Learning and Reasoning

Thomas G. Dietterich
Department of Computer Science
Oregon State University
Corvallis, OR 97331

May 26, 2003

Abstract

What is the relationship between learning and reasoning? Much recent work in machine learning has been criticized for focusing on learning and ignoring reasoning. This paper attempts to describe the various ways in which machine learning research has (and has not) incorporated reasoning. The paper argues that there are important computational, statistical, and engineering constraints that have produced the current state of affairs. These reasons are reviewed and assessed in the light of future research directions.

1 Introduction

Open a textbook on machine learning, and the first task description that you will see there is the task of supervised classification (Hastie, Tibshirani, & Friedman, 2001; Duda, Hart, & Stork, 2000). The goal of learning is to construct a *classifier*—that is, a function f that takes as input a description $\mathbf{x}$ of an object and outputs the class label y for the object. To achieve this goal, the learning algorithm is given a set of *training examples*, each of the form $(\mathbf{x}_i, y_i)$. This simple task has many applications (ranging from optical character recognition to face recognition to disease diagnosis), and it forms the basis of many commercial applications of predictive data mining. Nonetheless, students of artificial intelligence, psychology, and philosophy find this learning task to be distant from the kinds of learning that people engage in daily, and while they are impressed by the range of practical applications, they don't see how supervised classification relates to the larger goal of building broadly-intelligent systems. Where, in particular, is the *reasoning*?

This paper attempts to answer this question and discuss its implications for future learning reseach.

2 Learning for an Agent

Figure 1 shows the more-or-less standard model of an agent that is typically adopted in artificial intelligence research. The agent consists of a knowledge base and an inference engine, and it operates in a loop. In each iteration, it receives input in the form of observations of the environment (e.g., sensors), instructions (e.g., goals and/or utility functions), and queries. The inference engine carries out reasoning to answer the queries and to choose actions to execute in the environment. This reasoning process is driven by the data structures in the knowledge base.

Within this model, it is natural to focus learning on the contents of the knowledge base. Indeed, this is the primary justification for separating out the knowledge base from the inference engine. The

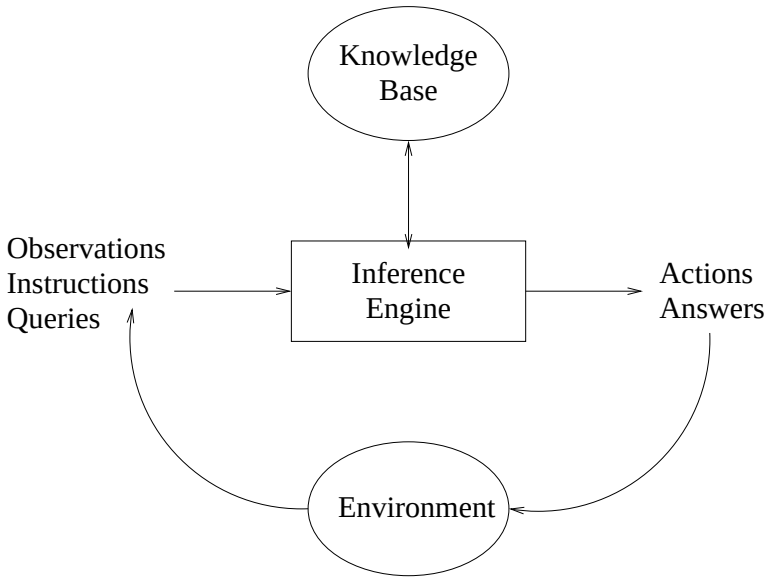

Figure 1: The Standard Model of an Agent

inference engine is kept fixed, and learning (or knowledge engineering) modifies only the knowledge base.

There are many kinds of knowledge that are typically represented in the knowledge base:

- ontologies (which define the kinds of objects that exist in the world and the attributes they possess),
- relationships (which define how objects and attributes are related; these can take the form of hard constraints (people have no more than 2 legs) or probability distributions (the height of Caucasian men in North America follows a normal distribution with a mean of 176cm),
- goals and utilities (which define desirable and undesirable states of the world and states of the agent), and
- policies, control rules, and heuristics (which prescribe ways of acting).

A "first principles" approach to agent design concentrates on the first three categories of knowledge: ontologies, relationships, and utilities. Given these, a sufficiently-powerful inference engine can infer optimal outputs (actions and answers). For example, consider the task of medical diagnosis. A patient with an illness visits a doctor. The doctor must choose a sequence of actions (queries, medical tests, attempted therapies) to diagnose the disease and heal the patient. This sequence of actions can be computed if the doctor knows the possible costs and effects of each potential action, the probabilistic relationships among all possible diseases and symptoms (including the results of queries, tests, and therapies), and the goals/utilities of the patient (e.g., preference for quality of life versus length of life). In short, the problem of choosing agent actions can be solved purely by reasoning given a sufficiently complete knowledge base and a sufficiently powerful inference engine. In this first-principles view, the role of learning is to learn the relevant ontologies, relationships, and utilities.

