Using Meta-Languages for Learning

Simon Anthony and Alan M. Frisch

Department of Computer Science, University of York, York. YO1 5DD. UK.
Email: {simona, frisch}@minster.cs.york.ac.uk

Abstract. This paper proposes the use of meta-languages to provide an effective representation scheme for Machine Learning. This proposal is supported by both:

- a demonstration of the benefits that meta-languages can bring to Inductive Logic Programming, and
- the successful and widespread use of meta-languages in other search intensive areas of Artificial Intelligence, such as planning.

1 Introduction

A typical Machine Learning setting involves a *teacher* selecting a *target concept* from a *concept space*, and providing a *learner* with a *labelled training sample* of *examples* of this concept. Each example is drawn from an *example space*, and is labelled *positive* if it is an example of the target concept, or *negative* otherwise. In response, the learner searches a *hypothesis space*[1] to find a *hypothesis* of the target concept. This selection is influenced by *bias* which both restricts the size of the hypothesis space, and allows a learner to judge between competing hypotheses. To build such a learning system we must choose the most suitable language to represent the concept, example and hypothesis spaces. Many different choices have been suggested, sparking a long standing debate within Machine Learning.

In this paper, we address this representation question from a different angle, and proposes that a distinction should be made between the representation of the space searched by the learner, and the representation of the hypothesis that is reported to the teacher. More specifically, we propose that a learner should search a space of *descriptions* of hypotheses (represented in some *meta*-language) and, from the selected description, obtain and report a specific hypothesis (represented in some *object* language). Although decisions are still required concerning which meta- and object languages should be used, we are primarily concerned with promoting the use of meta-languages for learning.

Our proposal is supported by examining the benefits that using meta-languages can bring to Inductive Logic Programming—a Machine Learning paradigm which uses logic programs as the hypothesis language. Further support is provided by briefly looking at the successful use of meta-languages in planning—another area of Artificial Intelligence involving search. We conclude by considering the benefits of a more general use of meta-languages within Inductive Logic Programming and Machine Learning.

2 Inductive Logic Programming

Inductive Logic Programming (ILP) [7,4] is a recent and rapidly expanding area of Machine Learning, distinguished by its use of a very expressive object language—that of logic programs. This

[1] The concept and hypothesis spaces are often assumed to be equal.

expressiveness has proved rather a mixed blessing—on the one hand, it has allowed ILP to be successfully applied to problems interesting to researchers from other disciplines, most notably problems in Biochemistry [6,10]. On the other hand, it is the source of a number of shortcomings. Logic Programming has always had difficulty in handling numbers and ILP inherits these. Additionally, learning algorithms often impose an ordering over clauses and literals in hypotheses which can cause the hypothesis space to contain equivalent members. We briefly review these problems together with some recent attempts at their solution, and show that a common feature of each is their (albeit limited) use of a meta-language.

2.1 Handling Numbers

The need to handle numbers in ILP is exemplified by the range of concepts that involve some form of numerical reasoning. However, many ILP learners, including state-of-the-art systems, have difficulty reasoning with numbers. We shall use Progol [5], perhaps the most widely used ILP system, to illustrate our discussion of this problem.

Progol is biased to learn clauses that entail a particular positive example. This bias requires that any constant symbols to appear in a literal must be determined from this single positive example. For *non-numerical*, *uninterpreted* constants this approach is effective since only the constants in the training sample and background knowledge need be considered. In contrast, *numerical* constants are *interpreted* and *ordered* which justifies the inclusion of new constant symbols, not already present in the training sample or background knowledge. Thus, when handling numbers, the bias in Progol causes it to make guesses and *over-commit* to particular numerical constants based on the evidence of a single positive example.

Recently, both Srinivasan and Camacho [9] and Anthony and Frisch [1] have proposed solutions to this problem of handling numbers. Working with the Progol system, Srinivasan and Camacho argue that extra-logical numerical predicates are sufficient for handling numbers, and that constants should be selected *lazily*. Where a position is required to contain a numerical constant, a meta-symbol (a skolem constant) is initially inserted instead. This symbol *describes* the possible numerical constants that might replace it in a hypothesis clause. As such, it prevents Progol over-committing to particular constants from a single positive example.

During the search of the hypothesis space, numerical constants are found to replace the meta-symbols based on the evidence of *all* the examples in the training sample. Srinivasan and Camacho demonstrate how lazy constant selection can be used to solve the abstract pole and cart problem, as well as the non-trivial mutagenesis problem.

Anthony and Frisch adopt a similar approach that can be incorporated in any learning system. An object language of Constraint Logic Programs (CLPs) [2,3] is used with meta-variables occupying positions requiring numerical constants. Examples are then used to place constraints on these variables, attempting to ensure that the chosen constant will allow a clause to entail as many positive and as few negative examples as possible. Once all the examples have been considered, numerical constants for the literal can be found by solving the constraints on each meta-variable.

