Poster presentation at the *19th Intl Conf. on Inductive Logic Programming*, Leuven, Belgium, 2009.

ILP for Bootstrapped Learning: A Layered Approach to Automating the ILP Setup Problem

S. Natarajan[*], G. Kunapuli[*], C. O'Reilly[**], R. Maclin[+], T. Walker[*], D. Page[*], & J. Shavlik[*]

[*]Depts. of Biostats & Medical Informatics and Computer Sciences, Univ. Wisconsin - Madison

[+]Department of Computer Science, University of Minnesota-Duluth

[**]Artificial Intelligence Center, SRI International

Abstract. This paper introduces a new type of application for ILP called Bootstrapped Learning (BL). BL brings several challenges to ILP, including the need to (a) automate the "ILP setup" problem, (b) exploit the fact that a well-meaning teacher is providing pedagogically chosen examples and may be offering hints, (c) deal with small numbers of training examples and sometimes no explicit negative examples; and (d) "bootstrap", i.e., to automatically base learning in part on the results of earlier learning sessions.

Keywords: helpful teachers, running ILP systems without human intervention

1 Introduction

It has long been recognized that ILP systems require substantial knowledge engineering in preparation to run on a new dataset or domain, especially in contrast to standard feature-vector learners such as decision trees or support vector machines. Motivated by a new area of machine learning – Bootstrapped Learning (BL) – this paper addresses the task of, at least partially, automating the process of preparing an ILP algorithm to run on a new dataset or domain.

Bootstrapped Learning. BL is a new machine-learning paradigm, proposed by Dan Oblinger [3], that focuses on learning progressively more complicated concepts through a "ladder" of lessons; lower (earlier) rungs of the lesson ladder teach simpler concepts needed to learn the concepts at the higher rungs. The key assumption underlying BL is the presence of a helpful teacher. Therefore, the emphasis in BL is on efficient communication between teacher and student, rather than on *de novo* discovery as in much of the rest of supervised machine learning.

In the BL paradigm, the machine student should be able to learn from a variety of modalities of teacher input, including from pedagogical examples, from being told, from noticing, and from experimentation and feedback. One of the motivations of BL is that it may be easier for a teacher to instruct, say, an agent to play "Robocup" than to program it to do so. The teacher might tell the student the rules, give examples of the ball being in or out of bounds, provide initial strategy advice, and then give feedback as the student plays the game, much as a human coach would do. One can imagine similar situations in domains where a teacher instructs an unmanned aircraft how to fly a recon mission, or teaches a system how to diagnose a problem on a ship.

Inductive logic programming [1] is especially well-suited for the "learning from examples" component of a BL system for two reasons. First, it can use a rich knowledge base that may have been provided to the learner initially or may have been learned/augmented during earlier lessons. Second, the declarative representation of both examples and learned rules makes it easier for teacher and student to communicate about what has been learned so far; for example, a teacher can identify and correct student mistakes from earlier lessons. Similarly, the use of logic allows

for sharing lessons of learned knowledge between modules that learn from different kinds of instruction. This paper describes our work on ILP within the context of 18 months of a large project to construct a BL student.

Motivations and a Proposed Approach. BL poses several key challenges for ILP, addressing them can drive future research, and will be beneficial for other applications of ILP beyond BL, which could extend the usability of ILP systems to non-ILP experts. The major BL challenge for ILP is that ILP has to be used, not only for different lessons within the same domain, but also across different domains; this necessitates the automation of the ILP setup problem *without the intervention of an ILP expert*.

Another important aspect requiring automated ILP runs is that the *parameter settings cannot change between different runs*. We cannot expect any human guidance regarding settings and need to find good default values that work broadly. Actually, we are able to have our algorithms themselves try out a few parameter settings and use cross validation to choose good settings. However, given the large number of parameters in typical ILP systems (maximum rule length, modes, minimal acceptable accuracy of learned clauses, etc.), our algorithms cannot exhaustively try all combinations and hence we must choose an appropriate set of candidate parameter settings that will work across dozens of learning tasks.

A separate group of researchers is tasked with producing natural lessons for the BL student across different domains. These lessons are taught via natural instruction methods designed to be used by non-experts; the "teaching team" provides an appealing source of lessons from outside the ILP community (these lessons will be made publicly available).

