Repeat Learning Using Predicate Invention

Khalid Khan[1,2], Stephen Muggleton[1], and Rupert Parson[2]

[1] Department of Computer Science, University of York
{kmk,stephen}@cs.york.ac.uk
[2] Oxford University Computing Laboratory
rupert.parson@comlab.ox.ac.uk

Abstract. Most of machine learning is concerned with learning a single concept from a sequence of examples. In *repeat learning* the teacher chooses a series of related concepts randomly and independently from a distribution $\mathcal{D}$. A finite sequence of examples is provided for each concept in the series. The learner does not initially know $\mathcal{D}$, but progressively updates a posterior estimation of $\mathcal{D}$ as the series progresses. This paper considers *predicate invention* within Inductive Logic Programming as a mechanism for updating the learner's estimation of $\mathcal{D}$. A new predicate invention mechanism implemented in Progol4.4 is used in repeat learning experiments within a chess domain. The results indicate that significant performance increases can be achieved. The paper develops a Bayesian framework and demonstrates initial theoretical results for repeat learning.

1 Introduction

Predicate Invention (PI) has been investigated for some time within Inductive Logic Programming (ILP) [7, 11, 3, 9, 4, 10]. The literature describes various mechanisms for the introduction of new predicates into the learner's vocabulary, as well as conditions under which this is necessary. However, surprisingly little, if any, experimental evidence exists to indicate that learners which employ PI perform better than those which do not. On the face of it, there are good reasons to believe that since increasing the learner's vocabulary expands the hypothesis space, PI could degrade both the learner's predictive accuracy and learning speed. Indeed this is consistent with what happens in the experiments described in Sect. 3.

However, the situation is different in the case in which the learner is allowed to repeatedly employ PI to discover background definitions for a series of related concepts. In this case predicates which successfully help describe one concept may decrease the sample requirements for subsequent concepts. This is also supported by experimental evidence in Sect. 3. In this paper we suggest a theoretical framework for *Repeat Learning* (RL) over a series of inter-related concepts. This framework is related to existing theoretical results of Baxter [1].

The paper is organised as follows. Section 2 introduces the experimental domain involving moves of the various chess pieces on the open board. The experiments described in Sect. 3 indicate that a) PI *does not* reduce sample requirements for learning a single concept and b) does significantly reduce sample requirements when successful invented predicates are provided as background knowledge for subsequent concepts. Section 4 describes how constraint solving in Progol4.4 was used to carry out PI in the experiments. A framework for RL is described in Sect. 5. Future directions for the research are discussed in Sect. 6. Pointers to an FTP site containing the experimental materials used in this paper are given in Sect. 7.

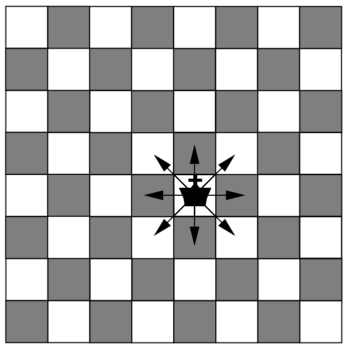
2 Examples

Consider the problem of repeat learning the definitions of legal moves on the open board for chess pieces. A concept will be associated with each piece to represent its legal moves. For example Fig. 1. shows the legal moves which can be made by a King and a Knight. Note the symmetries common to the legal moves of the pieces. In the experiment (Sect. 3) the initial background knowledge does not take account of symmetries. However once a predicate related to symmetry for one piece has been learnt it reduces the sample complexity of the learning in the second.

If we label the squares on a chess board according to *file* (column letter a-h) and *rank* (row number 1-8), we can then define two predicates `fdiff/2` and `rdiff/2` which compute the difference in file and rank for a move. All the legal chess piece moves can be defined entirely using these two predicates. However the size of the resulting theories is large. Both the King and Knight predicates would need 8 clauses to be defined in this way for each of the 8 possible directions (see Table 1, Fig. 1). We therefore attempt to introduce symmetry predicates into our definitions. This is done via PI and is used to demonstrate the general principle of RL - that a set of related concepts can better be learnt by also learning a common bias in which to represent those concepts. In the chess setting we might start off learning one piece, then use bias shift operations such as PI in order to help in the learning of subsequent pieces.

