Chapter 12

Explanation-Based Learning

12.1 Deductive Learning

In the learning methods studied so far, typically the training set does not exhaust the version space. Using logical terminology, we could say that the classifier's output does not logically follow from the training set. In this sense, these methods are *inductive*. In logic, a *deductive* system is one whose conclusions logically follow from a set of input facts, if the system is sound.[1]

To contrast inductive with deductive systems in a logical setting, suppose we have a set of facts (the training set) that includes the following formulas:

$$\{Round(Obj1), Round(Obj2), Round(Obj3), Round(Obj4),$$

$$Ball(Obj1), Ball(Obj2), Ball(Obj3), Ball(Obj4)\}$$

A learning system that forms the conclusion $(\forall x)[Ball(x) \supset Round(x)]$ is inductive. This conclusion may be useful (if there are no facts of the form $Ball(\sigma) \wedge \neg Round(\sigma)$), but it does not logically follow from the facts. On the other hand, if we had the facts $Green(Obj5)$ and $Green(Obj5) \supset$

[1] Logical reasoning systems that are not sound, for example those using non-monotonic reasoning, themselves might produce inductive conclusions that do not logically follow from the input facts.

$Round(Obj5)$, then we could logically conclude $Round(Obj5)$. Making this conclusion and saving it is an instance of deductive learning—a topic we study in this chapter.

Suppose that some logical proposition, ϕ, logically follows from some set of facts, Δ. Under what circumstances might we say that the process of deducing ϕ from Δ results in our *learning* ϕ? In a sense, we implicitly knew ϕ all along, since it was inherent in knowing Δ. Yet, ϕ might not be obvious given Δ, and the deduction process to establish ϕ might have been arduous. Rather than have to deduce ϕ again, we might want to save it, perhaps along with its deduction, in case it is needed later. Shouldn't that process count as learning? Dietterich [Dietterich, 1990] has called this type of learning *speed-up* learning.

Strictly speaking, speed-up learning does not result in a system being able to make decisions that, in principle, could not have been made before the learning took place. Speed-up learning simply makes it possible to make those decisions more efficiently. But, in practice, this type of learning might make possible certain decisions that might otherwise have been infeasible.

To take an extreme case, a chess player can be said to learn chess even though optimal play is inherent in the rules of chess. On the surface, there seems to be no real difference between the experience-based hypotheses that a chess player makes about what constitutes good play and the kind of learning we have been studying so far.

As another example, suppose we are given some theorems about geometry and are asked to prove that the sum of the angles of a right triangle is 180 degrees. Let us further suppose that the proof we constructed did not depend on the given triangle being a right triangle; in that case we can learn a more general fact. The learning technique that we are going to study next is related to this example. It is called *explanation-based learning (EBL)*. EBL can be thought of as a process in which *implicit* knowledge is converted into *explicit* knowledge.

In EBL, we *specialize* parts of a *domain theory* to *explain* a particular *example*, then we *generalize* the explanation to produce another element of the domain theory that will be useful on similar examples. This process is illustrated in Fig. 12.1.

12.2 Domain Theories

Two types of information were present in the inductive methods we have studied: the information inherent in the training samples and the information about the domain that is implied by the "bias" (for example, the

