Reduction of ILP Search Space with Bottom-Up Propositionalisation

Hadeel Al-Negheimish[1,2] and Alessandra Russo[1]

[1] Imperial College London, United Kingdom
[2] King Saud University, Saudi Arabia

Abstract. This paper introduces a method for algorithmic reduction of the search space of an ILP task, omitting the need for explicit language bias. It relies on bottom-up propositionalisation of examples and background knowledge. A proof of concept has been developed for observational learning of stratified normal logic programs.

1 Introduction

Machine learning can be defined as automatic program improvement in certain tasks through experience [8]. One such task is defining a concept based on examples labelled according to their membership. When examples along with background knowledge are expressed as logic programs, its related task is seen as an inductive logic programming task. Learning with ILP leverages existing domain knowledge, and produces hypotheses that are understandable and expressive due their logical form.

Most ILP tasks take as input a tuple: positive examples, negative examples, background knowledge and language bias. Language bias serves to restrict the search space, as it defines which predicates can appear in the hypotheses, in addition to any constraints that may apply. Specification of language bias arguably requires prior knowledge of the form the hypothesis should be in, which may not always be available. On the other hand, the size of an unconstrained search space makes an unguided traversal intractable. One way to reduce the search space is by propositionalisation. Propositionalisation techniques aim to transform the task from a relational to an attribute-value form, which makes it easier to learn, but possibly loses some information [2].

This work aims at restricting the search by finding clues of relevance that are inherent in the examples and background knowledge, utilising a combination of incremental propositionalisation and extended set operations, omitting the need for language bias. Our contribution is two-fold: a definition of bottom-up propositionalisation for first-order logic programs, and an algorithmic mechanism for generating ILP-hypotheses that uses these propositional features. A brief background is presented in the next section. We describe the workings of our approach in section 3, along with an illustrative example. A discussion of related work is presented in section 4.

2 Background

Before delving into learning with bottom-up propositionalisation, it is useful to first formally define an ILP task and present an overview of how ILP learners typically work.

Definition 1. *Let $\mathcal{L}$ be a language and let T be the learning task of our approach, where $T{=}\langle M, B, E^+, E^-\rangle$, defined in $\mathcal{L}$, where:*

M *Mode declarations, where $\mathcal{L}_M$ is the language restricted by M.*
B *Background knowledge, a logic program.*
E$^+$ *Positive examples, a set of ground atoms of the same predicate.*
E$^-$ *Negative examples, also a set of ground atoms of the same predicate as E^+*

A solution H is a (set of) clause(s) such that:

1. $H \in \mathcal{L}_M$
2. $\forall e^+ \in E^+ : B \cup H \models e^+$
3. $\forall e^- \in E^- : B \cup H \not\models e^-$

The hypotheses space of an ILP problem has a lattice structure, with partial ordering based on hypothesis generality and subsumption. (We say that $H \succ H' \Leftrightarrow H \models H'$). Existing systems vary on how they traverse this search space; with a top-down approach, overly general hypothesis are refined until they are consistent with negative examples, and branches that do not cover some of the positives are pruned. With a bottom-up, only clauses that do not cover any negatives are generalised, until a solution to the task is found.

We take a different approach towards finding a solution, which does not traverse the hypotheses space one-step at a time. Propositional features are constructed bottom-up, and then a subset of these is selected -depending on their coverage of examples- to find a solution. Since all examples are considered at each step, it allows for learning normal logic programs.

3 Learning with Bottom-Up Propositionalisation

Given a set of examples, it is intuitive to try to define them by discerning which characteristics they have in common, which separate them from the instances that are known not to belong to that concept. How can we identify these characteristics in ILP tasks? This is the basis upon which this work is held.

Propositionalisation is done *bottom-up* in the sense that instead of starting out with templates of the features and checking which of them are satisfied, we generalise the existing atoms. The atoms that are generalised are the ones in the model of the background program and are relevant to some given example, with respect to how that example maps to the target hypothesis head. We present a few definitions before explaining the approach step by step.

The entire details of the work and reasoning behind some design decisions, along with proofs on correctness and minimality are in [1].