There are two well-known weaknesses of the first-principles approach. First, it is often computationally infeasible to perform the necessary inferences, particularly within the time constraints required for action in the world. Second, learning (or manual knowledge engineering) is an error-prone process, so the knowledge base will not be complete or correct. To be robust, the agent must take into account the uncertainty of the learned (or hand-encoded) knowledge when performing inferences and choosing actions.[1]

An alternative to relying on inference to choose actions is to store policies, control rules, and heuristics in the knowledge base and to use these for action selection. A policy is a function that maps directly from inputs (e.g., observations of the world) to actions. One can view policies as "cached" results of first-principles inference, but in many application problems, it is easier to describe the policy than it is to articulate the underlying ontological and relational knowledge sufficient to allow an inference engine to infer the policy. So the knowledge base may contain manually-engineered (or automatically-learned) policies that cannot be inferred (or explained) by the declarative knowledge in the knowledge base. Control rules and heuristics are bits of knowledge that constrain and guide the inference engine so that it arrives at the correct conclusions more efficiently. They typically incorporate knowledge of the task, the knowledge base, and the inference engine to prune unnecessary inferences and prioritize promising ones.

How does machine learning relate to this agent architecture? Most work on machine learning has centered on learning policies rather than declarative knowledge. Hence, as the critics point out, machine learning has avoided the need for an inference engine (and it has avoided the computational cost of reasoning). Even in cases where machine learning has studied learning declarative knowledge, it has learned knowledge only for very simple inference engines. There are many reasons for this. First, the machine learning community has focused on end-to-end performance, and this naturally leads to an emphasis on learning policies whose end-to-end performance can be directly evaluated (i.e., without dealing with an inference engine). Second, there is a statistical component to machine learning. Consequently, the inference engine must be probabilistic, and probabilistic inference is more costly and more complex than logical inference. Third, machine learning has studied only the simplest form of experience for learning: pairs of input observations and output actions. It has (for the most part) not exploited richer inputs such as natural language queries, explanations, instructions, and so on. There are severe statistical limitations to what can be learned from input-output pairs alone. A rough rule of thumb is that the number of input-output pairs needed for learning scales as the log of the size of the hypothesis space. In the case of rich first-order representations for ontologies and relations, the hypothesis space can be immense. There are 2^{2^n} boolean functions over n variables, so just learning boolean functions requires an exponential amount of data in the worst case. And boolean functions are not nearly as expressive as the kinds of languages used in modern knowledge bases!

3 Analysis of Existing Learning Approaches

Let us now analyze several lines of research in machine learning in terms of our agent model. We first consider the problem of classification and describe a series of approaches that start by learning a simple one-shot classification policy for classifying single objects and end up with methods that learn a declarative knowledge base that relates thousands of individuals and relies on complex reasoning methods. Then we consider the problem of sequential decision making and describe a range of approaches beginning with methods that learn policies and ending with methods that learn

[1]Note, however, that many expert systems do *not* take this into account, which is one of the sources of their brittleness.

declarative knowledge from which policies can be constructed.

3.1 Learning for Classification

The standard task of supervised learning is to learn a decision-making policy for "one-shot" classification tasks. In the one-shot classification task, the agent receives an input observation (e.g., age, body temperature, weight, height, blood count, strep test, etc. describing a patient) and must make a single decision (e.g., classify as "strep throat" versus "common cold" versus "healthy"). Each patient is assumed to be independent of the next, and the goal is to maximize the probability of correctly classifying a patient's disease. The reader should consider what is *not* included in this task. The agent makes only one decision rather than a sequence of decisions—for example, deciding which tests to perform, trying various therapies. The agent classifies each patient separately—that is, rather than looking for trends across patients (e.g., today, most of the patients have had a cold).

Many classification algorithms can be viewed as learning a propositional boolean formula that describes the classes. For example, the CN2 algorithm (Clark & Niblett, 1988) learns a decision list, which is a list of boolean conjunctions which are evaluated in order until one is satisfied. The FOIL algorithm (Quinlan, 1990) extended this to first-order logic. The AQ family of algorithms (Michalski, 1983) learn propositional or first-order disjunctive normal form formulas. Decision tree algorithms learn propositional (Quinlan, 1993) decision trees that constitute a compact representation of a DNF formula. Finally, several researchers have studied methods for learning propositional and first-order Horn clause programs, including recursive logic programs (Muggleton, 1992; Richards & Mooney, 1995). All of these methods require a run-time inference engine ranging in complexity from a simple evaluator of boolean formulae to an evaluator of pure prolog. It is important to note that all of these algorithms are learning classification policies rather than declarative knowledge.