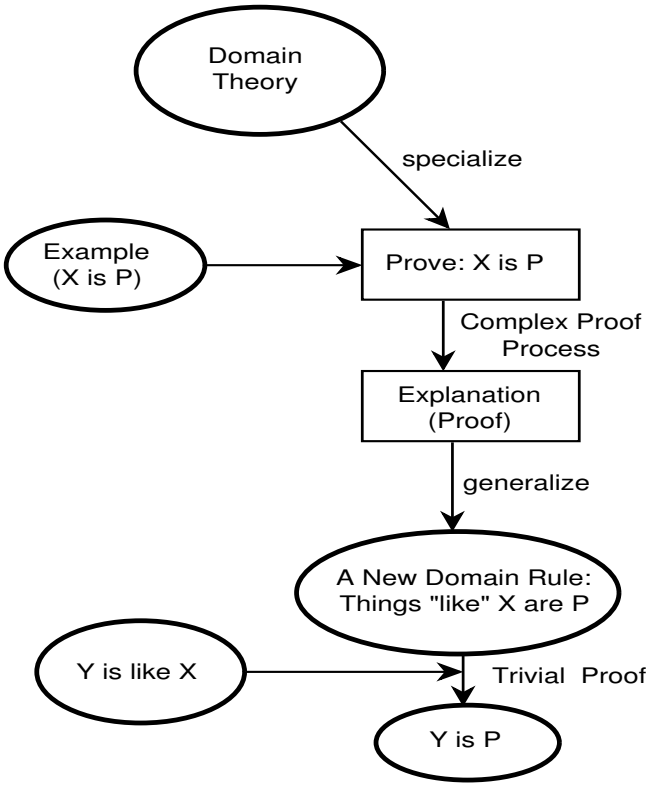


Figure 12.1: The EBL Process

hypothesis set from which we choose functions). The learning methods are successful only if the hypothesis set is appropriate for the problem. Typically, the smaller the hypothesis set (that is, the more a priori information we have about the function being sought), the less dependent we are on information being supplied by a training set (that is, fewer samples). A priori information about a problem can be expressed in several ways. The methods we have studied so far restrict the hypotheses in a rather direct way. A less direct method involves making assertions in a logical language about the property we are trying to learn. A set of such assertions is usually called a "domain theory."

Suppose, for example, that we wanted to classify people according to whether or not they were good credit risks. We might represent a person

by a set of properties (income, marital status, type of employment, *etc.*), assemble such data about people who are known to be good and bad credit risks and train a classifier to make decisions. Or, we might go to a loan officer of a bank, ask him or her what sorts of things s/he looks for in making a decision about a loan, encode this knowledge into a set of rules for an expert system, and then use the expert system to make decisions. The knowledge used by the loan officer might have originated as a set of "policies" (the domain theory), but perhaps the application of these policies were specialized and made more efficient through experience with the special cases of loans made in his or her district.

12.3 An Example

To make our discussion more concrete, let's consider the following fanciful example. We want to find a way to classify robots as "robust" or not. The attributes that we use to represent a robot might include some that are relevant to this decision and some that are not.

Suppose we have a domain theory of logical sentences that taken together, help to define whether or not a robot can be classified as robust. (The same domain theory may be useful for several other purposes also, but among other things, it describes the concept "robust.")

In this example, let's suppose that our domain theory includes the sentences:

$$Fixes(u, u) \supset Robust(u)$$

(An individual that can fix itself is robust.)

$$Sees(x, y) \wedge Habile(x) \supset Fixes(x, y)$$

(A habile individual that can see another entity can fix that entity.)

$$Robot(w) \supset Sees(w, w)$$

(All robots can see themselves.)

$$R2D2(x) \supset Habile(x)$$

(R2D2-class individuals are habile.)

$$C3PO(x) \supset Habile(x)$$

(C3PO-class individuals are habile.)

. . .

(By convention, variables are assumed to be universally quantified.) We could use theorem-proving methods operating on this domain theory to conclude whether certain robots are robust. These methods might be computationally quite expensive because extensive search may have to be performed to derive a conclusion. But after having found a proof for some particular robot, we might be able to derive some new sentence whose use allows a much faster conclusion.

We next show how such a new rule might be derived in this example. Suppose we are given a number of facts about Num5, such as:

$$Robot(Num5)$$

$$R2D2(Num5)$$

$$Age(Num5, 5)$$

$$Manufacturer(Num5, GR)$$

. . .

We are also told that $Robust(Num5)$ is true, but we nevertheless attempt to find a proof of that assertion using these facts about Num5 and the domain theory. The facts about Num5 correspond to the features that we might use to represent Num5. In this example, not all of them are relevant to a decision about $Robust(Num5)$. The relevant ones are those used or needed in proving $Robust(Num5)$ using the domain theory. The proof tree in Fig. 12.2 is one that a typical theorem-proving system might produce.

