Appears in Proceedings of the 20th International Conference on Inductive Logic Programming

Automating the ILP Setup Task: Converting User Advice about Specific Examples into General Background Knowledge

Trevor Walker[1], Ciaran O'Reilly[2], Gautam Kunapuli[1], Sriraam Natarajan[1], Richard Maclin[3], David Page[1] and Jude Shavlik[1]

[1]University of Wisconsin – Madison, USA
{twalker, kunapg, natarasr, shavlik, page}@biostat.wisc.edu

[2]SRI International
ciaran.oreilly@sri.com

[3]University of Minnesota, Duluth, USA
rmaclin@d.umn.edu

Abstract. Inductive Logic Programming (ILP) provides an effective method of learning logical theories given a set of positive examples, a set of negative examples, a corpus of background knowledge, and specification of a search space (e.g., via mode definitions) from which to compose the theories. While specifying positive and negative examples is relatively straightforward, composing effective background knowledge and search-space definition requires detailed understanding of many aspects of the ILP process and limits the usability of ILP. We introduce two techniques to automate the use of ILP for a non-ILP expert. These techniques include automatic generation of background knowledge from user-supplied information in the form of a simple relevance language, used to describe important aspects of specific training examples, and an iterative-deepening-style search process.

Keywords: Advice Taking, Human Teaching of Machines.

1 Introduction

Inductive Logic Programming (ILP) provides a method to learn logical theories that cover most of a given set of positive examples and as few as possible of a set of negative examples. Unlike many other supervised learning approaches, ILP often needs a complex corpus of background knowledge beyond just information provided as part of the example description (i.e., the example features). This information is vital to both forming the hypothesis space and guiding the search; effective use of ILP depends upon this background knowledge. The ILP-setup problem of articulating background knowledge can be difficult and requires detailed understanding of the ILP algorithm, greatly limiting ILP's usability by non-experts.

At least two possible solutions to this problem exist. One is a two-step process in which an ILP expert closely works with a domain expert in the first step to tailor the general-purpose ILP system to a specific domain, such as drug design (e.g., [3]). In the second step, domain experts, who are not ILP experts, can then use the tailored system. A second solution to this ILP-setup problem, which retains the general-purpose nature of the ILP system, is to allow a teacher to, as naturally as technically

possibly, explain *why* specific examples are positive or negative through some advice language. This *teacher-provided advice* supplies hints about the concept being learned, beyond the traditional labeling of examples. Given this teacher-provided advice, the automated learner can generate background knowledge and set appropriate search parameters. This paper is the first study to explore this second approach, although some prior ILP work is related and is reviewed in Section 5.

Consider the following sample dialog between the teacher and the learner. Assume the formula $(p(X) \wedge q(X,Y)) \vee r(X)$[1] is a relevant piece of background knowledge for concept C. The teacher might express this indirectly via the following dialogue about a small number of training examples:

"In example 1, object a is a positive instance of concept C because $p(a)$ is true."

Note that, in human instruction, the teacher might say this to mean simply that p is relevant to C rather than the complete definition of C.

"In example 2, object b is a positive instance of C because $r(b)$ is true."

Note here that an algorithm that induces background knowledge from these statements needs to map both objects a and b to the same variable.

"In example 3, object d is a negative instance of C because $q(d, d)$ is false."

Note that the teacher is telling the learner about relevant background knowledge through a negative example. The piece of advice in this case needs to be negated. In addition, the machine learner does not know whether the advice is about (1) all possible choices for the second (or first) argument of q, (2) restricting the choice of the second argument to be the same as the first, or (3) just the specific choice of constant d as the second (or first) argument.

Although $(p(X) \wedge q(X, Y)) \vee r(X)$ may be the formula necessary to define the concept, formulas such as $(p(X) \vee q(X, Y)) \wedge r(X)$, or $p(X) \wedge q(Y, X) \wedge r(X)$, or $p(X) \wedge (q(X, d) \vee r(X))$, or yet still others are also consistent with this human-provided advice.

Allowing the teacher to provide such advice permits the use of ILP in an unexplored setting in which only a few examples, along with teacher-provided annotations, are sufficient to learn the target concept. However, in a setting with few examples, while the target concept might be complex, such as $(p(X) \wedge q(X, Y)) \vee r(X)$, a simple clause, say $p(X)$, by itself might be sufficient to discriminate between examples. Thus, in this setting the advice should motivate the learner to prefer formulas that use all the teacher-mentioned predicates (i.e., p, q and r), rather than just the simplest formula consistent with the labeled examples.

Below, we present an algorithm to convert teacher-provided advice into ILP background knowledge. We designed this algorithm with the sparse example setting in mind. Motivations for the algorithm we present include the following:

1. High accuracy of the learned concept definition on teacher-labeled training examples.

[1] We use standard Prolog notation for constants and variables throughout this paper, but use standard logical notation otherwise.

2. Robustness in the presence of a small number of training examples and perhaps a total lack of negative examples.
3. Inclusion of most, if not all, teacher-mentioned predicates in the learned concept definition.
4. Flexible combination and generalization of the teacher's advice within and across examples.
5. Robustness to teacher errors, both in data labeling and advice.
6. Learned concepts may need to include predicates not mentioned in teacher-provided advice but supplied as part of example descriptions.

