Integrating Knowledge Capture and Supervised Learning through a Human-Computer Interface

Trevor Walker, Gautam Kunapuli, Noah Larsen, David Page, Jude Shavlik
University of Wisconsin - Madison
Madison, WI, USA
{twalker, kunapg, larsen, page, shavlik} @ biostat.wisc.edu

ABSTRACT

Some supervised-learning algorithms can make effective use of domain knowledge in addition to the input-output pairs commonly used in machine learning. However, formulating this additional information often requires an in-depth understanding of the specific knowledge representation used by a given learning algorithm. The requirement to use a formal knowledge-representation language means that most domain experts will not be able to articulate their expertise, even when a learning algorithm is capable of exploiting such valuable information. We investigate a method to ease this knowledge acquisition through the use of a graphical, human-computer interface. Our interface allows users to easily provide advice about specific examples, rather than requiring them to provide general rules; we leave the task of properly generalizing such advice to the learning algorithms. We demonstrate the effectiveness of our approach using the Wargus real-time strategy game, comparing learning with no advice to learning with concrete advice provided through our interface, as well as comparing to using generalized advice written by an AI expert. Our results show that our approach of combining a GUI-based advice language with an advice-taking learning algorithm is an effective way to capture domain knowledge.

Categories and Subject Descriptors

I.2.6 Learning – *Knowledge acquisition.*

General Terms

Algorithms, Experimentation, Human Factors.

Keywords

Advice Taking, Human-Computer Interface.

INTRODUCTION

Many domains exist in which experts possess extensive knowledge and know how to apply that knowledge to perform domain tasks; cardiologists determine the likelihood of heart disease given diagnostic test results; analysts viewing surveillance imagery identify suspicious activity; and coaches determine the strategy of a game by examining past games of an opposing team. While traditional supervised-learning algorithms use input-output pairs, often referred to as positive and negative examples in the case of two-class learning problems, some algorithms can also use domain knowledge during learning. For instance, various inductive logic programming algorithms [12] accept background knowledge in the form of Horn clauses, knowledge-based support vector machines [5] use knowledge in the form of constraints over regions of a task's feature space, and Markov logic networks [15] accept knowledge in the form of weighted first-order logic.

Articulating domain knowledge for any given algorithm requires an in-depth understanding of the specific knowledge representation used by the algorithm. However, formulating domain knowledge in the correct representation either requires training a domain expert to provide knowledge in the necessary formal representation or requires the domain expert to rely on a third party to translate the domain knowledge. This bottleneck greatly limits the applicability of these algorithms.

One approach to overcoming this limitation is to provide the domain expert a *human-computer interface* (HCI) that facilitates the acquisition of the domain knowledge in a manner easily understood by the domain expert, but which constrains the knowledge such that, through some algorithmic transformation, it is also useable by the learning algorithm. Extensive research exists studying HCIs for this purpose. Some approaches rely on demonstration by the domain expert of some process [2,10]. Others provide an interface in which the expert may specify additional examples to guide or correct the learning algorithm [17]. Some treat domain knowledge as a form of constraints and provide an interface to specify those constraints [8].

We investigate a method of using an HCI to obtain relational domain knowledge, in the form of concrete (i.e., ground) logical *advice* (i.e., domain knowledge that may be incomplete or incorrect, but may still be useful) about specific training examples, leaving the task of generalizing the domain knowledge to an automatic algorithm. Relational domains are characterized as domains with objects and relationships among them (expressed through predicates).

K-CAP'11, June 25-29, 2011, Banff, Alberta, Canada.

Unlike the so-called *fixed-length feature vectors* of standard machine learning, examples in relational domains often contain a variable number of objects and relationships.

By focusing on ground statements about specific examples, we provide the domain expert a method to simply state why something is true (or false, depending on the example), without requiring them to understand the final knowledge representation. Additionally, by supporting relational advice, we allow richer knowledge and better support relational-learning algorithms. Our HCI approach to obtaining relational knowledge in a ground format and automatically processing it into the required knowledge representation is, we believe, unique about this research.

We demonstrate our approach's effectiveness by examining a task in a real-time strategy game. We providing a simple GUI through which a domain expert can specify relational advice explaining various scenarios and show that combining the HCI and a suitable advice-taking learning algorithm is effective. We compare successfully against both (a) using no advice and (b) hand-written advice.