Table 1. Move definitions - with and without Symmetry Predicates

	King			**Knight**		
	No. Clauses	length	Total	No. Clauses	length	Total
Without Symmetry Predicates	8	3	24	8	3	24
With Symmetry Predicates	2	5,2	7	4	5,5,2,2	14


a) The King

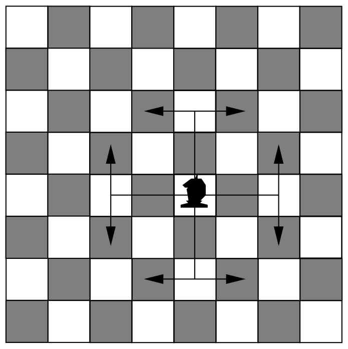

b) The Knight

Fig. 1. The legal moves of the pieces

3 Experiments

3.1 Training Data

The training data was generated by hand-coding a definition of the move for each piece, then using Progol's built in `sample/3` predicate to generate random examples. In this domain purely random training data will result in (typically 8 times) more negative examples than positive examples. Given this the positive and negative examples were generated separately and combined in a ratio of 1:1 in the training sets. For each piece the training sets contained 5,10,15,20,30,40 and 80 positive and negative examples. For each training set size, 20 random and independent example sets were generated. The results were then averaged over these 20 example sets to reduce variance.

3.2 Test Data

One of the advantages of using a well defined enclosed domain such as chess is the ability to obtain a perfect testing metric, based on the fact that there are only a finite number of possible moves. In particular there are 64 squares on a chess board, so this gives us a total of $64 \times 64 = 4096$ possible moves, most of which will clearly be negative examples for any given piece. In the interests of simplicity each piece was allowed to stay still in its definition of move. It is easy to compensate for this mis-specification simply by introducing an additional predicate. If we are trying to build up a move generator as in [2], we would want to state such a predicate only once, so it would be piece independent. This justifies our assumption of the existence of such a predicate.

3.3 Progol Settings

The following Progol settings were used when training on the data.

```
:-set(h,10000), set(c,5),set(i,2),set(nodes,200)?
```

There were two background predicates defined `rdiff` and `fdiff` for computing the file and rank difference of two squares. Finally Progol was told to Prune certiain clauses from its search. The first two prune statements did not allow clauses with more than 1 `rdiff` and `fdiff` literal in the body respectively. The third allowed a maximum of two invented predicates in any clause. Finally the last two prune statements ensured that any `rdiff` and `fdiff` predicates operated on two different squares.

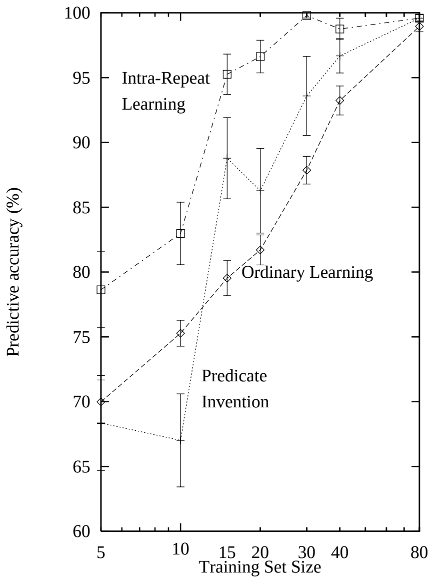
3.4 Runs

For each piece there were 4 trials conducted. In the first trial "Ordinary Learning" Progol was given the opportunity to learn the definition of the piece without any predicate invention, and with only the `fdiff/2` and `rdiff/2` predicates as background knowledge. In the second trial "Predicate Invention", Progol was supplied with the same background knowledge, but this time allowed to invent new predicates. This process is described fully in Sect. 4. In the third trial "*Intra*-Repeat Learning", Progol was supplied with the successful invented predicates from the second trial as background knowledge. Finally in the fourth trial "*Inter*-Repeat Learning", the learning of each piece was helped with the invented predicates from the other piece. This is how it is hoped Repeat Learning would work in general. Similar concepts would share common defining predicates and predicate invention performed on some of them could be used to help the learning of others.