There are several ways in which machine learning research has relaxed this simple one-shot classification task. The first relaxation is that instead of learning a classifier $f(\mathbf{x})$, the system learns a conditional probability estimator $P(y|\mathbf{x})$. An advantage of this approach is that at run time, the patient (or doctor) can specify a cost matrix $C(\hat{y}, y)$ which specifies the cost of classifying the disease as $\hat{y}$ when the true disease of the patient is y. This allows the patient to say, for example, that a false negative diagnosis (of strep) is 10 times more serious than a false positive diagnosis. With this utility information, the agent can choose the classification that minimizes the expected cost:

$$\hat{y} = \operatorname*{argmin}_k \sum_y P(y|\mathbf{x})C(k, y). \tag{1}$$

Here, the sum over y considers each of the possible true diseases, computes the probability that this is the true disease (according to $P(y|\mathbf{x})$), and then weights this by the cost of making decision k when the true disease is y.

Equation 1 can be viewed as an extremely simple inference engine, but this approach is the first step in making the learned knowledge $P(y|\mathbf{x})$ declarative rather than just learning a classification policy. It allows the cost matrix to be input at run time rather than being specified at learning time.

Machine learning research has taken additional steps toward learning declarative knowledge. One approach is to learn the joint distribution $P(\mathbf{x}, y)$ of the patient symptoms $\mathbf{x}$ and the disease y. This joint distribution is typically represented by some form of Bayesian network. The simplest such network is the Naive Bayes network shown in Figure 2. Here, the disease node y generates each of the observed symptoms $x_1, \ldots, x_n$. Each symptom is assumed to be conditionally independent given the disease. Figure 3 shows a not-so-naive Bayesian network for diabetes. We can interpret this causally as saying that age and number of pregnancies influence body mass, and all three of

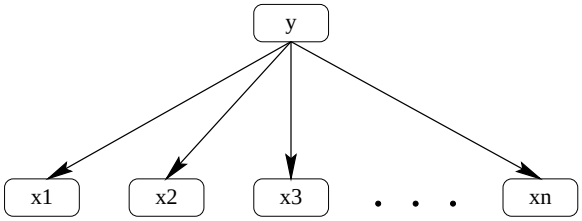

Figure 2: The Naive Bayes network

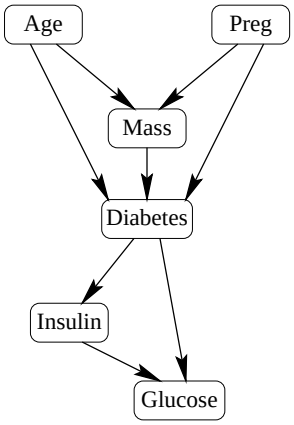

Figure 3: A Bayesian network for diabetes disease

these factors influence the probability of diabetes. Diabetes in turn influences the level of blood insulin, and, in combination with insulin, the level of blood glucose.

Given this representation of the joint distribution $P(\mathbf{x}, y)$, we now require a more powerful inference engine to make diagnostic decisions. The inference engine must compute $P(y|\mathbf{x})$ and then apply the minimum expected cost formula (equation 1) to make decisions. An important accomplishment of research in probabilistic reasoning is the development of general-purpose inference engines for this task (Jensen, 2001).

Recent research has taken an additional step in complexity by considering the simultaneous prediction of multiple patients via probabilistic relational models. Figure 4 shows a relational Bayesian network learned automatically using a system developed by Getoor, Friedman, Koller, and Pfeffer (2001) for tuberculosis. Rather than describing a single patient, this model describes a whole set of 1300 patients as well as 2300 people with whom they have been in contact and the various strains of TB that they may be infected with. This kind of network could be used to predict the strain of TB of a patient based on properties of the patient such as their age, ethnicity and HIV status but also based on the strains of TB carried by other people with whom they have been in contact. In order to make such a prediction for a new patient, nodes and links for that patient must be added to the network and then an inference engine must compute the probability of each TB strain given all of the information stored in the rest of the network. This inference problem is quite challenging, but current research in probabilistic reasoning has developed several good approximate reasoning strategies (Murphy, Weiss, & Jordan, 1999).