OVERVIEW OF APPROACH

We consider a learning paradigm designed to assist domain experts (we will also refer to them as *users*) in the process of creating and refining domain knowledge through the use of an HCI. We view domain knowledge as a form of advice provided by the user to the learning algorithm. We also consider advice acquisition to be an iterative process (see Figure 1) of a user specifying advice, an algorithm learning a model, a user reviewing results, and a user refining or augmenting the advice. Although this paradigm applies to many forms of learning, we specifically consider supervised learning algorithms that take as inputs both training examples and additional domain knowledge.

Our iterative learning process proceeds in four stages. First, we present an HCI through which the user specifies advice. Our HCI accomplishes this by displaying information about a single training example and asking the user why that example was positive (or negative). We specifically consider advice in the form of concrete logical statements. For instance, in a medical domain, the HCI might provide the user with a patient's information, health history, etc. and ask why that patient was high risk for heart problems. The user might have the domain knowledge that the patient was a high risk because "the patient's cholesterol was high during her last visit and her father's family has a history of heart disease." Through the HCI, the user would be able to express this knowledge in the ground logical format without knowing the final representation.

Although the domain expert does need to understand basic logic (that is *and*, *or*, and *not*), beyond that they need only makes statements about why a particular example is true or false. We believe that domain experts can specify this ground representation of advice more easily than other representations. Additionally, relying on ground advice reduces the complexity of the HCI since it does not need to support logical variables in the advice.

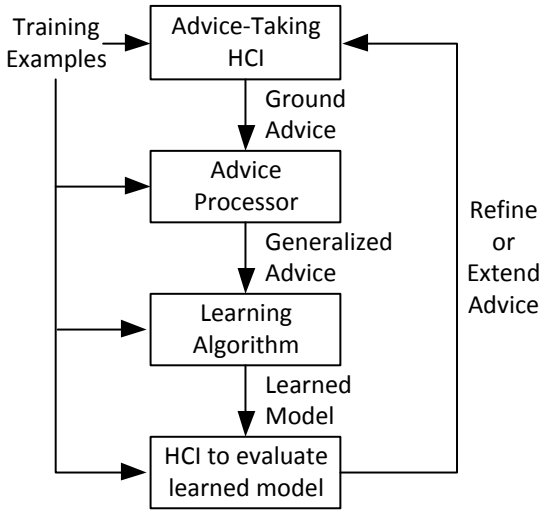

Figure 1. Our human-computer learning paradigm. Initially the user specifies advice through an HCI. Then the advice is processed and learning occurs. Afterward the results are presented to the user via an evaluation HCI. The process iterates until the user is satisfied with the results.

After the user finishes entering advice for a number of training examples, the second phase of our process translates the advice into a form usable by the learning algorithm. This usually entails generalizing the advice and possibly changing the representation of the advice. The process of converting the advice can be quite complex. Often there are multiple generalizations with distinct meanings. Since we do not expect the user to understand the underlying representation of the advice, it is often infeasible to ask the user to correct the advice directly and we rely upon our iterative process to indirectly improve the advice. Additionally, the user might specify advice over multiple examples and the algorithm must determine the meaning of the multiple statements.

Phase three of our process performs the actual learning. At this point, we provide to the learning algorithm both the training examples and the generalized advice. The learning algorithm then produces a *model*. After learning, we pass this model onto phase four. Here, we evaluate the model against additional examples and present an HCI allowing the user to review the effectiveness of the model.

The reviewing HCI presents the user information about which 'testset' examples were correctly or incorrectly predicted by the model. Based on this, the user may elect to return to phase one in order to adjust previously presented advice, provide new advice, or label new examples.

Below we present further details about the first three phases of this learning process. While we believe that the iterative process of refining advice is an important step, we do not investigate it further here.

Table 1. Features describing tower-defense world.

Category	Values
Units	archer, swordsman, ballista, peasant, tower
Unit properties	x-location(Unit), y-location(Unit), health(Unit)
Group Properties	unitInGroup(Group, Unit), groupSize(Group)
World Properties	countOf(UnitType), moatExists, contentsOfTile(X,Y)

BACKGROUND AND RELATED WORK

Extensive research exists studying domain expert knowledge acquisition through the use human-computer interaction [16]. Additionally, previous research examines how to exploit domain knowledge obtained either through an HCI, generated algorithmically or by hand.