3.1 Definitions

We extend set operations to account for generality ordering between atoms, by checking $\theta - subsumption$. Below are the definitions for these extended operations:

Definition 2 ($\cap^*$). *Let A,B be sets of (possibly ground) atoms and a be one such atom,*
$a \in A \cap^* B \quad iff \quad either$

$$\bullet\ a \in A \wedge (a \in B \vee \exists a' \in B \quad s.t. \quad a \models a')$$
$$\bullet\ a \in B \wedge (a \in A \vee \exists a' \in A \quad s.t. \quad a \models a')$$

If we assume that sets A and B do not contain predicates in multiple levels of generality in the same set, keeping only the most specific form, then A $\cap^*$ B will contain only the predicates common to both sets, in the most general form.

Definition 3 ($\setminus^*$). *Let A,B be sets of (possibly ground) atoms and a be one such atom,* $a \in A \setminus^* B \quad iff \quad a \in A \wedge (a \notin B \wedge \nexists a' \in B \quad s.t. \quad a \models a')$

Definition 4 (Target atom). *Let* $T{=}\langle B, E^+, E^- \rangle$*, a target atom is an unground atom such that target =* $lgg(e_1, \ldots, e_n)$ $\forall e \in E^+$.

Example 1. Let $E^+ = \{p(a), p(b), p(c)..\}$ then target= p(X)

The target atom represents the head of the main clause in the hypothesis; using least-general-generalisation helps identify constants, and eliminates negative examples that are not subsumed by the target atom.

Definition 5 ($\mathcal{M}(e)$). *Let e be an example,* $\mathcal{M}(e)$ *is a mapping of terms in e to head (universally quantified) variables, more formally:*
$\mathcal{M}(e) = \{t_i \mapsto V_i \quad where \quad V_i \mapsto t_i \in \theta \wedge target\theta = e\}$

$\mathcal{M}(e)$ can be seen as the inverse of the substitution θ in $target\theta = e$. Instead of matching each variable with a term, we match each term with the variable's symbol, to be used in generalising atoms later.

Example 2. Let target be p(X, Y), e_1 = p(a,b), and e_2 = p(c,c), then $\mathcal{M}(e_1) = \{a \mapsto X, b \mapsto Y\}$ and $\mathcal{M}(e_2) = \{c \mapsto \{X, Y\}\}$

Definition 6 (Links(term)). *Let t be a constant term, B some logic program, links(t) is a set of all atoms in the model of B that contain the term t, more formally:*
$links(t) = \{p(t_1, \ldots, t_n) | p(t_1, \ldots, t_n) \in model(B) \wedge t = t_i \text{ for some } i \ \ 1 \leq i \leq n\}$

For each example, we construct sets of relevant atoms entailed by the background program. To know which atoms are relevant to each example, we link the atoms based on the terms that occur in both of them.

Example 3. Let $B = \{q(a).s(b).r(a,b).s(a).\}$ then $links(a) = \{q(a), r(a,b), s(a)\}$

Definition 7 (Generalise(a, $\mathcal{M}(e)$)). *Let a be a ground atom, $\mathcal{M}(e)$ is a mapping as defined in definition 5, Generalise(a, $\mathcal{M}(e)$) is a set of unground atoms where terms are replaced by their mapping in $\mathcal{M}(e)$, and terms not in $\mathcal{M}(e)$ are replaced with a new skolem variable.*

Generalisation of an atom is an ungrounding process that is a direct substitution of the ground term to the variable symbol it maps to in $\mathcal{M}(e)$. The same atom can be generalised to different features depending on the mapping it uses, which in turn depends on the example used.

3.2 Algorithm and Implementation

Our work takes on a more algorithmic approach to solving an ILP problem, the basic steps of Bottom-up Propositionalisation ILP are outlined below:

Algorithm 1: BPILP: Bottom-up Propositionalisation ILP

Input: **B**: Background Program, $\mathbf{E}^+$: Positive Examples and $\mathbf{E}^-$: Negative Examples
Output: H_{Min}: Minimal hypothesis