Motivated by our prior experiences with ALEPH [5], we are developing a Java-based ILP system called the Wisconsin Inductive Logic Learner or WILL**,** for which the BL framework caused us to add capabilities not found in ALEPH. Our main approach to automatically handling domains is via a multi-layer strategy that investigates various combinations of strategies such that the hypothesis space is steadily expanded until an acceptable theory is learned. We discuss the approach, which is a work in progress, in Section 3. We conclude in Section 4 by presenting some directions for future research.

2 Bootstrap Learning Domains

The domains of the BL project we have been provided so far are Blocks World, RoboCup [2], and Unmanned Aerial Vehicle (UAV) control. We assume the first two already are familiar to the ILP community. In Blocks World, one of the goals was to learn the concept of isAStack given only positive examples. WILL was also used to learn makeStack which requires the student to learn a plan using ILP. In RoboCup, one task was to learn whether a ball is outOfBounds given the real-valued position of the ball.

The UAV domain involves operating a UAV and its camera to execute a reconnaissance mission. Tasks include determining if the UAV has enough fuel to accomplish a mission, achieving appropriate latitude, altitude, etc., of the UAV, achieving appropriate pan, tilt and zoom of its camera, and recognizing which objects in the camera's field of vision are of interest. The learned has to deal with complex

structures such as position, which consists of attributes such as latitude, longitude, altitude, etc. Encoding these spatial attributes as part of one position literal would enable WILL to learn a smaller clause, but would increase the branching factor during search due to the additional arguments introduced by such a large-arity predicate. Representing these spatial attributes as separate predicates would decrease the branching factor at the expense of the target concept being a longer clause. In addition, the tasks involve learning the concept of "near" that can exist between any two objects of interest. In a later lesson, this concept might be used, for instance, to determine if a truck is at an intersection. It is a challenge for the ILP systems to automatically generalize and specialize at different levels of the type hierarchy.

3 Tackling the Bootstrapped Learning Challenges for ILP

Often, researchers face the problem of designing new predicates, guiding ILP's search, setting additional parameters, etc. These domain-specific necessities greatly limit the applicability of an ILP system across different problems. BL brings a major challenge for ILP in this area, because WILL must automatically set up training without the intervention of an ILP expert. This is needed because teachers cannot be expected to understand the algorithmic details of a learning approach; rather they communicate with the student in and as natural and human-like dialog as is feasible **[**3**]**. This necessitates the guiding of search automatically in a *domain independent* manner. Automatic parameter selection methods such as the one proposed in [7] are not useful in our system due to the fact that we do not have access to a large number of examples. Instead we resort to a multi-layered strategy that tries several approaches to learn the target concept.

Generation of Negative Examples. In general, ILP requires a large number of examples (both positive and negative) to learn a concept. While this is a challenge in all of supervised learning, the need to sometimes learn complex relational concepts makes it even more so in ILP. In some domains, it is natural for a teacher to say that a particular world state contains a single positive example; for example, it is natural for a teacher to point to a set of three blocks and state that they form a stack. It is a reasonable assumption that various combinations of the rest of the blocks in that scene do not form a stack and hence, WILL assumes these are (putative) negative examples. We have found that for most of the lessons provided in BL there is such a need for automatically constructing negatives [6] because instruction contains mainly positive examples. This issue of limited or positive-only data arises because human teachers often provide only a few carefully and pedagogically chosen examples rather than hundreds of examples drawn randomly from some probability distribution.

Another natural way for expressing negative examples is to say some world state does *not* contain any instances of the concept being taught: "the current configuration of blocks contains no stacks", for example. Here, WILL is more confident about the negative examples it creates from such instruction. Assume the teacher indicates `isaStack` takes three arguments, each of which is of type `block`. If WILL is presented with a world containing N blocks where there are no stacks, it can create N^3 negative examples. Such a scenario occurs in the UAV domain, where the goal is to learn if two objects are near one another. The teacher might present an instance and point out that no two objects are near one another. In general, negative examples are

generated by instantiating the arguments of predicates whose types we may have been told (if not, their type is `any`), in all possible ways using typed constants encountered in world states; finally, examples known to be positive are filtered out. Depending on the BL task, the BL student may have either teacher-provided negatives (either directly specified or via a world state that contains no positives) or induced negatives. As we do not want to treat these identically, WILL allows *costs* to be assigned to examples ensuring that the cost of covering a putative negative can be *less* than covering a teacher-provided one.