One method used to obtain knowledge is *programming by demonstration*. In programming by demonstration, a domain expert performs a sequence of actions demonstrating how to perform some task. From this demonstration, a learning algorithm builds a procedural program intended to solve the task. Often, the learned programs must be adjusted through further interaction with the user. One such system [10] allows the user "nudge" the system through the inclusion or removal of training examples. Another approach [2] allows users to adjust the training data directly, adding missing information after the demonstration process. Unlike our approach, both of these system work directly with the training examples without providing explicit background knowledge. Some approaches do allow explicit background knowledge to be specified. For instance, Vander Zanden and Myers [17] provide a method to specify background knowledge, but require understanding of the underlying knowledge representation represented in Lisp. Another method of human-computer interaction is *programming by example*. Here the user provides a prototypical example of the desired result, such as the result from a database query. These approaches [4,9] again differ from our approach in that they operate on the examples not on additional background knowledge.

Wargus Real-Time Strategy Game

We use the Wargus video game to illustrate our advice-acquisition HCI. In Wargus, a game in the real-time strategy genera, two or more players direct units, such as peasants, swordsmen, archers, etc., in an attempt to conquer the opposing players. Play involves constructing buildings, producing unit, harvesting resources, and directing attacks against opponents.

To demonstrate our HCI we use a subset of the Wargus game we call *tower-defense*. Here an attacking team, consisting of peasants, archers, swordsmen, and ballista, assault a single tower belonging to the defenders. The learning task we consider consists of predicting whether the tower will survive the attack given the size and composition of the attacking force. Figure 2 depicts a typical tower-defense game board. Variations of the game board include the existence of a moat, the size and composition of the attacking units, and the layout of the game board. Table 1 provides a brief description of the features in the tower-defense domain. Additionally, we define a number of background predicates, listed in Table 2, which will be available to the user to specify domain knowledge and may be used in the learned models.

Table 2. Background predicates available in tower-defense predication task.

Category	Predicates
Numeric Comparators	$>$, $<$, $\geq$, $\leq$, $=$
Spatial Comparators	isNearTo, isFarFrom, canReach

An open-source Wargus game engine exists [19] that allows us to simulate the outcome of any game configuration (i.e., will the tower stand or fall?) according to the game's rules. Thus, for any given game board, we can determine the example label as either *towerStands* or *towerFalls*. This also allows us to generate as many examples as desired in our experiments.

Boosted Relational Dependency Networks

We have chosen one advice-taking algorithm to evaluate our approach. *Boosted relational dependency networks* (bRDNs) [13] provide a relational, probabilistic graphical-model based learning algorithm. A *dependency network* [7] approximates a joint distribution over the variables as product of conditional distributions. *Relational dependency networks* (RDN) [14] extend these dependency networks to a relational setting. The bRDN algorithm combines relational dependency networks with a form of gradient-tree boosting [3].

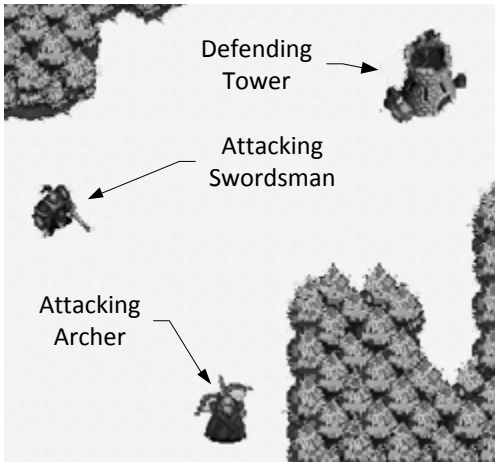

Figure 2. The Wargus tower-defense task. Multiple attacking units, consisting of swordsmen, archers, and ballista, assault the defender's tower. Depending on the composition of the attacking force, the tower may survive or be destroyed. The Wargus tower-defense learning task involves predicting which of these outcomes will occur.

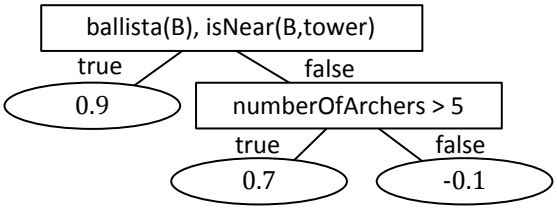

Figure 3. A logical decision tree representing a conditional probability distribution for determining the probability a given tower falls. Each interior node is a logical decision point, with the left branch representing a true evaluation and the right branch a false evaluation. Leaves represent output potentials that must be normalized (see Equation 1) to produce the output probability.