1 $M \leftarrow model(B)$
2 $Target \leftarrow lgg(e_1, e_2, \ldots e_n) \quad \forall e_i \in E^+$
3 **foreach** *e in* E^+ *and* E^- **do**
4 $\quad \mathcal{M}(e) \leftarrow$ term to variable mapping wrt $Target$
5 $\quad Links(e) \leftarrow$ all atoms in M with at least one term in e
6 $\quad Feats(e) \leftarrow$ Generalise(a, $\mathcal{M}(e)$) $\forall a \in Links(e)$
7 $body^+ \leftarrow \bigcap^*(Feats(e_i^+)) \quad \forall e_i^+ \in E^+$
8 $body^- \leftarrow \{f | f \in \bigcup(Feats(e_j^-)) \setminus^* \bigcup(Feats(e_i^+))$
$\qquad \wedge \nexists V \in args(f) \quad s.t. \quad is\_skolem(V) \quad \forall e_i^+ \in E^+ \quad \forall e_j^- \in E^-\}$
9 $H \leftarrow \{Target\text{:-}\ b_1^+, \ldots, b_n^+, \text{not } b_1^-, \ldots, \text{not } b_m^- . | b_i^+ \in body^+ \wedge b_j^- \in body^-\}$
10 **if** *H is a solution* **then**
11 $\quad H_{Min} \leftarrow reduce(H)$
12 **else**
13 $\quad$ expand skolem variables in $body^+$ (by repeating steps 3, 4 and 5) or add auxiliary predicate definitions until a solution is found
14 $\quad H_{Min} \leftarrow reduce(H)$
15 **return** H_{Min}

First, we get the unique model of the background program by feeding it to an ASP solver[3]. We then find the target atom which will represent the head of main clause of the hypothesis, by finding the lgg of positive examples. Afterwards a set is constructed for each given example in the ILP task, containing the links for all terms in that example. These sets are then generalised relative to the

[3] We assume B is a stratified normal logic program, with no occurrence of the target predicate

example and the target hypothesis head (using the mapping as defined in def 5), only the most specific generalisations are kept as features.

The importance of bottom up propositionalisation lies in the reduction of propositional features to work with; if we have t variables in the clause head, an n-ary predicate can be generalised in $(t+1)^n$ ways (since each argument can take any one of the head variables or a new skolem). By simply generalising our links, we obviate the need to construct all of these feature templates and check for their satisfaction. BPILP produces a way that is able to compare the generalisations between the example feature sets in order to easily find the features that discriminate positive from negative examples.

We test the resulting hypothesis, if it satisfies the problem we reduce it by removing redundant literals based on their coverage. Otherwise, we repeat the linking and generalising step for the features in $body^+$ which contain skolem variables, such that propositionalisation is incremental. This is currently constrained by setting a finite length for the depth of skolem variable expansion.

If the previous step does not result in satisfying features, the algorithm adds a new predicate, not in $\mathcal{L}$. This invented predicate can be defined by multiple clauses, each containing complementary features, or by a negative feature with a skolem variable.

Example 4. A demonstration of BPILP on a full task, where:

$$T = \langle B = \{q(X) : -w(X), not\ t(X).\quad t(a).\quad t(b).\qquad t(c).\quad w(c).\quad w(d).\},$$
$$E^+ = \{p(a).\quad p(b).\},\qquad E^- = \{p(c).\quad p(d).\}\rangle$$

1. Get model	$M = \{q(d).\quad t(a).\quad t(b).\quad t(c).\quad w(c).\quad w(d).\}$			
2. Find target	$Target = lgg(p(a), p(b)) = p(X)$			
	e_1	e_2	e_3	e_4
3. Get Links	$\{t(a).\}$	$\{t(b).\}$	$\{t(c).\ w(c).\}$	$\{q(d).\ w(d).\}$
4. $\mathcal{M}(e)$	$\{a \mapsto X\}$	$\{b \mapsto X\}$	$\{c \mapsto X\}$	$\{d \mapsto X\}$
5. Generalise Links	$\{t(X).\}$	$\{t(X).\}$	$\{t(X).\ w(X).\}$	$\{q(X).\ w(X).\}$
6. Saturated Hypothesis	$p(X) : -t(X), \text{not } w(X), \text{not } q(X).$			
7. Reduce Hypothesis	$p(X) : -\text{not } w(X).$			

4 Related Work

We introduced an algorithmic approach to tackle ILP tasks without the need for explicit language bias to restrict the search space, by powering bottom-up propositionalisation.