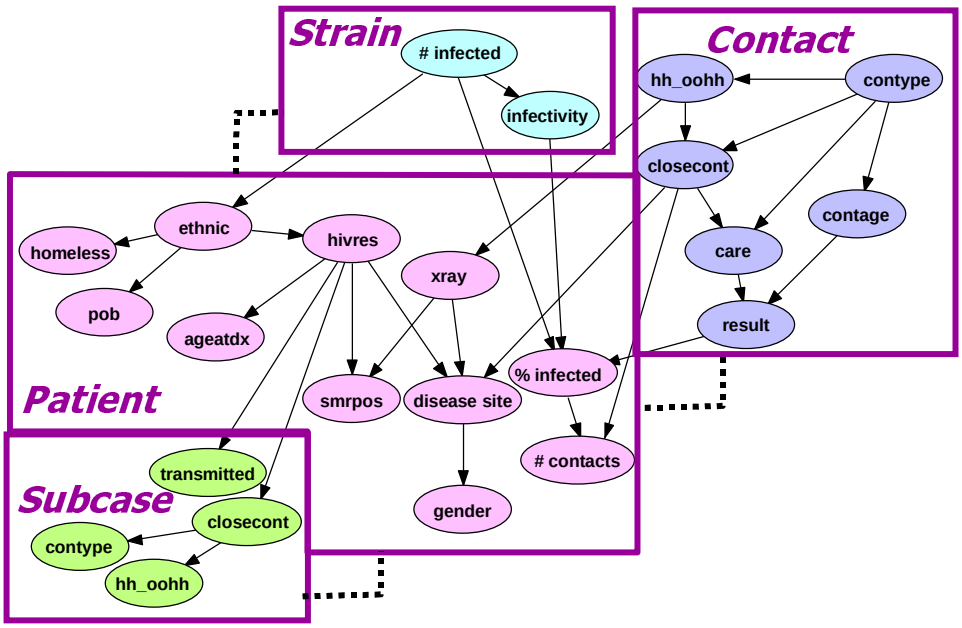

Figure 4: Probabilistic relational model (taken from Getoor, et al., 2000) of tuberculosis patients.

3.2 Learning for Sequential Decision Making

The methods discussed above focus on assigning a class to one or more individuals. These classes are assigned at a single time step—there is no consideration of making a sequence of decisions over time. We now consider this problem of sequential decision making.

The subarea of machine learning known as "reinforcement learning" studies the following problem (Sutton & Barto, 1998). At each time step t, the agent observes the current state $\mathbf{x}_t$ of the environment. The agent has available a set of actions A, and at each time step, it must choose one of these actions $a \in A$ to execute. When the chosen action a_t is executed, the environment makes a transition to a new state $\mathbf{x}_{t+1}$ according to the probability distribution $P(\mathbf{x}_{t+1}|\mathbf{x}_t, a_t)$. The environment also provides a real-valued reward r_t. The agent can then observe the resulting state $\mathbf{x}_{t+1}$ and choose a new action. The goal of the agent is to learn a policy π for choosing actions in order to maximize the expected long term sum of rewards.

Reinforcement learning research has explored three main approaches to solving this problem: (a) model-free policy search, (b) model-free value function learning, and (c) model-based learning. Model-free policy search starts with an initial policy π_θ (typically represented as a neural network with weights θ) and makes small changes to this policy after each interaction with the environment in order to improve the policy. These changes are based on estimating the derivative of the long-term expected reward with respect to θ and then changing θ in the direction of increasing reward (Williams, 1992; Baxter & Bartlett, 2000). A disadvantage of these methods is that they are very slow to learn a good policy. An advantage is that the policy can be computed very efficiently, so the agent can choose each action a_t in constant time. This gives good real-time performance in applications such as controlling autonomous helicopters (Ng & Jordan, 2000).

The second approach to sequential decision making is the model-free value function method. This approach learns a data structure known as the *action-value function*, which is denoted as Q^π. The action-value function assigns a value $Q^\pi(\mathbf{x}, a)$ to each state $\mathbf{x}$ and action a. This value is the

expected long term reward of starting in state $\mathbf{x}$, executing action a, and then selecting future actions according to a particular policy π. It is known that the optimal policy π^* must satisfy the Bellman equation:

$$Q^{\pi^*}(\mathbf{x}, a) = \sum_{\mathbf{x}'} P(\mathbf{x}'|\mathbf{x}, a) \left[R(\mathbf{x}, a, \mathbf{x}') + \max_{a'} Q^{\pi^*}(\mathbf{x}', a') \right], \tag{2}$$

where $R(\mathbf{x}, a, \mathbf{x}')$ is the average value of the reward received when executing action a in state $\mathbf{x}$ and making a transition to $\mathbf{x}'$. The model-free method known as Q-learning updates the action-value function Q after each interaction with the environment (Watkins, 1989).

Given the optimal Q function, a simple inference engine can compute the optimal policy as follows:

$$\pi^*(\mathbf{x}) = \operatorname*{argmax}_a Q(\mathbf{x}, a). \tag{3}$$

Hence, model-free value-function methods take a small step away from having no inference engine. (Indeed, equations 2 and 3 are the multi-step analogues of equation 1, which was a simple step away from direct classification.)

A much more significant step is taken by model-based reinforcement learning methods. These methods view each interaction with the environment as providing a four-tuple $\langle \mathbf{x}, a, r, \mathbf{x}' \rangle$ of data. From this four-tuple, these methods learn a declarative representation (referred to as a "model") of the transition function $P(\mathbf{x}'|\mathbf{x}, a)$ and the expected reward function $R(\mathbf{x}, a, \mathbf{x}')$. The model can take the form of tables for each of these quantities, or it can take other forms such as Bayesian networks. In any case, once the model has been learned, an inference engine can be applied to compute the optimal policy (and/or the optimal action-value function). The inference engine is usually based on the value iteration and policy iteration dynamic programming algorithms.