RDNs consist of a set of predicate symbols composing the nodes of a graphical model. For each predicate Y_i, a conditional probability distribution $P(Y_i | X_i)$, defines a distribution over the values of Y_i given the values of the other features. The distribution of a variable y_i is estimated as

$$Prob(y_i|x_i) = \frac{e^{\psi(y_i;x_i)}}{\sum_{y'} e^{\psi(y';x_i)}} \forall x_i \in \boldsymbol{x}_i \neq y_i \qquad (1)$$

where $\psi(y_i; \boldsymbol{x}_i)$ is the potential function of y_i given all other features $x_i \neq y_i$.

The bRDN algorithm approximates these conditional probability distributions (the Ψ's) through *relational decision trees* [1]; a sample is depicted in Figure 3 (in this figure and elsewhere in this article, upper case arguments are logical variables, following Prolog notation). In these relational decision trees each interior node is a logical decision point and the leaf nodes represent the various potentials, i.e., the ψ's, of the conditional probability distribution. In order to obtain the final probabilities, the potentials must be normalized according to Equation (1).

While the conditional probability distributions in RDNs can be represented by a single relational decision tree [6], in bRDNs each conditional probability distributions is estimated by a sequence of trees, based upon an initial potential ψ_0 and iteratively adjusted via a set of gradients Δ_i. Thus, after m iterations, the potential is given as $\psi_m = \psi_0 + \Delta_0 + \cdots + \Delta_m$. Here, Δ_i is given by $\Delta_m = \eta_m \cdot E_{x,y}[\partial / \partial \psi_{m-1} log P(y|x; \psi_{m-1})]$ where η_m is a scalar controlling the gradient step size. Thus, a set of trees are learned for every predicate such that at each iteration a new set of regression trees estimates the maximum likelihood of the distributions with respect to the potential function.

By providing advice, users indirectly produce new background knowledge rules for use by the bRDN algorithm as interior nodes in its trees, as is further explained later.

OUR ADVICE-TAKING HCI

Next we provide details of the requirements of an advice-taking HCI designed to obtain ground advice from the domain expert. Design of the HCI depends greatly upon the type of the knowledge being gathered. Figure 4 depicts a simple prototypical GUI designed for the Wargus tower-defense game and used to gather relational advice pertaining to it. In our approach, the user provides advice through the following process:

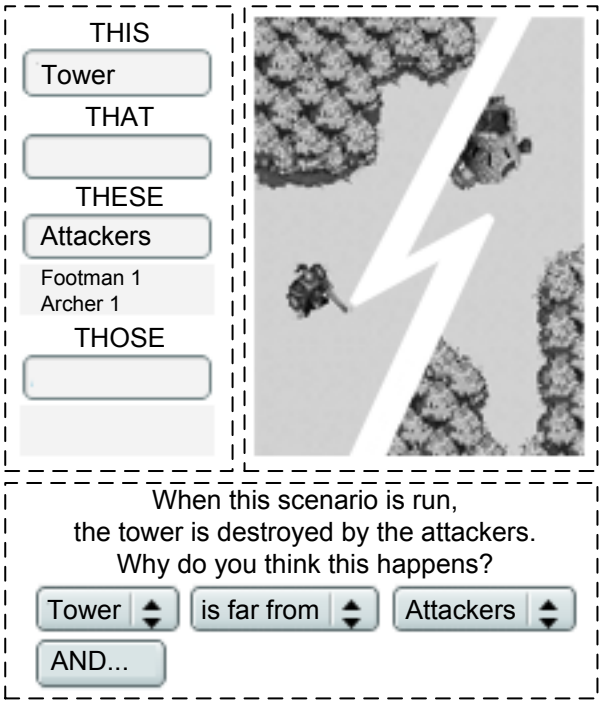

Figure 4. Prototype GUI for advice taking in Wargus. The GUI consists of 4 sections indicated here (but not in the actual GUI) by overlaid dashed boxes: (upper-left) entity selection and naming, (upper-right) display of current game board, (lower) controls specifying relations between selected entities, and (not shown) a list of previously specified advice. Layout and relative size of elements have been adjusted for clarity. Only a partial game board is shown in the upper right.

1. An example is selected, either manually by the user or through some automatic process.
2. The HCI provides the user (ideally visually) information about the example.
3. The user provides advice through the HCI to explain why this example was either positive or negative.
4. The user either returns to Step 1 to provide more advice or stops.