Our primary motivation is to allow human users of ILP systems to express their knowledge about the learning task at hand in whatever means seems most natural to them, from explaining (partially or fully) why some specific examples are positive or negative members of the concept being learned, to simply stating the proper categories (i.e., positive or negative) for other examples. We present our approach as a "batch" system that is given a set of labeled examples and possibly some advice about the examples, and then produces a set of one or more logical clauses ("inference rules") that best capture the concept being taught. However, we envision our approach as being best situated in a setting where the human-machine dialog is continual; the human teacher provides some initial training, the algorithm then learns, after which the teacher can provide additional guidance and the process repeats until an acceptable concept description results (where 'acceptable' can either be based on inspection of the learned clauses, or, more likely, on the quality of the predictions of the learned clauses for new examples).

ILP systems search a space of possible clauses composed from background knowledge; the space is typically defined declaratively by a set of mode definitions. For this work, we used an implementation that closely follows the Aleph ILP system [19], although we present general techniques with few Aleph dependencies. We made one major changes to the default Aleph implementation, wrapping Aleph in an iterative-deepening style search algorithm, as further explained in the next section.

As mentioned above, we address the problem of effectively incorporating, into the ILP framework, teacher-provided advice; a human teacher usually provides the latter and this interaction can be viewed from the wider perspective of human-machine interaction. Such teaching refers to humans teaching computers concepts and/or behaviors, through as *natural* and human-like dialog as possible. In our setting, the taught concepts take the form of logical theories and the teacher provides relevance advice about specific examples. The relevance advice takes a number of different forms, from simple "this feature is important" advice to complex statements that can be mapped to a grounded form of the concept being taught. The advice can be provided by a human familiar with the advice language but with no ILP experience, i.e., a non-expert. In the experiments we report later in this paper, all the instruction was provided by non-ILP-experts, all who are independent from this paper's authors.

Figure 1 illustrates, using propositional logic for simplicity, how advice-generated background knowledge can help focus ILP's search. A common ILP search strategy is to build clauses in a top-down manner, successively adding various literals that might improve a rule. If a long clause is needed, the search space can be

exponentially large, and if there are only a few training examples, many possible rules can accurately match the training examples. However, good background knowledge can quickly lead to the consideration of long clauses, as the figure shows. It can also help choose among many equally performing rules.

Section 3 discusses the conversion of this teacher-provided advice into ILP background knowledge. To accommodate differing amounts of advice, and different levels of concreteness of advice, we employ a control structure, explained in Section 2, which is capable of exploring successive layers of the hypothesis space, from a layer that tightly follows the advice to an outermost, rarely-used layer that effectively ignores the advice. Section 4 presents a discussion of our empirical study. Section 5 discusses the relationship to prior work within ILP.

2 The Onion

Our ILP advice algorithm generates a number of hypotheses, some of which we deem less likely than others. Additionally, since we assume the teacher has little to no ILP expertise, we must utilize a method to select ILP parameters automatically, in a way that consistently supports the motivations discussed above. We developed the *ONION*, a control structure that iteratively searches a set of successively larger and less likely ILP search spaces.

Algorithm 1 presents the basic structure of the ONION. Essentially, the ONION iterates over a set of ILP parameter settings and priority levels, performing an ILP search for each. When a given ILP search returns a theory, the ONION evaluates that theory against a set of acceptance criteria (such as minimum accuracy and coverage, i.e., precision and recall), and if the theory is acceptable, the ONION returns it.

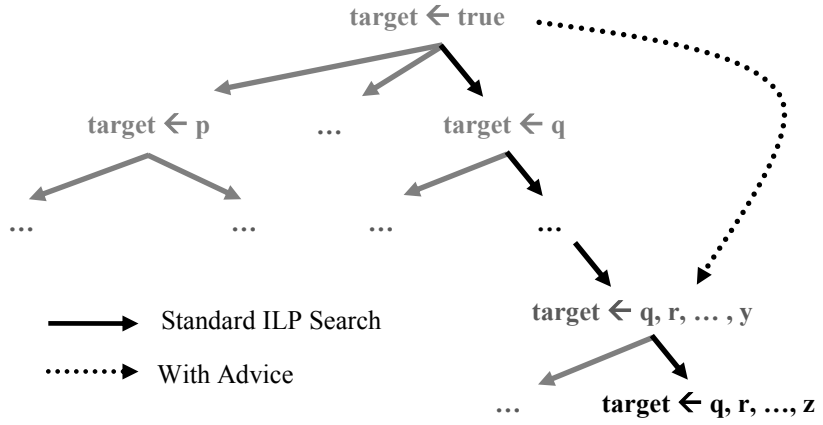

Figure 1. An illustration of a top-down ILP search for a inference rule to predict the literal *predicate*, whose definition is the conjunction of literals *q* through *z*. Finding a long clause such as this can be quite hard, but if a teacher gives advice (possibly across multiple examples) that the conjunction of literals q through y is relevant, then finding the correct definition is much easier.

Algorithm 1: THE ONION

```
1:  For each set of criteria C in {C1, C2, …, Cn} for an acceptable theory
2:    For each priority level P in {High, Medium, Low, None}
3:      For each set of Aleph parameter settings S in {S1, S2, …, Sm}
4:        Only consider literals or generated clauses whose relevance is a least P
5:        Call Aleph with parameter settings S
6:        If Aleph's learned theory meets criteria C then Return theory
7:  Return FAIL
```

We order the sets of acceptance criteria and possible settings from most restrictive to least restrictive, while we order the generated background according to a priority. We discuss how we assign that priority in the next section. The priority level "None" is given to those literals not mentioned by the teacher's advice, but which appear in the descriptions of examples.