A limitation of all three of these approaches is that in most real-world problems, the space of possible states $\mathbf{x}$ is very large, and these algorithms typically require time that scales as the cube of the number of states. Hence, much recent research has focused on methods that construct computationally manageable approximations of the policy, value function, and model.

3.3 Declarative Methods versus Direct Methods

In both classification and sequential decision-making, methods that combine declarative knowledge with inference engines have several advantages. For example, the algorithms for learning Bayesian networks typically require time linear in the number of examples, whereas methods for learning classifiers directly (e.g., support vector machines, neural networks), are slower by several orders of magnitude. Model-based methods for reinforcement learning require many fewer training experiences than model-free policy and value function methods (Moore & Atkeson, 1993). Declarative methods in general allow the goal, utility function, or reward function to be changed at "inference time", whereas direct methods require that these be known at "learning time." Finally, declarative knowledge structures are much easier for people to understand, which makes them easier to validate. Why then do researchers and practioners often prefer model-free methods?

There are at least four reasons. First, declarative methods must make some assumption about the form of the joint probability distribution $P(\mathbf{x}, y)$ or the transition distribution $P(\mathbf{x}'|\mathbf{x}, a)$. In contrast, direct methods make fewer assumptions about the nature of the joint distribution. Conditional methods that only learn $P(y|\mathbf{x})$, for example, make no assumptions about the distribution of the $\mathbf{x}$ values. If the assumptions made by declarative methods are wrong (which they always are to some extent), then the inference engine may give results that are worse than those produced by the direct methods. This has been documented in several studies (e.g., Ng & Jordan, 2002).

Second, direct methods require no run-time reasoning, so they are easier to implement and they are faster at run time.

Third, there is a substantial body of psychological evidence showing that human skill takes the form of direct decision-making policies rather than run-time inference over declarative knowledge structures (e.g., Lee & Anderson, 1997). There is also evidence that declarative descriptions of this knowledge (e.g., as observed in verbal reports from human subjects) is reconstructive and not effective for problem solving.

Fourth, while declarative methods do not require knowledge of the task for which the learned knowledge will be used, they cannot exploit such knowledge if it is available. In contrast, direct methods can optimize their learning to address the task (Greiner, Grove, & Schuurmans, 1997). Consider learning a Bayesian network of the joint distribution $P(\mathbf{x}, y)$. Learning algorithms for this task try to find the network that best explains the data, which means best explaining both the $\mathbf{x}$ and the y values. But if the goal of learning is to predict y given $\mathbf{x}$, it is not necessary to explain the $\mathbf{x}$ values during learning.[2] Indeed, sometimes models of $P(\mathbf{x}, y)$ end up deciding that y is independent of the $\mathbf{x}$ values (even in cases where this is not true). This produces a Bayesian network that is useless for predicting y given $\mathbf{x}$ (Friedman, Geiger, & Goldszmidt, 1997). A direct method can ensure that this does not happen.

A fifth reason may be that virtually all machine learning systems are constructed and trained to perform only a single task. Given this focus, there is no benefit to factoring the learned knowledge into separate chunks that must then be recombined (e.g., by backward chaining or resolution) at run time to make inferences. If the learned knowledge were constructed in factored form, then this inference would need to be performed many times during the learning process, which would be slow and provide little benefit. Instead, a single (possibly very large) structure—such as a decision tree—is created that performs the task directly. It is interesting to note that the applications of machine learning exhibiting the most run-time inference are precisely those where there is a payoff to factoring the learned knowledge into reusable chunks (e.g., Richards & Mooney, 1995).

3.4 Probabilistic Inference Engines

All of the inference engines described above perform probabilistic reasoning. Elsewhere in artificial intelligence, inference engines perform logical reasoning. Why is there so little logical reasoning in machine learning?

Before answering this question, it is important to note that its premise is not quite true. As mentioned above, there are many algorithms that construct logical rules, and these do not require a probabilistic inference engine at run time. However, these are not declarative methods—they are learning direct decision-making policies and representing them as logical rules. Hence, it is generally correct to say that existing learning algorithms do not learn declarative knowledge represented in logic.

The reason that probabilistic inference is adopted is that it gives more accurate results. Consider, for example, the problem of speech recognition. The standard approach to speech is to learn declarative knowledge in the form of a hidden Markov model (HMM) that models $P(s, y)$, the joint distribution of the speech signal s and the sequence of words y (Jelinek, 1999). At run time, this learned knowledge is employed by a probabilistic inference engine, the Viterbi algorithm, to determine the most likely sequence of words given the observed speech signal. The Viterbi algorithm is normally executed using a beam search that keeps track of the top N word sequence hypotheses as it scans the speech signal. These hypotheses are sorted by their posterior probabilities. However,

[2] Technically, this is the difference between maximizing the joint likelihood of the model m, $P(\mathbf{x}, y|m)$, and maximizing the conditional likelihood, $P(y|\mathbf{x}, m)$.

if N is set to 1, then the Viterbi algorithm is equivalent to a propositional inference engine that irrevocably assigns the most likely word at each step given the corresponding segment of the speech signal. This gives significantly worse results than $N > 1$. We see in this case that logical inference gives worse results than probabilistic inference.