In order to facilitate this process, our HCI needs to:

1. Provide information about specific examples.
2. Allow selection and naming of entities or groups of entities.
3. Provide a method to indicate relations among selected entities.
4. Allow the user to review, and possibly edit, previously specified advice.

Providing Example Information

The HCI must provide information to the user about examples. In our knowledge capture approach, each advice statement is about a specific example. Thus, the HCI needs

to provide at least the information about one example. In the Wargus tower-defense game we can depict all information about an example as a picture of the game board. In some domains, this is not possible or not desirable. For example, examining patient information in a medical domain would require a much different GUI design.

Selecting and Naming Entities

Advice in our system consists of relations among entities. Here, an entity is any object in the domain. For instance, in the Wargus domain, the various units on the game board are entities. A user needs a method to indicate the entities that should be related. Thus, we require some sort of entity-selection mechanism.

The entity-selection methodology will different depending on the domain. In board game domains, such as Wargus, simply clicking on one or more entities is sufficient. However, for other domains, a much more complex approach may be necessary.

One extension we found particularly useful in our Wargus GUI was the ability to name *groups* of entities. When selecting entities in Wargus, by default we named single entities either THIS or THAT. When selecting groups of entities, the default names are THESE or THOSE. Thus, later on, when specifying a particular relation, we could state THIS is related to THAT or THESE are related to THAT.

In addition to selecting entities, in many domains, objects have additional properties. For instance, in the Wargus domain, all units have a property indicating each unit's *x-y* location. Thus, a mechanism is required to access the properties.

Specifying Relations

In relational logic, we specify relations through predicates. For instance, *isNearTo(archer1, tower)* or *colorOf(archer1, green)*. The HCI needs to know what predicates the user may use to specify relations. Here, the complexity of the HCI will depend greatly on the complexity of the domain. Our Wargus domain contains only binary relations, so we provide drop-down menus to the entity-relation-entity information.

Our ground advice format allows conjunctions, disjunctions, and negations. Thus, the HCI must also support specifying advice with these logical connectives. In our prototype, we provide the negation of all predicates in the "relation" menu. A button provides the ability to create conjunctions of multiple relations. Disjunctions are implied implicitly between different pieces of advice.

Reviewing Advice

From a usability standpoint, allowing the user to review previous actions is important. We do not explore this aspect of the HCI other than to mention that while using our GUI, we often examine previous advice to determine if we needed to state something new or not.

GENERATING GENERALIZED ADVICE AND LEARNING

Here we look at an advice-processing approach that converts the ground advice obtained through the advice-taking HCI into background knowledge in the form of generalized Horn clauses. A *Horn clause* is a disjunctive clause containing at most one positive literal (informally, one can view it as an IF-THEN rule written in first-order logic). Several relational learning algorithms, such as various inductive logic programming approaches and the bRDN algorithm described earlier, use a Horn-clause representation for background knowledge and can use such generated background knowledge directly.

Output from our advice-taking HCI is represented as a set of ground logical implications. As shown in Table 6, the consequence of a piece of advice states the category of a specific training example. The antecedent is a conjunction of literals defined within the domain (either as 'raw' features or via background rules). Multiple pieces of ground advice may be specified for a single training example and multiple examples may have associated advice. So a challenge we need to face is coherently combining multiple pieces of advice into one or more Horn clauses.

We transform the ground advice into generalized Horn clauses using the advice-handling process of Walker et al. [18]. Algorithm 1 presents the process. The algorithm performs essentially two phases. In the first phase (lines 5-11), it creates a number of conjunctive and disjunctive combinations of the ground advice. These new implications attempt to deduce possible meanings of the advice when considered as a whole. The algorithm considers three different combinations. First, it creates "per-piece" advice by considering each advice statement independently. Second, it creates "per-example" advice by conjoining all of the advice specified for a given example into a single combination. Finally, it creates "mega-rules." The set of mega-rules contains various combinations using all of the specified advice.

The "mega-rules" attempt to explain the possible intended meaning of all pieces of advice when considered together. For instance, the user might have intended that 'positive' advice indicates properties all positive examples have while 'negative' advice states properties the concept lacks ("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. …"). Alternatively, the user might have been specifying a more disjunctive concept ("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."). We generate multiple "mega-rules" since there is no way to directly determine the correct combination (if any).