3 Converting Advice to Background Knowledge

Teacher advice provides a method for the user to instruct our learning algorithm. The advice takes the form of logical statements. From this information, we construct new background knowledge representing sub-concepts. We also generate the necessary ILP modes (these modes specify the types of arguments and state, for the arguments of a new literal being considered for addition to a clause, which need already appear in the clause, which can be new variables, and which should be constants). Additionally, we attach priorities to all of the generated background knowledge for use by our ONION algorithm. Currently, we assume the teacher talks about a specific example (either positive or negative) and specifies the advice in a ground format that we then variablize into a general form. It is straightforward to extend our system to allow the teacher to provide generalized advice, but we believe that for most users it will be easier to explain why specific examples are or are not members of the concept being taught and that is the interaction style on which we focus.

Although we assume that the user understands basic logic (i.e., the meaning of AND, OR, and NOT), we attempt to allow the user to communicate advice in a natural, and possibly somewhat inaccurate manner. Thus, although we specify the exact logical format of the advice below, our system attempts to rectify common user misunderstandings, such as predicate/function confusion. We also do not expect the user to understand the algorithmic details of the underlying ILP system.

Table 1 shows two training examples and three pieces of teacher provided advice for a sample concept ReadyToFly. The ReadyToFly concept indicates, as one might guess, that an airplane is ready to fly. We will be using this simple concept to demonstrate our approach. We define the concept as:

$$\text{readyToFly(Plane)} \leftarrow \text{fueled(Plane)} \wedge \text{gearDown(Plane)} \wedge \neg\text{damaged(Plane)}.$$

Algorithm 2 details the process of creating background knowledge from the teacher-provided advice. The process proceeds in several phases. The first phase (lines 4 to 10), variablizes the ground advice statements via applying anti-substitution, i.e., a mapping from occurrences of ground terms to variables. For our purposes, we

Table 1. ReadyToFly concept. Training data includes two examples, one positive, one negative, along with three pieces of teacher provided advice, two pieces for the first example and one for the second.

Advice #	Ground Example	Pos/Neg	Teacher Advice
1	readyToFly(plane1)	Positive	fueled(plane1)
2	readyToFly(plane1)	Positive	gear_down(plane1)
3	readyToFly(plane2)	Negative	damaged(plane2)

only need to map constants to variables. The anti-substitution may be either a *direct-mapping* that maps all occurrences of the same constant to the same variable and occurrences of distinct constants to distinct variables, or an *indirect-mapping*, where occurrences of the same constant can be mapped to different variables.

Indirect-mappings address cases where two constants are coincidentally equivalent. This occurs regularly in examples with numeric constants, where common numbers such as 1.0 may perform two different roles. Later, when we assign priorities to generated background knowledge, those created with indirect-mappings receive a lower priority than those created with direct-mappings. Indirect mapping anti-substitutions perform what is sometimes called "variable splitting" in ILP [18], where two occurrences of the same term are generalized to different variables. It is well-known that variable splitting can lead to an increase in run-time that is exponential in the number of occurrences of the same term within a formula. In practice, such multiple occurrences are rare, except in the case of very common constants within a domain, for example, the 1.0 case discussed above. To prevent this exponential worst-case increase, in practice, we limit the maximum number of variable splittings (cases of two occurrences of the same term being mapped to distinct variables) by an anti-substitution to some small constant k. Alternative approaches to controlling the cost of variable splitting, such as employing domain-specific heuristics about commonly-occurring constants, are a direction for future research.

Table 2 depicts both a direct and indirect anti-substitution. As shown, we perform the same anti-substitution on both the example and the piece of advice, linking variables in the example to variables in the advice. Although not shown in Table 2 we generalize all advice for a single example at the same time. Thus, constants can be tied together across different advice for the same example, but are not tied across advice for different examples.

Given the generalizations from the first phase, the second phase of GENERATEBACKGROUNDKNOWLEDGE (lines 12 to 17) performs a unification to merge variables that arose from constants found in the examples themselves. For instance, if we consider the direct anti-substitutions of all pieces of advice, we have *readyToFly(A) ⟵ fueled(A); readyToFly(A) ⟵ gear_down(A);* and *readyToFly(B) ⟵ damaged(B).* After the unifications, the implications would be *readyToFly(X) ⟵ fueled(X); readyToFly(X) ⟵ gear_down(X); and readyToFly(X) ⟵ damaged(X)* where X is shared. This allows distinct constants from different examples that played the same role to be merged into a single variable.