In the language of EBL, this proof is an *explanation* for the fact $Robust(Num5)$. We see from this explanation that the only facts about Num5 that were used were $Robot(Num5)$ and $R2D2(Num5)$. In fact, we could construct the following rule from this explanation:

$$Robot(Num5) \wedge R2D2(Num5) \supset Robust(Num5)$$

The explanation has allowed us to prune some attributes about Num5 that are irrelevant (at least for deciding $Robust(Num5)$). This type of pruning is the first sense in which an explanation is used to generalize the classification problem. ([DeJong & Mooney, 1986] call this aspect of explanation-based

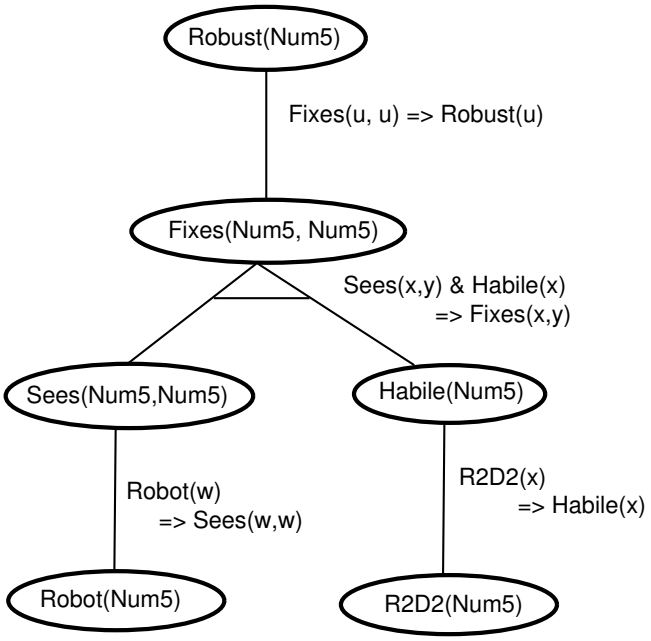


Figure 12.2: A Proof Tree

learning *feature elimination.*) But the rule we extracted from the explanation applies only to Num5. There might be little value in learning that rule since it is so specific. Can it be generalized so that it can be applied to other individuals as well?

Examination of the proof shows that the same proof structure, using the same sentences from the domain theory, could be used independently of whether we are talking about Num5 or some other individual. We can generalize the proof by a process that replaces constants in the tip nodes of the proof tree with variables and works upward—using unification to constrain the values of variables as needed to obtain a proof.

In this example, we replace $Robot(Num5)$ by $Robot(r)$ and $R2D2(Num5)$ by $R2D2(s)$ and redo the proof—using the explanation proof as a template. Note that we use different values for the two different occurrences of $Num5$ at the tip nodes. Doing so sometimes results in more general, but nevertheless valid rules. We now apply the rules used in the proof in the forward direction, keeping track of the substitutions imposed by the most general

unifiers used in the proof. (Note that we always substitute terms that are already in the tree for variables in rules.) This process results in the generalized proof tree shown in Fig. 12.3. Note that the occurrence of $Sees(r, r)$ as a node in the tree forces the unification of x with y in the domain rule, $Sees(x, y) \wedge Habile(y) \supset Fixes(x, y)$. The substitutions are then applied to the variables in the tip nodes and the root node to yield the general rule: $Robot(r) \wedge R2D2(r) \supset Robust(r)$.