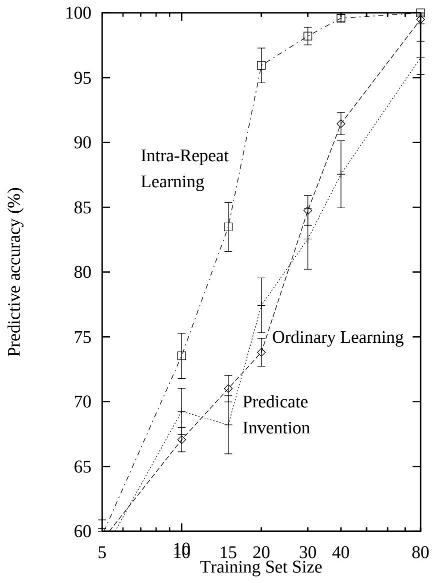
The accuracy results for Ordinary Learning, Predicate Invention and Intra-Repeat Learning are plotted in Fig. 2 (standard deviations for the sample mean over 20 runs are shown as error bars). A comparison of Inter-Repeat Learning to Ordinary Learning is given in Fig. 3.

3.5 Analysis of Results

The results obtained clearly show that predicate invention on the one hand does not significantly increase accuracy, but on the other hand drastically increases training time. Taken within a RL setting however, the results indicate that RL using PI leads to both significant gains in accuracy and training time. In particular there is approxiamtely a 2-fold reduction in sample complexity for accuracy levels between 70% and 100%. Although the somewhat ideal and artificial case of learning the same concept (Intra-Repeat Learning) was primarily considered, the results are seen to hold also when learning related but not equal concepts such as the King and Knight move definitions (Inter-Repeat Learning).

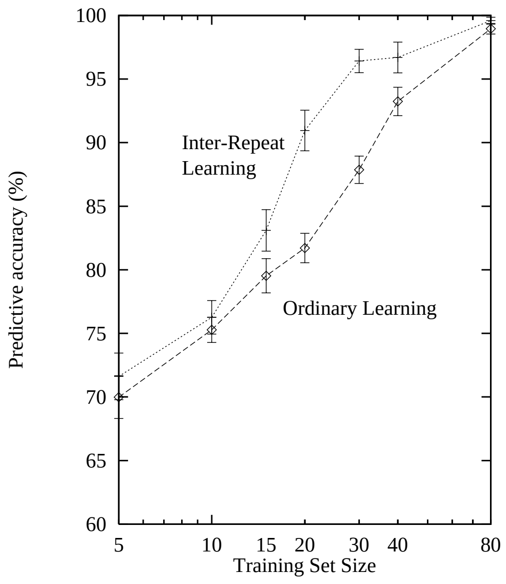


a) The King

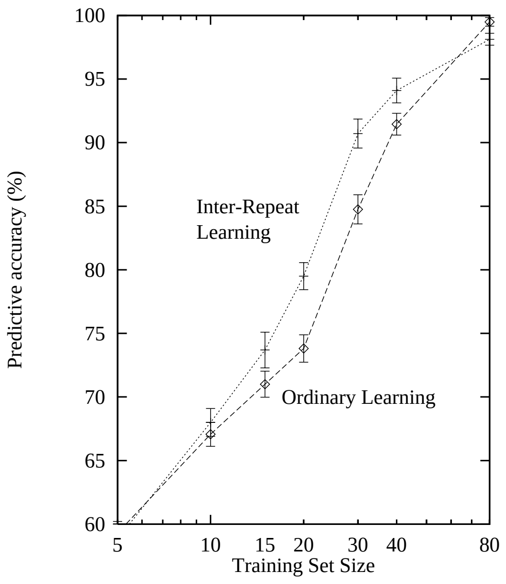


b) The Knight

Fig. 2. Learning curves for each piece (with standard deviations of sample means)



a) Learning King using Knight symmetries



b) Learning Knight using King symmetries

Fig. 3. The Inter-Repeat Learning Setting

4 PI Using Constraint Solving

Predicate invention in the chess experiments (Sect. 3) is carried out using the constraint-solving mechanism of Progol4.4[1]. To allow invention of arity 1 predicates the following declarations are made within Progol's input file.

```
:- modeb(1,invent(#pn,+pmrank))?
:- constraint(invent/2)?
```