Algorithm 2: GENERATEBACKGROUNDKNOWLEDGE

```
1:  Given: Labeled examples, some of which have associated advice
2:  Do:    Infer generalized background knowledge
3:
4:  For each example e_i ∈ {e_1…e_n}
5:    Given advice A_i associated with example e_i
6:    if e_i is positive example then create an associated implication e_i ⟵ A_i
7:    else create an associated implication e_i ⟵ ¬A_i
8:
9:    Generate all non-equivalent formulas via anti-substitution
10:     from the implication to yield the set of formulas F_i
11:
12: Let F denote the set F_1 ∪ F_2 ∪ … F_n
13: Standardize apart all formulas in F
14: Let θ be the most general unifier of all consequents of formulas in F
15: For each F_i
16:   Apply θ to all formulas in F_i to yield F'_i
17:   Collect all antecedents from formulas in F'_i to yield G_i
18:
19: Let H = {}, a set of generated rule antecedents
20: For each generalized advice-piece G_i
21:   Let H = H ∪ G_i   // Per-piece antecedents
22:
23: For each example e_j ∈ {e_1…e_n} with associated advice
24:   Let K_j = ∪ { g ∈ G | g was generated from example e_j advice}
25:   Let H = H ∪ K_j  // Per-example antecdents
26:
27: Let H = H ∪ { "Mega-Rules" }  // See text
28:
29: For each generated logical combination h ∈ H, introduce a new predicate
30:  p and assert p(V_1, V_2, …, V_k) ⟵ h,  where V_1, V_2, …, V_k are variables
31:  generalized from constants in the example literal
32:   If p ⟵ h is a Mega-Rule then assign p ⟵ h High priority
33:   else if p ⟵ h is per-example then assign p ⟵ h Medium priority
34:   otherwise assign p ⟵ h Low priority
35:
36: Return set of all generated implications p ⟵ h along with priorities
```

In the third phase (lines 19 to 27), we generate compound logical formulas by connecting the generalizations for different pieces of advice with the AND and OR logical connectives. We generate three different styles of formulas: per-piece, per-example, and "Mega Rules". The per-piece formulas correspond to the individual pieces of advice specified by the teacher. The per-example formulas aggregate all the advice provided for a single example into one formula with the individual pieces of advice joined via AND connectives. The per-example formulas allow the teacher to provide many small pieces of advice about an example instead of requiring the teacher to compose a single complex piece of advice. Finally, Mega Rules attempt to

Table 2. Direct and indirect anti-substitutions. Direct anti-substitutions generalize equivalent term to the same variable. Indirect anti-substitutions generalize equivalent terms to different variables.

Ground Example & Advice	Anti-substitution	Type
readyToFly(plane1) ← fueled(plane1)	readyToFly(X) ← fueled(X)	Direct
	readyToFly(X) ← fueled(Y)	Indirect

capture all of the advice into one logical statement, by conjoining direct-mapping generalizations of all advice from all positive and negative examples.

More specifically, we do the following to produce our Mega Rules:

Let F_i be the logical formula that our algorithm produces by conjoining ("ANDing") all of the relevance statements about positive example *i*.[2]

Let G_j be the logical formula that our algorithm produces by conjoining all of the relevance statements about negative example *j*.

We make the following Mega Rules, where *i* ranges over the positive examples with advice and *j* over those negative examples with associated advice:

$$(F_1 \wedge \ldots \wedge F_i) \wedge \neg(G_1 \vee \ldots \vee G_j) \rightarrow example$$
$$(F_1 \wedge \ldots \wedge F_i) \wedge \neg(G_1 \wedge \ldots \wedge G_j) \rightarrow example$$
$$(F_1 \vee \ldots \vee F_i) \wedge \neg(G_1 \vee \ldots \vee G_j) \rightarrow example$$
$$(F_1 \vee \ldots \vee F_i) \wedge \neg(G_1 \wedge \ldots \wedge G_j) \rightarrow example$$

The above are all ways to explain a collection of teacher-provided advice, though some are more natural than the others. In the first one, our algorithm interprets the teacher as using each positive example to provide aspects of a conjunctive concept and each negative example to state properties that members of the concept lack ("this is a bird because it has wings, this other example is a bird because it lays eggs, this third example is not a bird because it has leaves, this fourth example is not a bird because it is made of metal. …"). The third and fourth lines are appropriate for disjunctive concepts ("Alice got to work by taking the bus. Bob got to work by walking. … Carl did not make it to work because he slept all day."). In addition to the rules shown above, we also generate four additional Mega Rules in which we negate positive advice and do not negate the negative advice.

When only direct-mapping generalizations exist, only a handful of formulas are generated, providing excellent scalability. When indirect-mappings occur, we generate additional rules in which we substitute all combinations of the indirectly-mapped advice pieces into the per-piece and per-example formulas. This process scales exponentially in the number of indirect-mapping generated.

In the final phase (lines 29 to 34) we convert each of the generated formulas back into an implication (as a precursor to creating ILP background knowledge). During

[2] We conjoin statements about the same example because we assume the teacher is telling us various properties of that example that all hold, though it would be reasonable to extend our algorithm by disjunctively combining them as another alternate interpretation. By allowing only one or two ways to combine advice about the same example, we avoid the combinatorics that would arise by combining in all logically possible ways.

Table 3. Generated Background Knowledge. Three types of background knowledge are created during advice processing: per-piece, per-example, and mega-rule background knowledge. Per-piece is composed of single pieces of advice. Per-example is composed of all advice piece for a single example. Mega-rules use all provided advice combined via various logical operators.