Learning the Negation of a Concept. Human teachers typically gauge the difficulty of concepts being taught by human comprehensibility, in terms of which, accurate, short, conjunctive rules are preferred. When learning concepts such as `outOfBounds` in a soccer field, the target concept might have a large set of disjunctions (since it can be out of bounds on any of four sides). It is easier to learn if the ball is *in bounds* and then *negate the learned concept*. So, one general heuristic in WILL is:

```
When learning P(x1, ..., xn)
  look for one or more rules for predicting P(x1,..., xn)
 that individually have high coverage and high accuracy

  do the same, but now focus on predicting not P(x1,..., xn)
```

Our inductive bias here is that our benevolent teacher is teaching a concept that is simple to state, but we are not sure if the concept or its negation is simple to state as one or more Horn clauses, so we always consider both.

The main bottleneck was the number of available examples. For a small number of examples, it is usually hard to learn a disjunctive rule, especially if the examples are not the best ones, but rather only 'reasonable' in that they were *near* the boundaries, but not exactly *next to* them.

Automatic Background-Knowledge Generation. One of the bottlenecks for ILP is the creation of background knowledge: modes, facts, type hierarchy, etc. In our setting, it is not possible to hand-craft the background-knowledge and there is a necessity to automate the generation of the knowledge. BL domains have type hierarchies including mode specifications [5], which are used by WILL to control the search for good clauses. We first create the hierarchy by walking through the domain description; then modes are created by traversing this hierarchy. For each predicate, we climb the hierarchy until all the facts match the type. For instance, consider the type `eagle`, whose supertype is `bird` and its supertype, `animal`. For the predicate **`flies`**, it is sufficient to climb the hierarchy up to `bird`, but, for the predicate `numberOfLegs`, we will have to climb all the way up to `animal` (assuming here that all animals have legs but only birds have wings). Once the base types and modes have been constructed, it is possible to add special predicates such as bins (described below), actions, orderings, etc., to the background after mode construction. The existence of type hierarchies means WILL needs to handle hierarchical mode specifications, which led to some technical challenges when controlling the expansion of candidate clauses during WILL's search.

Handling Hints from Teacher. In our setting, the teacher can specify *relevance information* that provides advice while learning the target concept. Relevances can be specified at varying resolutions: a particular attribute, object, type or even the relationship between predicates (less than, greater than, etc.) can be designated relevant by the teacher. As the number of examples is low in BL, relevant information becomes quite significant. Relevance statements are exploited by WILL to speed up learning by introducing costs on predicates, which guide the search towards a minimal-cost solution. Our heuristic scheme is used to assign costs to predicates based on an ordering of relevance information. This is based on the imperatives provided by the teacher (indicating some features as more relevant than others) as well as the *specificity* of a designated relevant (more specific relevant features have lower costs).

Using Feedback. It is natural for the teacher to provide some kind of feedback to the target concept that WILL has learned. This feedback could be a judicious example that could guide WILL towards the correct concept. The feedback could also explain that a particular predicate is relevant and needs to be included in the target predicate. Yet another method of providing feedback is to present an example and explain why the target concept is true or false. This explanation could be a part of a disjunction that was not considered by WILL. We are currently working on incorporating such feedback in WILL. Extending WILL to address the problem of theory refinement in the lines of [8] remains an interesting direction for future research.

Handling Numeric Data. Several BL domains contain substantial numeric data and require WILL to perform numeric reasoning. The tasks may range from reasoning problems such as `outOfBounds` described above, to more complex tasks such as being able to learn a numeric relationships among data features. Our approach to introducing numeric reasoning is by adding several basic capabilities to WILL's search space. These include simple mathematical operators, such as `plus`, `product,` etc., and comparators for equalities and inequalities, including allowing for comparison up to a tolerance, e.g., `equalsWithTolerance`.

Tiling/Binning of Numeric Features. The numeric features (location of the ball and field dimensions) can be discretized by thresholding. For each numeric attribute, we sorted the values and determined the transition values (boundary of the bins) between positive and negative examples following the method used in decision-tree induction [4], creating a predicate corresponding to each bin. The learned `outOfBounds` predicate is shown below, where the thresholds are encoded in the predicate names ('m' means 'minus'; constructs like '10_5' represent floats like *10.5*):

```
outOfBounds(Ball, Field) :-
  NOT( position(Ball, P),
       locationX_gte_m10_5(P), locationY_gte_m10_5(P),
       locationY_lte_10_5(P),  locationX_lte_10_5(P) ).
```

As can be seen, the negated target concept was learned. We are currently extending this to 2D and 3D bins, since many domains involve reasoning about spatial features. It should be noted that the combinatorics grow quickly with the number of numeric features. One possible improvement is to consider only the *numeric attributes of the same type* while binning the values.