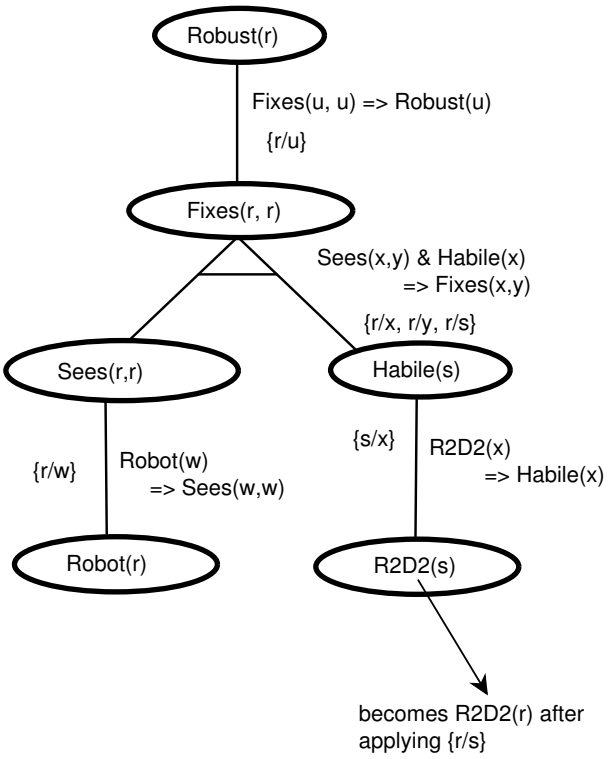


Figure 12.3: A Generalized Proof Tree

This rule is the end result of EBL for this example. The process by which $Num5$ in this example was generalized to a variable is what [DeJong & Mooney, 1986] call *identity elimination* (the precise identity of Num5 turned out to be irrelevant). (The generalization process described in this example is based on that of [DeJong & Mooney, 1986] and differs from that of [Mitchell, *et al.*, 1986]. It is also similar to that used

in [Fikes, *et al.*, 1972].) Clearly, under certain assumptions, this general rule is more easily used to conclude *Robust* about an individual than the original proof process was.

It is important to note that we could have derived the general rule from the domain theory without using the example. (In the literature, doing so is called *static analysis* [Etzioni, 1991].) In fact, the example told us nothing new other than what it told us about Num5. The sole role of the example in this instance of EBL was to provide a template for a proof to help guide the generalization process. Basing the generalization process on examples helps to insure that we learn rules matched to the distribution of problems that occur.

There are a number of qualifications and elaborations about EBL that need to be mentioned.

12.4 Evaluable Predicates

The domain theory includes a number of predicates other than the one occuring in the formula we are trying to prove and other than those that might customarily be used to describe an individual. One might note, for example, that if we used $Habile(Num5)$ to describe Num5, the proof would have been shorter. Why didn't we? The situation is analogous to that of using a data base augmented by logical rules. In the latter application, the formulas in the actual data base are "extensional," and those in the logical rules are "intensional." This usage reflects the fact that the predicates in the data base part are defined by their extension—we *explicitly* list all the tuples sastisfying a relation. The logical rules serve to connect the data base predicates with higher level abstractions that are described (if not defined) by the rules. We typically cannot look up the truth values of formulas containing these intensional predicates; they have to be derived using the rules and the database.

The EBL process assumes something similar. The domain theory is useful for connecting formulas that we might want to prove with those whose truth values can be "looked up" or otherwise evaluated. In the EBL literature, such formulas satisfy what is called the *operationality criterion*. Perhaps another analogy might be to neural networks. The evaluable predicates correspond to the components of the input pattern vector; the predicates in the domain theory correspond to the hidden units. Finding the new rule corresponds to finding a simpler expression for the formula to be proved in terms only of the evaluable predicates.

12.5 More General Proofs

Examining the domain theory for our example reveals that an alternative rule might have been: $Robot(u) \wedge C3PO(u) \supset Robust(u)$. Such a rule might have resulted if we were given $\{C3PO(Num6), Robot(Num6), \ldots\}$ and proved $Robust(Num6)$. After considering these two examples (Num5 and Num6), the question arises, do we want to generalize the two rules to something like: $Robot(u) \wedge [C3PO(u) \vee R2D2(u)] \supset Robust(u)$? Doing so is an example of what [DeJong & Mooney, 1986] call *structural generalization* (via *disjunctive augmentation*).