Generated Background Knowledge	Type	Priority
readyToFly(X) ← fueled(X) ∧ gear_down(X) ∧ ¬damaged(X)	Mega-Rule	High
readyToFly(X) ← (fueled(X) ∨ gear_down(X)) ∧ ¬damaged(X)	Mega-Rule	High
readyToFly(X) ← fueled(X) ∧ gear_down(X)	Per-Example	Medium
readyToFly(X) ← ¬damaged(X)	Per-Example	Medium
readyToFly(X) ← fueled(X)	Per-Piece	Low
readyToFly(X) ← gear(X, down)	Per-Piece	Low

this phase, we assign a search priority with a preference for longer formulas, i.e., those that use as much of the user-provided advice as possible. Mega Rules receive the highest priority, followed by per-example formulas, and finally per-piece formulas. The ONION uses these priorities to order the ILP search.

Table 3 shows several of the implications generated for the sample concept. The head of the generated clause exposes all of the variables from the logical formula. Variables tied to the example during the generalization phase become input variables for the clause. In many situations, it is also advantageous to expose variables occurring in the body of the rule as output variables. However, exposing additional variables increases the size of the ILP search space. In absence of other information, for each formula, we expose only a single output variable. For any given formula, we determine the output variable by considering all of the literals that we derived from positive pieces of advice and selecting the last variable that occurs. This approach scales well. However, in some cases, variables that would be helpful may not be exposed in the head of the generate clause. Clauses derived from formulas with OR connectives have the additional requirement that the selected output variable must occur in all of the OR-ed subformulas. Determining whether a variable occurs in all of the subformulas requires us to determine if two variables are equivalent. If argument-type information is available, we require only that type of the output variable match in all of the subformulas. In the absence of typing information, we disallow output variables for disjunctive formulas.

Statements about negative examples can be ambiguous. Imagine a teacher says an example is negative because *color* = *blue*. Does this mean the example is negative because it is blue or because it is not? Because the teacher is talking about specific examples that are observable by our learning algorithm, we address this in an obvious way. Namely, we evaluate the teacher's statement on the current example, and we then, if necessary negate the advice so that it says something that is true. Hence if the current example is red, we standardize the advice about color to $color \neq blue$.

Finally, we assign input variables both an ILP input mode of '+' (the argument must already be in the clause being constructed) and a constant mode of '#', plus we

assign output variables both an output mode of '–' (a new variable can be introduced) and a constant mode of '#'. Additionally, our algorithm also works when the ground advice contains logical functions. We convert the functions into Skolem-constants and perform generalizations over all possible combinations of the Skolem-constants.

4 Experiments

We performed several experiments to demonstrate the performance of the ONION with and without advice. In addition to measuring learning on our test beds, we also conducted comprehensive empirical analyses to study the performance of the ONION when there is noise in the labels on examples, as well as in the advice. These experiments are designed to demonstrate the effectiveness of the ONION in the absence of advice, its robustness to noise, and how the system is capable of generalizing advice about specific examples to all the available examples leading to improved learning and accuracy. The improvements in generalization performance can be significant, especially in the presence of a very small number of examples.

4.1 Test Beds

We used learning tasks developed by an independent third party under the Bootstrap Learning (BL) project [12] funded by the United States Defense Advanced Research Projects Agency (DARPA). In the BL setting, the machine learner induces concepts that build upon one another through a "ladder" of tasks, which are organized as self-contained *lessons*; lower rungs of the lesson ladder teach simpler concepts, which are learned first and then used to learn − i.e., *bootstrap* − more complex concepts. The lessons in the project incorporate a wide variety of natural teacher instruction methods, including providing domain descriptions, pedagogical examples, telling of general instructions, demonstration, and feedback. Our role in the project is supervised learning from examples. For our experimental setup considered here, we use 14 lessons from two domains of the BL project: Unmanned Aerial Vehicle (UAV) and Armored Task Force (ATF).

For each of the 14 lessons, third-party domain-experts, under the direction of DARPA and not under our control, generated "lessons" to teach these tasks. The lessons consist of a sequence of *messages* from the teacher to the learner. Teacher instruction includes providing *training examples* (up to 100 examples for each task) and expert advice for certain examples to help the student learn these tasks effectively. For each lesson, we wrote software that converted the messages into ILP facts and examples expressed in predicate calculus. The mean accuracy of always guessing the majority category across each of these 14 lessons is 57%.While we had access to the UAV and ATF testbeds during algorithm development, during Fall 2009 our algorithm was applied by DARPA to a "hidden" testbed, to which we had no access. Our approached produced 100% accuracy on learning in that testbed. Since a variant of that testbed will be used Spring 2011, at the time of this writing we have no knowledge of the testbed and, hence, cannot report anything more about it here.

Methodology. We are interested in studying the behavior of our advice algorithm, along with the ONION, with respect to several criteria: (1) its ability to learn diverse concepts across domains without the intervention of an ILP expert, (2) its ability to effectively exploit teacher advice in order to learn concepts with only a small number of examples, (3) its robustness to teacher errors of commission in the examples (mislabeled examples) and (4) its robustness to teacher errors of omission in the advice (incomplete or missing literals). Our experimental study consists of three experiments that we describe below. For each lesson we have 100 training examples and 100 test set examples. During our experiments, we split the training set to generate a tuning set, used by the ONION for evaluating parameter settings. For runs with more than 25 examples, we place two-thirds of the data provided to our learner into a training set and one-third the data into a tuning set. For runs where fewer than 25 training examples, we do not use a separate tuning set, instead relying directly on training-set accuracy to tune parameters in the ONION. In all experiments there were an equal number of positive and negative examples.