To understand this point more deeply, it is important to consider that learned knowledge is typically noisy and incorrect. Machine learning algorithms work from a set of training data, which is only a statistical sample of the full space of all possible cases. Hence, learning is a statistical process, and errors are inevitable. Probabilistic reasoning can partially recover from these errors. The Viterbi algorithm, for example, can find the correct word *sequence* even in cases where *one* of the correct words got a low score locally.

One alternative to probabilistic reasoning is provided by ensemble methods where multiple classifiers are constructed, non-probabilistic inference is performed for each classifier, and the results are combined (e.g., by voting) to produce a final decision. Like probabilistic reasoning, ensemble methods can recover from the statistical uncertainty of any particular classifier.

One form of logical inference that is common in expert systems is inference within a concept hierarchy. For example, people and cats are mammals, mammals and fish are animals, and so on. The reader might expect that in very complex classification problems, it would make sense to do some form of inheritance reasoning (e.g., learning that all mammals have hair and therefore not having to learn separately that people have hair and cats have hair). This is certainly true in general, but in the applications that have been studied so far, the concept hierarchy has been exploited in a different way. McCallum, Rosenfeld, Mitchell, & Ng (1998) studied the problem of classifying web pages into a topic hierarchy similar to Yahoo. They found that the best approach was to learn a classifier that "flattened" the hierarchy by only considering the leaves. However, their approach used the parent nodes in the hierarchy to "shrink" the probabilities of different leaf classes "toward" one another. As an example, consider the problem of estimating the probability of having hair given that you are a human, $P(hair|human)$ and the probability of hair given that you are a cat, $P(hair|cat)$. Cats are more likely to have hair than people (e.g., because people may be bald, may shave, etc.), so these probabilities are not the same. But these probabilities are more similar than $P(hair|salmon)$. In the training data, you may have only a few examples of humans and cats, which makes it hard to learn these probabilities accurately. You can learn them better if you partially pool the data by forcing the estimates of $P(hair|human)$ and $P(hair|cat)$ to be closer to each other because they are both subclasses of mammal. This gives better results than either learning $P(hair|mammal)$ or ignoring the hierarchy.

3.5 Summary

In response to the question, "Where is the reasoning?", I have argued that machine learning systems have explored a spectrum of methods ranging from purely direct methods (that require no reasoning at run time) to purely declarative methods (that require extensive, often probabilistic, reasoning at run time). I have argued that while there are many advantages to combining learned declarative knowledge with inference engines, there are also significant reasons why direct methods often outperform declarative ones. Finally, I have discussed the question of the relative paucity of run-time logical reasoning and emphasized the importance of probabilistic inference for good performance.

4 New Directions for Research in Learning and Reasoning

Machine learning poses a challenge to traditional research on reasoning and knowledge representation. The challenge is this: Are the logical representation and reasoning methods developed in

artificial intelligence at all useful when combined with learned knowledge? It is conceivable that the requirements and properties of learning (probabilistic representations and inference, statistical considerations such as shrinkage) may render logical representation and reasoning methods irrelevant.

However, existing high-performance systems based on traditional reasoning and knowledge representation pose an even greater challenge for machine learning. The challenge is this: How can machine learning automatically construct systems of similar high performance? It is conceivable that the methods developed to date in machine learning are irrelevant to the construction of these high-performance systems!

4.1 Challenge: Learning for high-performance inference engines

Let us consider the problem of scheduling. Scheduling systems typically employ inference engines based on constraint satisfaction or logical satisfiability. An advantage of AI-based scheduling methods is that they provide very expressive languages for representing constraints while still supporting efficient scheduling and re-scheduling (Zweben, Daun, & Deale, 1994). However, a tremendous amount of effort goes in to manually entering and maintaining the knowledge base of constraints, and new types of constraints are encountered from time-to-time that require extending the constraint language (and possibly the inference engine). The challenge for machine learning is to replace this manual entry and maintenance with a more autonomous approach.

The key issue in formulating this as a machine learning problem is to determine what kinds of information will be available to the learning system. One possibility would be to show the learning system examples of schedules constructed (or repaired) by people. Another possibility would be to allow people to provide natural language explanations of their scheduling decisions (or of what they don't like about the schedules constructed by the computer). In either case, the goal of the learning system would be to infer the classes of activities, resources, prerequisite constraints, and preferences that are consistent with the example schedules and the human-provided explanations.

I believe that our current stock of machine learning methods will require major changes to be able to handle learning from new, richer kinds of inputs. Conversely, the limitations of our current methods (e.g., to classification and sequential decision making) may reflect limitations in the kind of training data we have available. Removing or changing these limitations could have revolutionary consequences for learning research.

