on Artificial Intelligence July 1216 1992*, page 44. Aaai Pr, 1992.

13 Irene Stahl. Predicate Invention in Inductive Logic Programming. In Luc De Raedt, editor, *Advances in Inductive Logic Programming*, pages 34–47. IOS Press, 1996.

14 Ruediger Wirth and Paul O'Rorke. Constraints on predicate invention. In Stephen H Muggleton, editor, *Proceedings of the Eighth International Workshop on Machine Learning*, pages 457–461. Morgan Kaufmann, 1991.