Because there is an intended pedagogical order to the examples, some of which have associated advice, we did not perform 10-fold cross validation within each lesson (in addition, since we have data simulators, cross validation is not necessary – instead we simply use fresh samples of 100 examples as our test sets). The results presented for each experiment are the test-set accuracies averaged over all 14 tasks.

Across all of these tasks, we used the same parameter choices in the ONION. That is, over all of the experiments that we report here, our ILP system was run *unchanged*. Our ONION approach was able to find good parameter settings, trading off computer time for the ability to operate without intervention from an ILP expert.

4.2 Results and Discussion

Experiment A. In our first experiment, we compare the performance of the ONION with and without advice over all the 14 tasks. Figure 3 shows the results, where we plot *learning curves*, i.e., test-set accuracy as a function of increasing numbers of training examples. (As mentioned earlier, our implementation is not an on-line, incremental learner. We simply run in "batch mode" for various numbers of training examples.)

In the case where the learner is not given any advice, the ONION is able to generalize across tasks and domains, and obtain an average test-set accuracy of 74.0% when using all 100 training examples. Even when using smaller fractions of training data, the ONION is able to effectively select parameters and automate the setup task to obtain learning rates in excess of 57%, which is equivalent to random guessing. The main results in Figure 3 however, are the test-set accuracies achieved by the ONION in the presence of advice. Even when using only four training examples per lesson, the ONION with advice achieves an average test-set accuracy of 93.8%, and reaches 100% with only ten examples.

Experiment B. Experiment A involved advice from a 3rd party who was careful to create rich and accurate advice. However, real teachers are likely to make errors. In

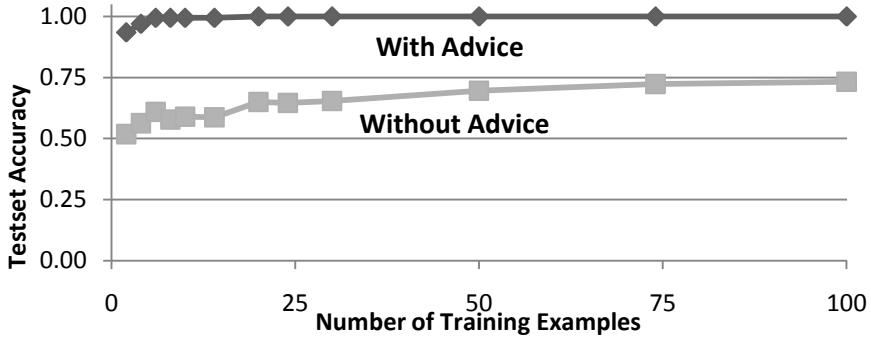

Figure 3. Experiment A: Testset accuracy as a function of the number of training examples ("learning curves"), with and without advice.

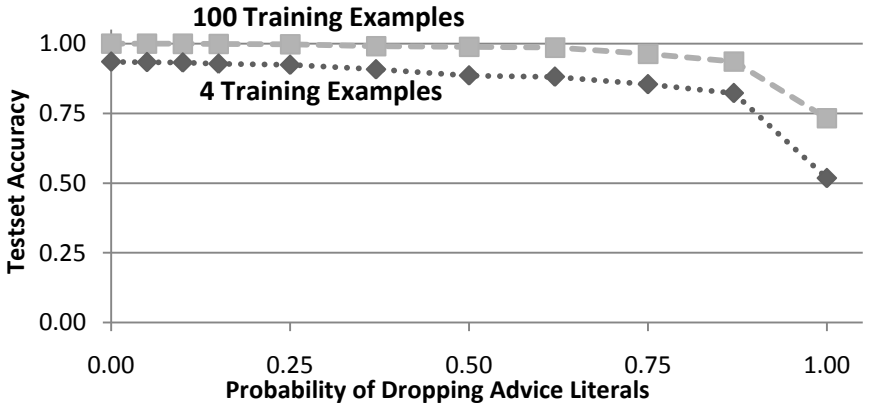

Figure 2. **Experiment B:** Impact of errors of omission in advice. The x axis indicates the probability value used in the advice-removal process (see text).

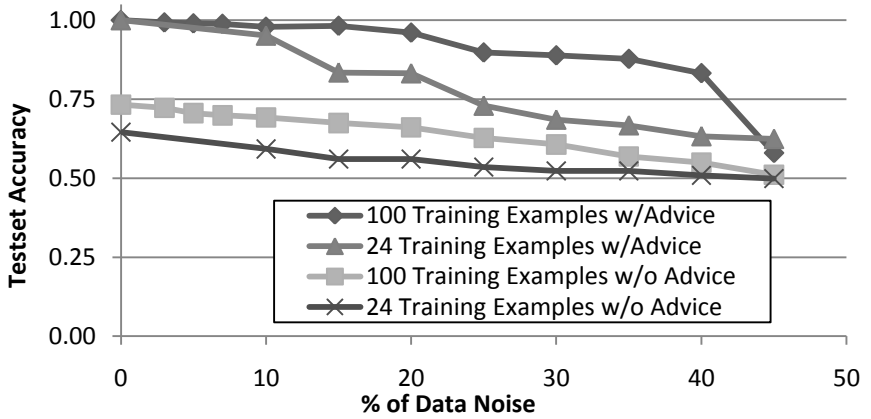

Figure 4. Experiment C: The impact of mislabeled examples under various conditions. The *x*-axis indicates the percentage of training data that is mislabeled.

our 2nd experiment we simulate *errors of omission* by dropping literals from advice. We randomly drop literals as follows. For each advice rule we flip a weighted coin, and it if comes up 'heads' we delete the *last* literal in the rule. If we deleted the last literal, we flip the coin again and consider deleting the second-from-last literal; this continues until either the rule's literals are exhausted or the coin comes up 'tails.' In the later case, we place the possibly truncated advice rule in our "noisy" advice set. We choose to remove from the end of advice rules, since prefixes of conjunctive rules are likely to be partially coherent, whereas dropping literals from the middle of multi-literal (i.e., conjunctive) statements may lead to nonsensical advice. A topic for future research is to create more realistic models of imperfect advice.