Automating Learning of Simple Plans. WILL was also used to learn simple plans, such as making a stack of three blocks from a set of blocks on the table. To deal with time, we incorporate the standard notion of state in the predicate. This led to a simple version of situation calculus that allows WILL to learn plans. Using WILL for planning tasks introduced another challenge. ILP typically discriminates positive examples from negatives. However, we need ILP to generate good plans and this requires a sequence of actions from the initial state to a terminal state (or possibly a set of them). For example, it is insufficient to simply learn that to separate the positive and negative cases of `makeStack`, the second step must involve a block with nothing on top of it.

We addressed this need for *generative* plans rather than *discriminative* clauses by adding to WILL, the ability to say that a certain literal (in this case `finalState`) must be in a clause for that rule to be acceptable. We also extended the modes used by WILL to include the ability to limit the number of times a given variable could be used in a clause. For example, we constrain the first argument of an action predicate to be a variable that appears exactly once in the existing clause if action() is to be added. This ensures that the rule WILL learns produces a linear sequences of `state` variables in actions. The learned plan is

```
makeStack(B1, B2, B3, State1)  :-
        action(State1, moveOnto(B2, B1), State2),
        action(State2, moveOnto(B3, B2), State3),
        finalState(State3).
```

This plan is buggy - the teacher did not show WILL how to deal with the case where blocks to be moved currently have another block on top of them. Such refinement was left to another lesson, in this case one where a reinforcement-learning agent was allowed to practice building stacks in the blocks world.

Yet another interesting issue is that WILL had to deal with *partially ordered plans* (moving one block from another or to move the bottom block of the stack to be on the table, etc.). We used a special predicate called `actionsInAnyOrder` to deal with this. Similarly, some of the steps could be optional for which we used another special predicate: `isAnOptionalStep`. Finally, the random creation of negatives for plan generation is not a trivial problem, since subsequent states need to be "physically realizable," and is an interesting research question.

A Multi-Layered Strategy for Controlling the Hypothesis Space. It is infeasible to empirically consider all the possible parameter settings for WILL owing to the combinatorial explosion of the possibilities. To this end, we are developing a *multi-layered strategy* that can try various (manually chosen) combinations in the hope of being able to automatically handle a variety of lessons in a diversity of domains.

The innermost layer implements the basic strategy: invoking WILL after automated mode construction, using only the relevant features (as told by the teacher), cross-validation to score a small set of candidate parameters. This means that WILL initially explores a very restricted hypothesis space. If no theory is learned or if the learned theory has a poor score (based on heuristics), then the hypothesis space is expanded, say by considering features not mentioned by the teacher and allowing for

longer clauses. Continuing this way, our multi-layered approach successively expands the space of hypotheses until an acceptable theory is found.

4 Conclusion

We have motivated the problem of Bootstrap Learning, presented the challenges for ILP in the different domains, and described the initial set of strategies we adopted to solve these problems. The initial results for the project are intriguing. Since we describe our current experiences with building a large-scale system using ILP, we do not present quantitative results here. Rigorous evaluation of our system remains an interesting direction for future research. The automation of ILP runs is critical in several problems beyond BL where human intervention is not feasible between problems. Currently, we are focusing on our layered approach, called the ONION, to more robustly automate ILP in these different tasks. Also, we are currently looking at more richly exploiting teacher-provided feedback beyond statements about which features and objects are relevant.

References

[1] S. Muggleton and L. de Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.

[2] I. Noda, H. Matsubara, K. Hiraki, and I. Frank. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12:233–250, 1998.

[3] D. Oblinger. *Bootstrap Learning-External Materials* (**`www.sainc.com/bl-extmat/)`**, 2006.

[4] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[5] A. Srinivasan. *The Aleph Manual*, 2001. **`web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/`**

[6] S. Muggleton. Learning from positive data. *ILP* 1997

[7] R.Kohavi and G. John. Automatic parameter selection by minimizing estimated error, *ICML* 1995.

[8] L. De Raedt. *Interactive Theory Revision: An Inductive Logic Programming Approach*, Academic Press, 1992.