As a second phase (lines 13-19), the algorithm generalizes each of the generated combinations. This occurs through a

Algorithm 1: CREATEGENERALIZEDADVICE

```
1.  Given:  G, a set of ground advice statements.
2.  Do:     Generate H, a set of generalized
3.          Horn-clause background knowledge
4.
5.  Let K = G // Per piece-of-advice formulas
6.  For each example e_j ∈ {e_1…e_n} with advice
7.     Generate "Per-example" formula p combining
8.        all advice from e_i into a single conjunctive
9.     Let K = K ∪ { p }
10.
11. Let K = K ∪ { "Mega-Rules" }  // See text
12.
13. For each ground advice statement k_i ∈ K
14.    Generalize k_i via anti-substitution yield
15.       the formula f_i
16.    Convert f_i to Horn clauses by expanding
17.       disjunctive formulas into multiple clauses and
18.       replacing negation by negation via failure
19.    Add generated Horn clauses to set H
20.
21. Return set H containing generalized clauses
```

process of anti-substitution in which it replaces each distinct constant that occurs in a given combination with a logical variable. At this point, it also converts the generalized implications into Horn clauses by transforming any implications that contain disjunctions into multiple logical implications. After converting the ground advice into generalized Horn clauses, we pass the background knowledge off to the learning algorithm. Table 3 illustrates the generated background knowledge for some simple ground advice.

EXPERIMENTAL RESULTS

In order to evaluate our HCI and advice-taking approach we performed experiments using the Wargus tower-defense game. We are interested in the effectiveness of advice generated using our advice-taking HCI versus both hand-written advice and using no advice.

Natural Language Advice versus HCI Advice

Our initial goal is to determine whether our HCI was capable of representing the types of advice a user might like to say in general. To evaluate that, we had group members who were not directly involved with Wargus nor its HCI (a) watch Wargus games, (b) learn (in their own minds) some basic strategy, and then (c) provide advice in ordinary English describing why a tower fell or stood for 5-10 specific initial states of our Wargus game.

A vast majority of the natural language advice was given in terms of the numbers and types of units in the attacking force. Overall, users provided 311 sentences of advice about 100 examples. Table 4 contains some general statistics gleaned from the natural language advice provided by the users. Users usually tended to give *specific* advice in terms of certain features such numbers of units, the presence or absence of a moat, and whether or not there was a ballista in the attacking force. For instance, "five archers are sufficient to destroy the tower," or conjunctive advice: "there is a moat and only footmen; hence the tower stands."

Table 3. Generated background knowledge for some simple ground advice. Initial ground advice is first generalized. Then various combinations are generated representing possible guesses at the meaning of the <u>set</u> of advice statements.

Description	Advice
Initial ground advice	ex(pos1) ← p(pos1) ∧ q(pos1). ex(pos1) ← r(pos1). ex(pos2) ← s(pos2).
"Per-piece" advice	a1(E) ← p(E) ∧ q(E). a2(E) ← r(E). a3(E) ← s(E).
"Per-example" advice	e1(E) ← p(E) ∧ q(E) ∧ r(E). e2(E) ← s(E).
"Mega" advice	m1(E) ← p(E) ∧ q(E) ∧ r(E) ∧ s(E). m2(E) ← p(E) ∧ q(E) ∨ r(E) ∨ s(E).

About 10% of the advice provide in natural language could not be expressed via our HCI. For instance, one user stated "the attacks are coming from many directions." Another mentioned "the north-most footman will absorb damage so the weaker archer can live longer." A small number of users also provided advice that was too *vague* (e.g., "there are too many attackers or "too few attackers") or described the existence of paths between attackers and the tower.

Our HCI is able to capture the vast majority of advice given by the users because it exploits the users' inclination to provide specific advice. Furthermore, it also provides mechanisms to allow users to provide general advice in terms of groups of units and even units in the scenario. The design is flexible enough to allow for various levels of specificity of advice as desired by the user.

Evaluating Advice Effectiveness

In order to evaluate our approach, we ran three separate experiments: one with no advice, one with hand-written advice, and one with advice obtained through our HCI. The HCI advice was based upon a representative sample of

Table 4. General statistics gleaned from the natural language advice provided by the users.

Feature Mentioned In Advice	Context	
	Tower stands	Tower falls
Total number of attackers	50	36
Number of Archers	43	62
Number of Footmen	50	46
Number of Ballistae	18	1
Number of Peasants	0	24
Presence of Moat	6	28
Other (terrain/distance/angle)	10	16

Table 5. The seven sentences of advice used.