Adding disjunctions for every alternative proof can soon become cumbersome and destroy any efficiency advantage of EBL. In our example, the efficiency might be retrieved if there were another evaluable predicate, say, $Bionic(u)$ such that the domain theory also contained $R2D2(x) \supset Bionic(x)$ and $C3PO(x) \supset Bionic(x)$. After seeing a number of similar examples, we might be willing to *induce* the formula $Bionic(u) \supset [C3PO(u) \vee R2D2(u)]$ in which case the rule with the disjunction could be replaced with $Robot(u) \wedge Bionic(u) \supset Robust(u)$.

12.6 Utility of EBL

It is well known in theorem proving that the complexity of finding a proof depends both on the number of formulas in the domain theory and on the depth of the shortest proof. Adding a new rule decreases the depth of the shortest proof but it also increases the number of formulas in the domain theory. In realistic applications, the added rules will be relevant for some tasks and not for others. Thus, it is unclear whether the overall utility of the new rules will turn out to be positive. EBL methods have been applied in several settings, usually with positive utility. (See [Minton, 1990] for an analysis).

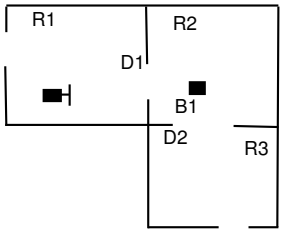
12.7 Applications

There have been several applications of EBL methods. We mention two here, namely the formation of macro-operators in automatic plan generation and learning how to control search.

12.7.1 Macro-Operators in Planning

In automatic planning systems, efficiency can sometimes be enhanced by chaining together a sequence of operators into *macro-operators*. We show an example of a process for creating macro-operators based on techniques explored by [Fikes, *et al.*, 1972].

Referring to Fig. 12.4, consider the problem of finding a plan for a robot in room $R1$ to fetch a box, $B1$, by going to an adjacent room, $R2$, and pushing it back to $R1$. The goal for the robot is $INROOM(B1, R1)$, and the facts that are true in the initial state are listed in the figure.



Initial State:

INROOM(ROBOT, R1)
INROOM(B1,R2)
CONNECTS(D1,R1,R2)
CONNECTS(D1,R2,R1)

. . .

Figure 12.4: Initial State of a Robot Problem

We will construct the plan from a set of STRIPS operators that include:

GOTHRU$(d, r1, r2)$

Preconditions: $INROOM(ROBOT, r1), CONNECTS(d, r1, r2)$
Delete list: $INROOM(ROBOT, r1)$
Add list: $INROOM(ROBOT, r2)$

PUSHTHRU$(b, d, r1, r2)$

Preconditions: $INROOM(ROBOT, r1), CONNECTS(d, r1, r2), INROOM(b, r1)$

Delete list: $INROOM(ROBOT, r1), INROOM(b, r1)$

Add list: $INROOM(ROBOT, r2), INROOM(b, r2)$

A backward-reasoning STRIPS system might produce the plan shown in Fig. 12.5. We show there the main goal and the subgoals along a solution path. (The conditions in each subgoal that are true in the initial state are shown underlined.) The preconditions for this plan, true in the initial state, are:

$$INROOM(ROBOT, R1)$$

$$CONNECTS(D1, R1, R2)$$

$$CONNECTS(D1, R2, R1)$$

$$INROOM(B1, R2)$$

Saving this specific plan, valid only for the specific constants it mentions, would not be as useful as would be saving a more general one. We first generalize these preconditions by substituting variables for constants. We then follow the structure of the specific plan to produce the generalized plan shown in Fig. 12.6 that achieves $INROOM(b1, r4)$. Note that the generalized plan does not require pushing the box back to the place where the robot started. The preconditions for the generalized plan are:

$$INROOM(ROBOT, r1)$$

$$CONNECTS(d1, r1, r2)$$

$$CONNECTS(d2, r2, r4)$$

$$INROOM(b, r4)$$

Another related technique that chains together sequences of operators to form more general ones is the *chunking* mechanism in Soar [Laird, *et al.*, 1986].