4.2 Challenge: Learning in support of multiple tasks

As discussed above, virtually all machine learning applications focus on a single task, and this may explain why these systems do not learn factored, declarative knowledge. An interesting challenge for machine learning research would be to learn knowledge useful for multiple tasks. For example, consider the challenge of learning simultaneously how to diagnose faults in a system (e.g., a computer network) and also how to optimize the performance of the system. Presumably, there is a shared core of knowledge about the structure and behavior of the system that could be learned and represented declaratively. In addition, there would perhaps be direct knowledge learned separately for diagnosis and for optimization.

Attempts to perform multiple tasks using the same knowledge base have encountered difficulties in the past. For example, when attempts were made to use the MYCIN knowledge base (which had been developed for diagnosis) to provide good explanations and to support intelligent tutoring, researchers found that the knowledge base needed to be substantially revised (Buchanan & Shortliffe, 1984).

4.3 Challenge: Learning from rich background knowledge and small samples

In many applications, there is a rich store of background knowledge available but rather small amounts of training data. This background knowledge could be easily represented using the existing knowledge representation tools. A challenge for machine learning is to develop methods for combining this background knowledge with the data to make decisions. There are three points at which the background knowledge could be incorporated into the learning and reasoning process: (a) in formulating the learning problem, (b) in constraining the learning process, and (c) in reasoning after learning. We explore each of these in turn.

4.3.1 Formulating the learning problem

Most machine learning applications succeed as a result of careful "feature engineering"—that is, careful design of the input features provided to the learning algorithm. In many cases, this process involves extensive reasoning by the data analyst. For example, in a project with Waranun Bunjongsat, I studied the problem of predicting grasshopper infestations in eastern Oregon. The available data consisted of 50 years of adult grasshopper population maps and daily weather data from about 75 weather stations. To learn from this data, we studied the life cycle of the grasshoppers to derive features that we thought would be useful for making the predictions. The grasshopper life cycle goes through three phases. Eggs are laid in the soil in late summer. The eggs mature underground until they hatch (typically in May). The grasshopper nymphs emerge and start eating (especially farmers' crops). Then as adults, they find mates and eggs are laid again. Before hatching, the eggs are largely insulated from weather events, although the temperature of the soil determines the rate of maturation. After hatching, cold weather can kill many of the nymphs.

To predict next year's grasshopper population, we need to know the number of eggs laid, the hatching date, and the weather after hatching. We need to learn the quantitative relationship between the weather and population losses. We cannot observe the number of eggs, but we know it is proportional to the number of adults, so we can use the number of adults as a proxy variable. We cannot observe hatching date, so we need to infer it. To infer it, we need to know soil temperature, but we cannot observe that. We can assume that it is proportional to air temperature, which we can observe. We know the maturation process is monotonically related to warm soil, so we can assume that hatching date can be predicted by comparing the number of degree days for air temperature (relative to some fixed level, say, 40°F) against a threshold θ. We know that hatching typically happens in May, so we can calibrate θ so that our predicted hatching dates are in May. Finally, we know that nymph deaths will be related to the duration and severity of the cold spells after hatching, so we can define the length of a cold spell in terms of a temperature threshold and define the severity by the minimum temperature during the spell. In our work, we hand-coded programs to extract these features and then provided them as input to a regression tree algorithm that estimated the adult grasshopper population (with very modest success). But we would have liked to have a system in which we could represent this background knowledge explicitly and then automatically or semi-automatically infer the relevant features. Such a system would provide a form of documentation for the feature engineering process. It might also improve the process by finding better features.

4.3.2 Constraining the learning process

The second role for background knowledge is to constrain the space of hypotheses considered by the learning algorithm. A nice example of this is the work by Clark and Matwin (1993) in which they applied a qualitative model of a factory to constrain the CN2 rule learning system to ensure

that the rules produced by CN2 were consistent with qualitative causal pathways in the model. Recently, there has been considerable attention on incorporating other kinds of constraints including monotonicity constraints (Bayardo, 2002) (e.g., that severity of disease increases monotonically with exposure to a toxin).

The general problem of incorporating background constraints into learning is very difficult. It is eased considerably if the learned knowledge is declarative, because then it reduces to the problem of preventing or detecting contradictions between learned knowledge and background knowledge. But for direct machine learning methods (policies, classifiers, etc.), it is much harder to relate the results of learning to the prior knowledge. This is an additional reason to prefer declarative representations.

4.3.3 Inference after learning

The third role for background knowledge is to perform inferences after learning. In part, this may be necessary if the knowledge could not be "compiled in" to the input features or the learning algorithm. The background knowledge can be applied to censor or veto bad decisions made by the learned policy. But more generally, if the goal is to combine learned knowledge with prior knowledge within the inference engine, then this will presumably impose additional constraints on the learning process and the representations involved. For example, to assess the quality of a proposed knowledge structure during learning, it might be necessary to provide the structure to the inference engine and see how well the engine was able to solve the overall task. Observed problems would need to be propagated back through the inference paths to the knowledge structure to guide repair.