Advice about *towerFalls* examples	Three or more footmen can take down a tower if there is no moat. Five or more archers can take down a tower. A single ballista is sufficient to destroy the tower.
Advice about *towerStands* examples	If there are only peasants, the tower stands. Four archers or less cannot take down a tower. One footman cannot take down the tower. If there is a moat, and no archers or ballista, the tower cannot be destroyed.

seven sentences (Table 5) expressed in natural language (we selected these seven sentences before running any experiments and did not modify them during the course of our experiments). We only used relations that could be entered through the HCI; we disregarded the rest of the natural language advice.

Table 6 shows some of the ground advice generated via the HCI. After creating the ground advice through the HCI, we used the advice generalization algorithm described previously to generate a set of background clauses, resulting in 21 separate pieces of background knowledge including the per-piece, per-example, and mega rules. While our advice taking learner can accept advice expressed in predicate calculus, the full richness of first-order logic was not needed to capture the human provided advice.

We created the hand-written advice without using the HCI, representing what a domain expert who understood the required knowledge representation would create, writing the advice directly as seven Horn clauses. Once we created the advice, we used the bRDN learning algorithm described earlier to generate learning curves comparing the use of (a) no advice, (b) hand-written advice, and (c) HCI generated advice.

Results and Discussion

We evaluated the learned models for the three separate experiments against a held-aside set of 900 testing examples. Figure 5 shows the area under the ROC curve (AUROC) for the three experiments. We tested significance of the results via the (two-tailed) sign test, a nonparametric test based on the binomial distribution [11]. The sign test is an exact test (McNemar's test is an approximation that was historically used purely because of computational limitations). The null hypothesis of the sign test is that both approaches are equally accurate; hence each test case for which the two approaches make different predictions is viewed as the flip of a fair coin. An approach "wins" such as test case if it predicts correctly and the other approach predicts incorrectly. Where there are N test cases for which the approaches give different predictions, and the most

Table 6. Sample ground logical statements about the Wargus tower-defense game for one positive (pos1) and one negative example (neg1), based on Table 5. Recall a positive example is when the tower stands.

Example	Advice
towerStands(pos1)	count(archers, pos1) = 0 ∧ count(footman, pos1) = 0 ∧ count(ballista, pos1) = 0.
towerStands(pos1)	count(archers, pos1) ≤ 4.
towerStands(pos1)	count(footman, pos1) = 1.
towerStands(pos1)	moatExists ∧ count(archers, pos1) = 0 ∧ count(ballista, pos1) = 0.
towerStands(neg1)	count(footman, neg1) ≥ 3.
towerStands(neg1)	count(archers, neg1) ≥ 5.
towerStands(neg1)	count(ballista, neg1) ≥ 1.

wins for either approach is h, the computed p-value is the probability of at least h heads by *either* method under the binomial distribution $b(N,0.5)$, that is, in N flips of a fair coin.

The difference in error rates between the HCI advice approach and the no advice approach is statistically significant ($p < 0.05$) at every point in the curve, with p-values as low as 3.89×10^{-15} at the largest difference in the curves (at 30 training examples). This demonstrates that, in this domain, the HCI-generated advice does improve learning, especially in the early regions of the learning curve.

The difference in error rates when using the two different types of advice is significant only for a few points in the curve, at 30, 70 and 100 examples. We consider this a positive result, as it indicates that our HCI approach performed as well as hand-written advice.

In addition to the bRDN experiments, we also compared using the HCI-generated advice with a knowledge-based

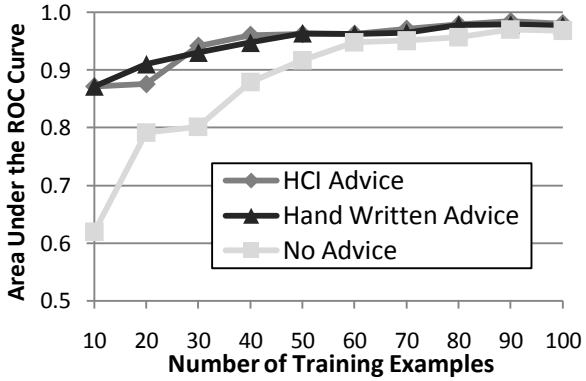

Figure 5. Learning curve showing test set performance in the Wargus tower-defense game comparing models learned with hand-written advice, HCI generated advice, and no advice. All models were learned the using boosted relational dependency network algorithm. The HCI generated advice was generalized using the techniques in Walker et al. [18].

support vector machine [5] as the learning algorithm. Although space does not permit a description of this algorithm, its results are similar to bRDN's.