The first argument of `invent/2` must be a predicate symbol (actually a pseudo-predicate symbol) of type `pn` and the second argument (representing the argument of the invented predicate) is a variable of type `pmrank`. The constraint declaration says that any invent/2 atom in the most-specific clause should have a Skolem constant in place of the predicate symbol to be invented (`#pn` above). In the search, during any clause refinement which introduces a constraint atom, the flag 'solving' is turned on, and `invent/2` is called with the second argument bound to a term representing all substitutions for this variable from the positive and negative examples. The bound term is given as a list of lists of lists(takes the form `[P,N]` where `P` is from the positive examples and `N` from the negatives, and `P`,`N` are lists of lists, each list giving all substitutions related to a particular example). The predicate `invent/2` returns an appropriate substitution for the constant, which will be either an existing predicate symbol or a new predicate symbol created using `gensym/1`. This predicate symbol replaces the Skolem constant in subsequent testing of the clause and its refinements. Definitions for constraint predicates have at least two clauses, with guards '`not(solving)`' and '`solving`' to define respectively the normal application of the predicate and the procedure for computing the constant. The following is the user-defined definition of `invent/2` used for the experiments.

```
invent(P,X) :-                % Non-solving clause
        not(solving),
        !, defn(P,X), !.      % Simply use definition of P
invent(P,[PosA,_]) :-
        solving,              % Solving clause,
        flatten(PosA,PosA1),  % Flatten pos substs.
        sort(PosA1,PosA2),    % Make normal form model for P.
        not(element(X,PosA2),not(number(X))),
                              % Check no Skolem constants.
        (clause(model(P,PosA2),true);        % Already exists?
         (gensym(P),          % Create new pred. symbol P.
          asserta(model(P,PosA2)), % Assert the model of P.
          assertdef(P,PosA2))). % Assert the definition of P.
```

The auxiliary predicates `defn/2` and `model/2` store the definitions and models of invented predicates respectively. The definition has a set of clauses associated

[1] This is an adaptation of Srinivasan and Camacho's lazy evaluation, originally devised to allow the estimation of numerical constants within clauses

with each invented symbol while the model has a single clause containing a list of all ground instances of a particular invented predicate. For instance, q8 might have a definition and model as follows.

```
defn(q8,-1).  defn(q8,0).  defn(q8,1).
model(q8,[-1,0,1]).
```

The model can be used to check whether a new predicate has already been invented. The definition allows for further generalisation of the invented predicate (given appropriate mode declarations).

Despite the apparent simplicity of the approach, the predicate invention technique allows simultaneous invention of multiple predicates, and also allows user choice of arity and argument typing for inventable predicates. In fact, the technique appears to be at least as powerful as anything in the literature. For instance, in the release examples of Progol4.4 (available by anonymous ftp from `ftp.cs.york.ac.uk` in directory `pub/ML_GROUP/progol4.4`), it is shown how `append1/3` can be invented when learning a recursive definition of naive reverse.

5 A Framework for Repeat Learning

The following describes a framework for the Repeat Learning setting, similar to one described by Baxter in [1]. This framework is then applied to ILP. This setting is a natural extension of the U-learnability model [8], [5]. The framework is a generalisation of the setting used in the experiments.

The teacher starts by choosing a distribution D_T over the target space $\mathcal{H}$, and a distribution D_X over the instance space X, where $\mathcal{H} \subseteq \mathcal{P}(X)$.

The learner $\mathcal{L}$ is given the distribution D_X, and is given a *prior distribution* D_1 over the target space $\mathcal{H}$. $\mathcal{L}$ is also equipped with a *repeat–adjustment* operator $r_{\mathcal{L}}$ that produces a refined posterior distribution D_{n+1} on the basis of all the data received by $\mathcal{L}$ so far.

The teacher uses D_T and D_X to carry out an infinite series of teaching sessions, starting with session 1. In session $n \geq 1$ the teacher chooses a target concept $T_n \in \mathcal{H}$ according to D_T. T_n is used to provide labels from $\{true, false\}$ for a new sequence of instances $\langle x_1, x_2, \ldots \rangle$ chosen randomly and independently from X according to D_X. An instance x_i is labelled true if $T_n \models x_i$, and false otherwise. An hypothesis is said to explain a set of examples E whenever it both entails and is consistent with E.