There is some work into trying to learn the language bias, McCreath and Sharma[7] extract meta knowledge that would be fed into an existing ILP system to restrict its search. It works on extensional examples and background

knowledge, finding type assignments, functional constraints and symmetry. It is limited due to its assumption of a complete list of positive examples, with a closed world assumption, and its restriction to the vocabulary of the language, no predicate invention is made.

Some ILP learning frameworks do not require language bias in their task definition, such as FOIL[10] and Meta-Interpretive Learning [9]. FOIL is a key framework to compare with, as it too uses an algorithmic approach to generate hypotheses, utilising no notion of proof. It builds one clause at a time, exploiting an information-based heuristic similar to ID3, to guide its search for simple, general clauses. FOIL may sometimes exhibit a short-sightedness, when all possibilities for the next literal to add have the same heuristic value. Our approach is safe from this limitation, since adding a literal with a skolem variable only occurs after that skolem has been expanded. Additionally, because of the ASP solver preprocessing step in our approach, we are not constrained to learning programs with purely extensional definitions as FOIL is.

The latter framework, MIL, boasts suitability for recursive definitions and predicate invention. It works by augmenting the background program with a meta-interpreter, which abduces higher order definite clauses, known as meta rules. However, it requires a total ordering over the predicate and object symbol sets to guarantee termination. Moreover, the current implementation constrains the language to a subset of H_2^2, which may compromise intuitive formulation of the problem. It does not learn normal logic hypotheses.

Propositionalisation is done in our work in a bottom-up manner to make it more tractable, hence not all propositional templates (top-down) are considered, as in the widely-known LINUS [6], which does not allow for skolem variables. SINUS, a later generation, allows skolem variables and is able to find propositionalisation to predicates that bind to more than one atom (multi-instance problems) only if the data is individual centred [11], which is too heavy a constraint.

Franca et. al.[3] use bottom clause propositionalisation for fast learning using neural networks. Like our work, propositionalisation is central to guiding the search for a valid hypothesis. However, BCP is dependent on mode declarations to construct bottom clauses, whilst our main objective is to obviate the need for explicit language bias.

5 Conclusion

We have introduced a novel approach to solving observational ILP tasks in which language bias is omitted in favour of algorithmic bias, by utilising incremental, bottom-up propositionalisation. It supports predicate invention to facilitate learning in some cases. It is able to learn normal logic program hypotheses for problems with stratified background programs, and has proved to work well on common problems, such as the Eastbound Trains [5] and Kinship [4] datasets.

6 Acknowledgements

This work was funded by a postgraduate scholarship awarded by King Saud University.

References

1. Al-Negheimish, H.: Towards an Inductive Logic Programming Approach with Hidden Bias. Master's thesis, Imperial College London (2015), unpublished
2. Deroski, S., Lavra, N. (eds.): Relational data mining. Springer, Berlin ; New York (2001)
3. França, M.V., Zaverucha, G., D'avila Garcez, A.S.: Fast relational learning using bottom clause propositionalization with artificial neural networks. Mach. Learn. 94(1), 81–104 (Jan 2014)
4. Hinton, G.E.: Learning distributed representations of concepts. In: Proceedings of the eighth annual conference of the cognitive science society. vol. 1, p. 12. Amherst, MA (1986)
5. Larson, J., Michalski, R.S.: Inductive inference of vl decision rules. ACM SIGART Bulletin (63), 38–44 (1977)
6. Lavrač, N., Džeroski, S., Grobelnik, M.: Learning nonrecursive definitions of relations with linus. In: Proceedings of the European Working Session on Learning on Machine Learning. pp. 265–281. EWSL-91, Springer-Verlag New York, Inc., New York, NY, USA (1991)
7. McCreath, E., Sharma, A.: Extraction of meta-knowledge to restrict the hypothesis space for ILP systems. In: AI-CONFERENCE-. pp. 75–82. Citeseer (1995)
8. Mitchell, T.M.: Machine Learning. McGraw-Hill Education, New York, 1 edn. (Mar 1997)
9. Muggleton, S.H., Lin, D., Tamaddoni-Nezhad, A.: Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. Mach Learn pp. 1–25 (Mar 2015)
10. Quinlan, J.R.: Learning logical definitions from relations. Machine learning 5(3), 239–266 (1990)
11. Raedt, L.D. (ed.): Logical and Relational Learning. Cognitive Technologies, Springer Berlin Heidelberg, Berlin, Heidelberg (2008)