Notice that the use of a meta-language in these two approaches allows a learner to *delay* its choice of numerical constants for particular positions within a numerical literal.

2.2 Avoiding Redundancy

Consider the problem of preventing a hypothesis space from containing two equivalent hypotheses, a problem known as *redundancy*. Many learners, including Progol, search the hypothesis space by repeatedly adding literals to the bodies of clauses and clauses to hypotheses in an ordered fashion, known as *refinement*. However, since clauses are *sets* of literals and hypotheses are *sets* of clauses, refinement imposes an ordering over literals and clauses which can cause redundancy.

It appears that redundancy can be avoided by learning a clause as a *partial* ordering of literals, and a hypothesis as a *partial* ordering of clauses. Such partial orderings can be represented in a meta-language, and each describes a number of total orderings. For instance, figure 1 shows how the following clause might be represented by a meta-language of trees in which the depth of each node corresponds to the depth of the deepest variable symbol in each literal.

$$p(X, Z) \leftarrow q(X, Y), r(Y), s(Y), t(X, Z)$$

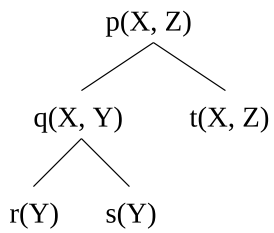

Fig. 1. Describing a clause using a meta-language of trees.

A totally ordered clause can be found from such a partial order description by a pre-order traversal of the tree. Thus, the use of a meta-language can also allow a learner to delay its choice of an ordering for literals in clauses and clauses in hypotheses, avoiding redundancy.

3 Meta-Languages in Planning

The benefits of using meta-languages in search problems is apparent by their use in other areas of Artificial Intelligence (AI). Planning is good example of such an area, since the relationship between a plan and its component actions is not dissimilar from that of a clause and its component literals.

Early research considered a plan to be a totally ordered sequence of actions in much the same way as current ILP research considers a clause as a totally ordered sequence of literals. As such, early planners search a space of such sequences in order to solve planning problems. However, it was soon discovered that they failed to find an optimal plan in even very simple planning problems, such as the infamous Sussman anomaly.

This problem persisted until Sacerdoti [8] noticed that although a plan is *executed* as a total order, the plan can be *described* as a partial ordering of actions with respect to time. Therefore, a planner can search a space of *descriptions* of plans, and produce a plan from the selected description. This use of meta-languages in planning has provided a solution to the Sussman anomaly, and has become the widely accepted standard approach to planning in AI.

4 Conclusion and Directions for Research

The use of meta-languages is not new to Machine Learning. Indeed, bias is often represented by a meta-language since its purpose is to describe 'good' hypotheses. Nor are they new to other areas of AI involving search. For instance, recall that the introduction of meta-languages in planning was revolutionary, and quickly become adopted as standard.

Research is therefore needed to establish the potential for using meta-languages in ILP, and also Machine Learning in general. As a first step, hypotheses might be described using a similar representation to that of plan descriptions—as a set of partial orderings of literals. With the ability to delay the selection of constants and the ordering of literals as seen above, such a meta-language should provide a powerful platform for learning, since it shoulders the expressive object language of logic programs whilst avoiding several of the corresponding shortcomings.

To this end, we are currently investigating the use of CLP languages for both the meta- and the object languages for learning. The early results are encouraging, and we hope to further demonstrate the potential of meta-languages through the successful application of ILP to new problems—for instance, problems involving numbers from the worlds of finance and economics.

References

1. S. Anthony and A. M. Frisch. Generating numerical literals during refinement. In *Proceedings of the 7th International Workshop on Inductive Logic Programming*. Springer Verlag, 1997.
2. J. Jaffar and J-L. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, pages 111–119, 1987.
3. J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19-20:503–583, 1994.
4. S. Muggleton. Inductive logic programming. *New Generation Computing*, 8:295–318, 1991.
5. S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.
6. S. Muggleton, C. D. Page, and A. Srinivasan. An initial experiment into stereo-chemistry-based drug design using ILP. In *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 245–261. Published as a technical report of the Royal Institute of Technology, Stockholm University, 1996.
7. S. Muggleton and L. De Raedt. Inductive logic programming – theory and methods. *Journal of Logic Programming*, 19-20:629–679, 1994.
8. E. D. Sacerdoti. The nonlinear nature of plans. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 206–214, 1975.
9. A. Srinivasan and R. Camacho. Experiments in numerical reasoning with inductive logic programming. To appear, 1997.
10. A. Srinivasan, S. Muggleton, M. J. E. Sternberg, and R. D. King. Theories for mutagenicity: A study in first-order and feature-based induction. Technical Report PRG-TR-8-95, Oxford University Computing Laboratory, 1995.

Acknowledgements We wish to thank Alistair Willis for his comments on this paper. The first author is funded by an EPSRC grant and by a travel award from Data Connection Ltd.