Although in our relatively simple testbed the value of advice largely disappears with only 100 labeled example, in many tasks it is hard to get sufficient numbers of training examples. Advice is most effective when it is hard to get a good number of labeled examples yet easy to articulate helpful domain knowledge.

CONCLUSIONS AND FUTURE WORK

Providing formalized domain knowledge provides an effective method of substantially increasing the performance of supervised learning algorithms. However, this effectiveness is offset by the difficulty of formulating that knowledge, especially in the case of domain experts with little or no understanding of the learning algorithm or the knowledge representation required by these algorithms. Allowing the domain expert to specify knowledge as ground advice about specific examples through a human-computer interface provides one appealing method of overcoming this difficulty. We have demonstrated that for the Wargus tower-defense domain, ground advice obtained through an HCI outperforms learning with no advice and performs as well as advice written by an AI expert.

One interesting future direction would be to examine the effectiveness of refining advice by presenting the user with an HCI displaying the results of learning. Investigating the ability to support the learning of multiple concepts built upon one another might also prove insightful. Finally, allowing the user to provide advice in ordinary English would greatly extend the effectiveness of the advice-taking approach to improving supervised machine learning.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support of 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.

REFERENCES

[1] Blockeel, H. Top-down induction of first order logical decision trees. *AI Communications* (1999), 12, 1-2:119-120.

[2] Chen, J. and Weld, D. Recovering from errors during programming by demonstration. *Procs. of Intl. Conf. on Intelligent User Interfaces* (2008).

[3] Dietterich, T., Ashenfelter, A., and Bulatov, Y. Training conditional random fields via gradient tree boosting. *Procs. of Intl. Conf. of Machine Learning* (2004).

[4] Fails, J. and Olsen, D. Interactive machine learning. *Procs. of Intl. Conf. on Intelligent User Interfaces* (2003).

[5] Fung, G., Mangasarian, O., and Shavlik, J. Knowledge-based support vector machine classifiers. *Procs. of Neural Information Processing Systems* (2002).

[6] Gutmann, B. and Kersting, K. TildeCRF: Conditional random fields for logical sequences. *Procs. of European Conf. of Machine Learning* (2006).

[7] Heckerman, D., Chickering, D., Meek, C., Rounthwaite, R., and Kadie., C. Dependency networks for inference, collaborative, and data visualization. *Journal of Machine Learning Research* (2001), 1:49-75.

[8] Huang, T. and Mitchell, T. Text clustering with extended user feedback. *Procs. of Special Interest Group on Information Retrieval* (2006).

[9] Lieberman, H., ed. *Your Wish is My Command: Programming by Example*. Morgan Kaufmann Publishers, Inc, 2001.

[10] McDaniel, R. and Myers, B. Getting more out of programming-by-demonstration. *Procs. of Conf. on Human Factors in Computing Systems* (1999).

[11] Mendenhall, W., Wackerly, D., and Scheaffer, R. Nonparametric statistics. In *Mathematical Statistics with Applications (Fourth ed.)*. PWS-Kent, 1989.

[12] Muggleton, S. and De Raedt, L. Inductive logic programming: Theory and methods. *Journal of Logic Programming* (1994), 19,20:629-679.

[13] Natarajan, S., Khot, T., Kersting, K., Gutmann, B., and Shavlik, J. Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning* (2011).

[14] Neville, J. and Jensen., D. Relational dependency networks. *Journal of Machine Learning Research* (2007), 8:653-692.

[15] Richardson, M. and Domingos, P. Markov Logic Networks. *Machine Learning* (2006), 62:107-136.

[16] Stumpf, S., Rajaram, V., Li, L., and Wong, W. Interacting meaningfully with machine learning systems: Three experiments. *Intl. Journal of Human-Computer Studies* (2009), 67:639-662.

[17] Vander Zanden, B. and Myers, B. Demonstrational and constraint-based techniques for pictorially specifying application objects and behaviors. *Transactions on Computer Human Interaction* (1995), 2(4):308-356.

[18] Walker, T., O'Reilly, C., Kunapuli, G., Natarajan, S., Maclin, R., Shavlik, J., and Page, D. Automating the ILP setup task: Converting user advice about specific examples into general background knowledge. *Procs. of Intl. Conf. on Inductive Logic Programming* (2010).

[19] *Wargus Real-Time Strategy Game*. http://wargus.sourceforge.net.