Figure 2 shows the results of our errors-of-omission experiment, where we plot the test-set accuracy as a function of the probability of randomly removing each literal as specified above. For each selected probability-of-removing, we generated 30 independent "noisy" advice sets for each of our 14 lessons. The impact of noisy advice depends on the number of training examples, so we perform this experiment using 2 and 100 training examples.

The behavior of the system with different training set sizes is nearly identical with the test-set accuracy dropping steadily as increasing fractions of advice literals are removed. However, with advice-omission rates as high as 50%, and with a small number of examples, the ONION is able to produce average test-set accuracies of over 80%. This demonstrates that even partial advice can be effective for learning, and that the ONION is able to leverage this information effectively even in the presence of significant imperfections in advice.

Experiment C. In this final experiment, we compare the performance of the ONION with and without advice in the presence of mislabeled training examples. Figure 4 shows the results; we plot test-set accuracy as a function of the percentage of mislabeled examples. We generate the noisy examples by first removing examples that have advice attached from the set of training examples. With the remaining training examples, we randomly select a fraction of the examples and flip their labels. We then replace the examples with attached advice into the training set. Care was taken to guarantee that the final fraction of mislabeled examples was correct when the examples with advice added back into the training set. This approach to noise generation limited the range of noise available, especially for small training sets. For instance, if we have a training set size of four and two of those were examples with advice attached, the minimum amount of noise that can be considered is 25% (the result of flipping a single, non-advice, example).

For experiment C, we generate 30 independent sets of mislabeled examples and, separately, ran with and without advice using each of these noisy data sets. The results are averaged over all random 30 runs and over all 14 tasks. As expected, the example noise reduces the performance of both the advice-free and with-advice cases. The main result from Figure 4 is that our advice algorithm, combined with the ONION, performs well even in the presence of large amounts of data noise. In contrast, the no-advice case degrades more quickly to about the level of random guessing (57%).

In summary, our experiments demonstrate that our advice-taking ILP system can learn well from advice about a small number of specific examples, while being robust

to errors in advice and example labels. They also show that our ONION approach can be an effective method even when users provide no advice.

5 Related Work

ILP research has a rich history of developing systems capable of initiating human-computer interaction and using them guide and constrain the search. The most notable such systems include MARVIN [15], MIS [16], DUCE [10], CIGOL [9] and CLINT [2] where the algorithm can ask the human one or more questions that would guide the search. For example MIS relied on the human answering queries by providing the labels of examples, together with a proof, or derivation, for each positive response. In contrast, in our present work the human initiates the input by providing advice, either in general or in association with the original training data.

Another general area of related work is theory refinement or theory revision (e.g., [7, 13, 19, 20]) where the user provides an initial logical theory that explains many and not all examples, and the learning system must modify this theory. As a result the search is constrained to prefer theories close to the original theory, similar to the present work. But a key distinction is that the advice in our present work is *example-specific*, which can substantially ease the burden on the user, as compared to expressing abstract rule(s) underlying the concept.

Our work is closely related to argument-based machine learning (ABML [8]) that takes as input user-provided advice about specific examples, in the form of an *argument*. A key distinction is that the present work does not assume the arguments are exactly correct and therefore may combine various pieces of different arguments in order to construct rules. Another distinction is that to our knowledge ABML has been applied strictly to propositional-rule learning.

Automatic parameter selection for machine learning methods has been explored earlier [5], where the goal is to use the expected error for each parameter setting to guide the selection of the parameters for decision trees. Lavrac et al. [6] proposed a feature selection framework for ILP that worked well in propositional learning and special cases of relational learning. This was later extended by Alphonse and Matwin [1] using powerful statistical feature-selection techniques to control the dimensionality of the search space. The key idea in their work is to reduce ILP examples to non-recursive Datalog clauses by removing irrelevant literals. Muggleton [11] theoretically shows that as the number of predicates in the background theory increases, the size of the search space of an ILP system can increase greatly. This necessitates the intervention of an ILP expert who can reduce the search space. In such a context, relevance information becomes crucial. Srinivasan et al. [17] conducted an empirical study in several biomedical domains and concluded that when not all the background knowledge can be used, the relevance information from the expert is very useful in construction of good domain models.

6 Conclusions and Future Work

Not surprisingly, teacher advice is useful to learning. The key challenge is the need to generalize hints and advice the teacher gives about specific examples so that it

accurately applies to future examples. We present a formal approach to incorporating this into the wider framework of ILP. The empirical results show that our system is able to learn well, across multiple concepts, from a combination of training examples and teacher-provided hints. Running our ILP system without these hints − i.e., only using the training examples – also produces reasonable accuracies on held-out ("test set") examples. Another key challenge is effective parameter selection and the automation of the ILP-setup task. The final challenge is to ensure that the system is robust to noise, both in examples and in advice.