In session n, having seen the labelled instances $E_{m_n} = \langle e_1, e_2, \ldots e_{m_n} \rangle$, $\mathcal{L}$ produces a hypothesis $\mathcal{L}(E_{m_n}, D_n)$ that explains E_{m_n}, using its current prior distribution over $\mathcal{H}$, D_n. The teacher stops the session after m_n labelled instances, if the learner's hypothesis $\mathcal{L}(E_{m_n}, D_n)$ has expected error less than ϵ (for some $0 \leq \epsilon < 1$) for predicting the correct label of any $x \in X$ chosen according to D_X. This successful hypothesis is then labelled H_n. The learner then generates a new prior distribution D_{n+1} for session $n+1$, using $r_{\mathcal{L}}$.

A repeat learner would be expected to show an improvement in performance after a number of sessions, by successively improving its approximation D_n of the teacher's distribution D_T over the target space. See Sect. 5.2 for a more precise description of this criterion.

5.1 Repeat Learning Under a Fixed Representation

The above framework can be extended to learning systems (such as ILP) that use a fixed representation during learning sessions.

The learning system is given an hypothesis language L that can describe target concepts in $\mathcal{H}$. We assume the existence of a universal linguistic bias generator G that, given a target space $\mathcal{H}$, and an hypothesis language L for it, returns a probability distribution $D_H = G(\mathcal{H}, L)$ over the target space. Any distribution consistent with Occam's razor can be taken as an example of such a generator. Such a distribution would assign a higher probability to hypotheses that can be expressed more simply in the hypothesis language.

It is assumed that the teacher has a predefined hypothesis language L_T, and that the bias generator gives the probability distribution $D_T = G(\mathcal{H}, L_T)$ over $\mathcal{H}$ for this language.

The learner has an initial hypothesis language L_1, giving the prior probability distribution $D_1 = G(\mathcal{H}, L_1)$ as before. The learning protocol for session n proceeds as before, but with the change that the learning algorithm's repeat–adjustment operator adjusts instead the hypothesis language L_n (using, for instance, predicate invention and retraction) to give language L_{n+1}, and thereby inducing a new distribution D_{n+1} over the target space.

The requirements for a repeat learner in this setting are the same as in Sect. **??**, except that the performance improvement is achieved by successively improving the approximation L_n of the teacher's hypothesis language L_T.

5.2 A Criterion for Repeat Learning

A tentative set of conditions that a machine learning algorithm must satisfy in order to qualify as a repeat learning algorithm will now be described. The authors would like to make it clear that the necessity, sufficiency and satisfiability of these conditions have yet to be established. The conditions are related to the U-learnability criteria described in [8].

Let the distributions D_T, D_X have associated parameters $\nu_{D_T}, \nu_{D_X} \geq 0$ (for example, the parameters might be the greater of the mean and standard deviation of the distributions). The parameters may provide some indication of the difficulty of learning from the distributions.

Let $\mathrm{LEARNTIME}(x) = x^c$ for some $c > 0$. Let $\mathrm{IMPROVEMENT–DELAY}(x, y)$, $\mathrm{LEARN–DELAY}(x, y)$, $\mathrm{ERROR–BOUND}(x, y)$ be polynomial functions of x and y. Let $\boldsymbol{X}_m$ be an m-tuple of instances chosen randomly and independently from X according to D_X.

Define $E_{\mathcal{L}}(T, \boldsymbol{X}_{m+1}, n) = 0$ in session n, if $\mathcal{L}$ correctly classifies (w.r.t T) instance x_{m+1}, given instances $\langle x_1, \ldots x_m \rangle$ labelled according to T, and $E_{\mathcal{L}}(T, \boldsymbol{X}_{m+1}, n) =$ 1 otherwise (where $\boldsymbol{X}_{m+1} = \langle x_1, \ldots x_{m+1} \rangle$).

Time Complexity The average case time complexity of $\mathcal{L}$ in any learning session is bounded by LEARNTIME(M), where M is the sum of $p(|x_i|)$ over the examples x_i seen to that point in that session. The time complexity of the repeat–adjustment operator r_L is unrestricted.