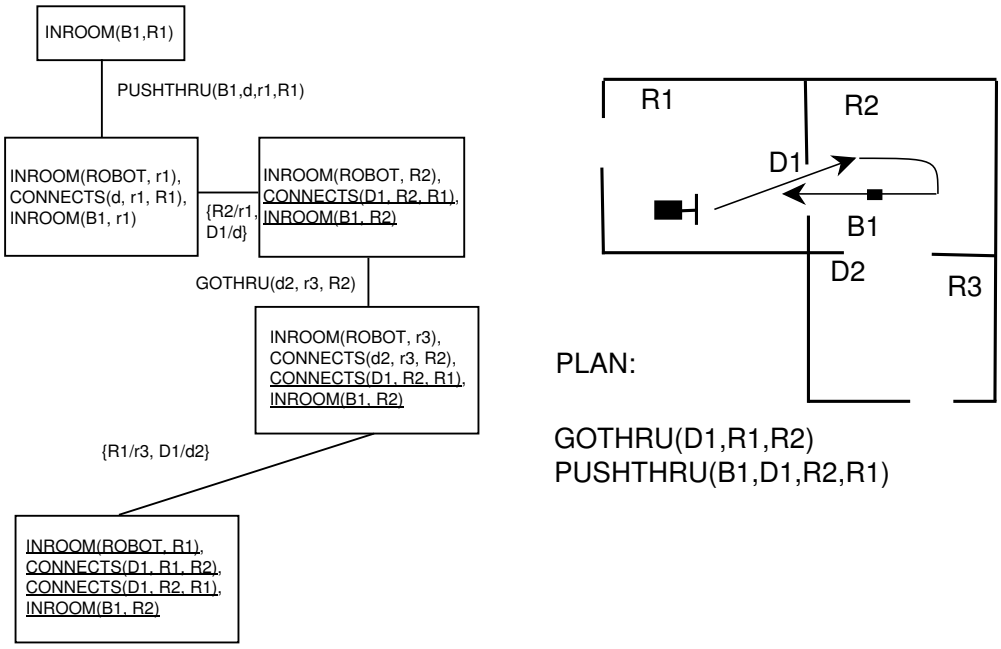


Figure 12.5: A Plan for the Robot Problem

12.7.2 Learning Search Control Knowledge

Besides their use in creating macro-operators, EBL methods can be used to improve the efficiency of planning in another way also. In his system called PRODIGY, Minton proposed using EBL to learn effective ways to control search [Minton, 1988]. PRODIGY is a STRIPS-like system that solves planning problems in the blocks-world, in a simple mobile robot world, and in job-shop scheduling. PRODIGY has a domain theory involving both the domain of the problem and a simple (meta) theory about planning. Its meta theory includes statements about whether a control choice about a subgoal to work on, an operator to apply, *etc.* either *succeeds* or *fails*. After producing a plan, it analyzes its successful and its unsuccessful choices and attempts to explain them in terms of its domain theory. Using an EBL-like process, it is able to produce useful control rules such as:

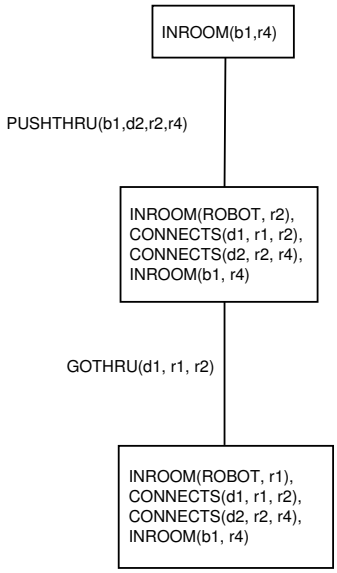


Figure 12.6: A Generalized Plan

```
IF (AND (CURRENT-NODE node)
        (CANDIDATE-GOAL node (ON  x y))
        (CANDIDATE-GOAL node (ON y z)))
THEN (PREFER GOAL (ON y z) TO (ON  x y))
```

PRODIGY keeps statistics on how often these learned rules are used, their savings (in time to find plans), and their cost of application. It saves only the rules whose utility, thus measured, is judged to be high. Minton [Minton, 1990] has shown that there is an overall advantage of using these rules (as against not having any rules and as against hand-coded search control rules).

12.8 Bibliographical and Historical Remarks

To be added.