We evaluated our algorithms, holding all default parameter settings constant, on 14 tasks from two domains designed by third-parties not under our control; these human teachers provided training examples as well as relevance information. In our experiments, we demonstrated that our system, the advice-taking ONION, is capable of (1) effectively automating the ILP-setup task over different tasks from significantly different domains, (2) exploiting teacher hints and relevance information to learn concepts with near-perfect test-set accuracies even if given only a small set of training examples, (3) performing effective parameter selection to learn concepts well when there is no teacher advice, (4) being robust to example-label noise that can arise from teachers' errors of commission, and (5) being robust to advice noise that is likely to arise from teachers' errors of omission.

Currently, we are focusing on improving our layered approach, to more robustly automate ILP in tasks that are more complicated. Also, we are currently looking at further exploiting teacher-provided feedback beyond statements about which features and objects are relevant, such as allowing teachers to provide corrections to previous advice statements. A possible future direction is to explore the possibility of refining the learned theories using teacher feedback in the lines of theory refinement for ILP [8, 11, 12, 13, 14]. Refining teacher's advice is important as it renders the ILP systems more robust to teacher errors that occur naturally in human teaching. Another future direction is deploying our approach in the context of probabilistic-logic learning [4]. A final appealing direction of this research is to embed it into some user interface where a human can train their software by a combination of making simple English statements, pulling down menus and selecting items, and gesturing at objects (e.g., clicking with the mouse, a pen, or even one's finger) to indicate relevance and objects of discourse ("this object should not be near that one").

In this work, we considered the problem of simplifying the use of ILP for non-ILP experts. More precisely, we considered a human teacher who is trying to teach an ILP learner using a mixture of examples and advice about these examples. We outlined the challenges and presented solutions for the generation of background knowledge from teacher advice, utilizing a layered ILP-search approach.

7 Acknowledgements

The authors gratefully acknowledge the support of the DARPA's Bootstrap Learning program via the United States Air Force Research Laboratory (AFRL) under grant HR0011-07-C-0060. Views and conclusions contained in this document are those of the authors and do not necessarily represent the official opinion or policies, either expressed or implied, of the US government, DARPA, or AFRL

8 References

1. Alphonse, E., and Matwin, S. 2002. Feature subset selection and inductive logic programming. In *Proceedings of the 19th Intl. Conf. on Machine Learning*, 11–18.
2. De Raedt, L. 1992. *Interactive Theory Revision: An Inductive Logic Programming Approach*. Academic Press.
3. Finn, P., Muggleton, S., Page, D., and Srinivasan, A. 1998. Discovery of pharmacophores using the inductive logic programming system Progol. *Machine Learning*, 30:241–270.
4. Getoor, L., and Taskar, B. (eds.) 2007. *Introduction to Statistical Relational Learning*. MIT Press.
5. Kohavi, R., and John, G. 1995. Automatic parameter selection by minimizing estimated error. In *Proceedings of the 12th International Conf. on Machine Learning*, 304–312.
6. Lavrac, N., Gamberger, D., and Jovanosk, V. 1999. A study of relevance for learning in deductive databases. *Journal of Logic Programming*, 40:215–249.
7. Mangasarian, O., Shavlik, J., and Wild, E. 2004. Knowledge-based kernel approximation. *Journal of Machine Learning Research*, 5:1127–1141.
8. Mozina, M., Zabkar, J., and Bratko, I. 2007. Argument based machine learning. *Artificial Intelligence*, 171: 922-937.
9. Muggleton, S., and Buntine, W. 1988. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the 5th Intl. Conf. on Machine Learning*, 339–352.
10. Muggleton, S. 1987. DUCE, an oracle based approach to constructive induction. In *Proceedings of the International Joint Conf. on Artificial Intelligence*, 287–292.
11. Muggleton, S. 1995. Inverse entailment and Progol. *New Generation Comp.,* 13:245–286.
12. Oblinger, D. 2006. Bootstrap learning - external materials. *http://www.sainc.com/bl-extmat.*
13. Pazzani, M., and Kibler, D. 1992. The utility of knowledge in inductive learning. *Machine Learning*, 9:57–94.
14. .Richards, B., and Mooney, R. 1995. Automated refinement of first-order Horn-clause domain theories. *Machine Learning*, 19: 95–131.
15. Sammut, C. 1981. *Learning Concepts by Performing Experiments*. Ph.D. Dissertation, Department of Computer Science, University of New South Wales.
16. Shapiro, E. Y. 1983. *Algorithmic Program Debugging*. MIT Press.
17. Srinivasan, A., King, R. D., and Bain, M. E. 2003. An empirical study of the use of relevance information in inductive logic programming. *JMLR* 4:369–383.
18. Srinivasan, A., Muggleton, S., and King, R. 1995. Comparing the use of background knowledge by inductive logic programming systems. In *Proc. 5th ILP Workshop.*
19. Srinivasan, A. *The Aleph Manual. http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/aleph.html.*
20. Towell, G., and Shavlik, J. 1994. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70:119–165.
21. Walker, T. 2011 (forthcoming). *Broadening the Applicability of Relational Learning*. Ph.D. Dissertation, Computer Sciences Department, University of Wisconsin – Madison.