Correctness For all n, For all $m \geq$ LEARN–DELAY(ν_{D_T}, ν_{D_X}):

$$\sum_{all(T_n,\boldsymbol{X}_{m+1})} Pr_{D_T,D_X}(T_n,\boldsymbol{X}_{m+1})E_{\mathcal{L}}(T_n,\boldsymbol{X}_{m+1},n) < \text{ERROR–BOUND}(\frac{1}{m},1)$$

Improvement For all $n \geq$ IMPROVEMENT-DELAY(ν_{D_T}, ν_{D_X}), For all $m \geq$ LEARN–DELAY(ν_{D_T}, ν_{D_X}):

$$\sum_{all(T,\boldsymbol{X}_{m+1})} Pr_{D_T,D_X}(T,\boldsymbol{X}_{m+1})E_{\mathcal{L}}(T,\boldsymbol{X}_{m+1},n) < \text{ERROR–BOUND}(\frac{1}{m},\frac{1}{n})$$

6 Discussion

This paper has introduced a theoretical framework for machine learning a series of related concepts, which we call the *Repeat Learning* setting. This framework was extended to an ILP setting, and the possibility of using Predicate Invention under ILP to achieve the aims of the framework was discussed.

We have shown in experiment how Predicate invention under Repeat Learning can decrease the sample complexity requirements of an algorithm. However a comprehensive theoretical explanation of this is yet to be found. Surprisingly little experimental and theoretical evidence of the utility of Predicate Invention exists. The results of this paper indicate that Predicate Invention is useful under the Repeat Learning setting.

Many, indeed most, of the real–world domains in which ILP has been successful can be rephrased in the repeat learning setting. Consider, for instance, the Mutagenicity and Predictive Toxicology domains. Chemical bond and structure relationships learnt in early problems can be reused, and such relationships could potentially improve the accuracy of the algorithm in later problems.

Machine learning experiments in real–world domains aim to show that an algorithm is successful at learning a certain *type* of problem. This is only useful when there are other unsolved problems of the same type to which a successful algorithm can then be applied. Repeat Learning may be beneficial in such multiple–problem domains. There is clearly a need for further theoretical analysis of Repeat Learning in ILP.

7 FTP site

The datasets used in the experiments described in this paper are available from anonymous ftp at: `ftp.cs.york.ac.uk/pub/ML_GROUP/Repeat/`

Acknowledgements

Thanks to David Page for vital input in discussions on this topic. This work was supported partly by the Esprit Long Term Research Action ILP II (project 20237), EPSRC grant GR/K57985 on Experiments with Distribution-based Machine Learning, an EPSRC Advanced Research Fellowship held by the second author and EPSRC postgraduate awards held by the first and third authors.

References

[1] J. Baxter. Theoretical models of learning to learn. In T. Mitchell and S. Thrun, editors, *Learning to Learn*. Kluwer, Boston, 1997.

[2] J. Goodacre. Inductive learning of chess rules using progol. Master's thesis, Oxford University, Oxford University Computing Laboratory, 1996.

[3] C. Ling. Inventing necessary theoretical terms in scientific discovery and inductive logic programming. Technical Report 302, Dept. of Comp. Sci., Univ. of Western Ontario, 1991.

[4] S. Muggleton. Predicate invention and utilisation. *Journal of Experimental and Theoretical Artificial Intelligence*, 6(1):127–130, 1994.

[5] S. Muggleton. Inverse entailment and progol. *New Generation Computing*, 13:245–286, 1995.

[6] S. Muggleton. Learning from positive data. In *Proceedings of the Sixth Workshop on Inductive Logic Programming*, Stockholm, 1996.

[7] S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the 5th International Conference on Machine Learning*, pages 339–352. Kaufmann, 1988.

[8] S. Muggleton and C. D. Page. A learnability model for Universal representations. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, 1994.

[9] I. Stahl. Constructive induction in inductive logic programming: an overview. Technical report, Fakultat Informatik, Universitat Stuttgart, 1992.

[10] I. Stahl. Predicate invention in inductive logic programming. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 34–47. IOS Press, Ohmsha, Amsterdam, 1996.

[11] R. Wirth and P. O'Rorke. Constraints on predicate invention. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 457–461. Kaufmann, 1991.