5 Concluding Remarks

This paper has reviewed the issues relating learning and reasoning in artificial intelligence. Much work in learning has pursued "direct" forms of knowledge that can be applied without any subsequent inference. However, over the past few years, the limitations of the direct approach have pushed machine learning researchers to study more declarative forms of knowledge. Virtually all of this work employs probabilistic representations and probabilistic inference engines. Examples of this work include probabilistic relational models and model-based reinforcement learning.

Future research along this trajectory will study methods for learning ever-more-complex probabilistic models and for solving the accompanying probabilistic inference problems. But there are limits to what can be learned exclusively from a practical number of input/output training examples. One important direction for future research is to study problems with richer input information (e.g., explanations in natural language). Another important direction is to find ways of combining prior knowledge with reasonable amounts of training data to support complex, high-performance inference engines.

Acknowledgements

This research was funded by DARPA under seedling funds for the Knowledge Plane program. The author thanks Pat Langley, Ray Mooney, and Prasad Tadepalli for their critical reading of the manuscript.

References

Baxter, J., & Bartlett, P. L. (2000). Reinforcement learning in POMDP's via direct gradient ascent. In *Proc. 17th International Conf. on Machine Learning*, pp. 41–48. Morgan Kaufmann, San Francisco, CA.

Bayardo, R. J. (2002). Special issue on constraints in data mining. *SIGKDD Explorations*, *4*(1).

Buchanan, B., & Shortliffe, E. (1984). *Rule-Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley.

Clark, P., & Matwin, S. (1993). Using qualitative models to guide inductive learning. In *Machine Learning: Proceedings of the Tenth International Conference*, pp. 49–56 San Francisco, CA. Morgan Kaufmann.

Clark, P., & Niblett, T. (1988). The cn2 induction algorithm. *Machine Learning*, *3*, 261.

Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern Classification, Second Edition*. John Wiley and Sons, Inc.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning*, *29*, 131.

Getoor, L., Friedman, N., Koller, D., & Pfeffer, A. (2001). Learning probabilistic relational models. In *Relational Data Mining*. Springer-Verlag.

Greiner, R., Grove, A. J., & Schuurmans, D. (1997). Learning Bayesian nets that perform well. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, pp. 198–207 San Francisco, CA. Morgan Kaufmann Publishers.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Verlag, New York.

Jelinek, F. (1999). *Statistical methods for speech recognition*. MIT Press.

Jensen, F. V. (2001). *Bayesian Networks and Decision Graphs*. Springer-Verlag, New York.

Lee, F. J., & Anderson, J. R. (1997). Learning to act: Acquisition and optimization of procedural skill. In *Proceedings of the 19th Annual Conference of the Cognitive Science Society*, pp. 418–423 Mahwah, NJ. Lawrence Erlbaum.

McCallum, A., Rosenfeld, R., Mitchell, T., & Ng, A. (1998). Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the Fifteenth International Conference on Machine Learning* San Francisco. Morgan Kaufmann.

Michalski, R. S. (1983). A theory and methodology of inductive learning. In Michalski, R. S., Mitchell, T. M., & Carbonell, J. (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Vol. 1, pp. 83–134. Morgan Kaufmann.

Moore, A. W., & Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, *13*, 103.

Muggleton, S. (1992). Inductive logic programming. In Muggleton, S. (Ed.), *Inductive Logic Programming*, pp. 3–27. Academic Press, New York, NY.

Murphy, K., Weiss, Y., & Jordan, M. (1999). Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–99)*, pp. 467–475 San Francisco, CA. Morgan Kaufmann Publishers.

Ng, A. Y., & Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In Dietterich, T. G., Becker, S., & Ghahramani, Z. (Eds.), *Advances in Neural Information Processing Systems 14* Cambridge, MA. MIT Press.

Ng, A. Y., & Jordan, M. I. (2000). Pegasus: A policy search method for large mdps and pomdps. In *Uncertainty in Artificial Intelligence: Proceedings of the Sixteenth Conference (UAI-2000)*, pp. 405–415 San Francisco, CA. Morgan Kaufmann Publishers.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, *5*(3), 239–266.

Quinlan, J. R. (1993). *C4.5: Programs for Empirical Learning.* Morgan Kaufmann, San Francisco, CA.

Richards, B. L., & Mooney, R. J. (1995). Automated refinement of first-order horn-clause domain theories. *Machine Learning*, *19*, 95.

Sutton, R., & Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. Ph.D. thesis, King's College, Oxford. (To be reprinted by MIT Press.).

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*, 229.

Zweben, M., Daun, B., & Deale, M. (1994). Scheduling and rescheduling with iterative repair. In Zweben, M., & Fox, M. S. (Eds.), *Intelligent Scheduling*, chap. 8, pp. 241–255. Morgan Kaufmann, San Francisco, CA.