Pruning Improves Heuristic Search for Cost-Sensitive Learning

Valentina Bayer Zubek BAYER@CS.ORST.EDU
Thomas G. Dietterich TGD@CS.ORST.EDU
Department of Computer Science, Oregon State University, Corvallis, OR 97331 USA

Abstract

This paper addresses cost-sensitive classification in the setting where there are costs for measuring each attribute as well as costs for misclassification errors. We show how to formulate this as a Markov Decision Process in which the transition model is learned from the training data. Specifically, we assume a set of training examples in which all attributes (and the true class) have been measured. We describe a learning algorithm based on the AO* heuristic search procedure that searches for the classification policy with minimum expected cost. We provide an *admissible heuristic* for AO* that substantially reduces the number of nodes that need to be expanded, particularly when attribute measurement costs are high. To further prune the search space, we introduce a *statistical pruning heuristic* based on the principle that if the values of two policies are statistically indistinguishable (on the training data), then we can prune one of the policies from the AO* search space. Experiments with realistic and synthetic data demonstrate that these heuristics can substantially reduce the memory needed for AO* search without significantly affecting the quality of the learned policy. Hence, these heuristics expand the range of cost-sensitive learning problems for which AO* is feasible.

1. Introduction

Given a training set of labeled examples, a classical machine learning task is to learn a classifier that can predict the class of new examples. The usual measure of performance for such classifiers is the error rate. However, in some application problems, we must also account for the cost of misclassification errors and the cost of measuring the attribute values. Consider, for example, the medical problem of diagnosing diabetes. Each medical test that a doctor performs on a patient has an associated cost. There are also costs for false positive and false negative diabetes diagnoses. Given a set of training examples, we wish to learn a diagnostic policy that minimizes the expected total cost of diagnostic decision making.

Some previous research has considered only misclassification costs (Margineantu, 2001), or only test costs combined with information gain heuristics (Norton, 1989; Nunez, 1991; Tan, 1993). More recently, researchers have begun to consider both test and misclassification costs (Turney, 1995; Greiner et al., 2002).

It is important to consider both types of costs together in diagnostic problems. If we consider only the misclassification costs, we ignore the delicate balance a diagnostician has to find in real life when choosing tests to perform, where she has to consider both how expensive the tests are and how informative they are. For example, if the most informative test is extremely expensive, the optimal classifier will prefer less-informative but less-costly tests. In contrast, a cost-insensitive method, such as C4.5, will choose the most informative test first, since it has the largest information gain.

If we consider only the cost of testing and just use the 0/1 loss to measure misclassification errors, other problems arise. In medicine, for example, the vast majority of patients are healthy, so the minimum-cost classifier will test no attributes and just predict that everyone is healthy. But a false negative diagnosis (i.e., declaring a sick patient to be healthy) is a life-threatening error in medicine that should be considered very expensive.

It is interesting to note that the problem of learning good diagnostic policies is difficult only when the cost of testing is comparable to the cost of misclassification. If the tests are very cheap compared to misclassification errors, then it pays to measure all of the attributes. If the tests are very expensive, then it pays to classify directly without measuring any attributes.

In this paper we introduce a new algorithm for learning classifiers to minimize the total cost of attribute measurement plus misclassification error. The algorithm is based on formulating the classification process as a Markov Decision Process whose optimal policy gives the optimal diagnostic procedure (on the training data). We then show how the AO* algorithm can search for this optimal policy, and we present an admissible search heuristic for AO*. This heuristic results in large reductions in the size of the search graph, particularly when the attribute measurement costs are high. To address problems where the attribute costs are low, and to reduce overfitting, we introduce an additional pruning heuristic that we call "statistical pruning." Statistical pruning exploits the fact that, given a small training sample, many MDP policies are statistically indistinguishable from one another. This means that we can safely prune some policies from the AO* search space without eliminating a policy that is indistinguishable from the optimal policy. We show experimentally that the combination of our admissible heuristic with statistical pruning yields large reductions in the size of the AO* search space without significantly changing the true cost of the learned policy. Hence, these heuristics expand the range of cost-sensitive learning problems for which AO* is feasible.

2. Cost-sensitive Learning with Attribute Costs is an MDP

The problem of cost-sensitive learning with attribute costs can be represented as a Markov Decision Process (MDP). We begin by defining the actions of this MDP. We assume that there are N measurement actions (tests) and K classification actions. Measurement action n (denoted x_n) returns the value of attribute x_n, which we assume is a discrete variable with values $\{v_1, ..., v_{V_n}\}$. Let $C(x_n)$ denote the cost of measurement action x_n. We assume it depends only on the action itself and not on which other measurements have already been performed, nor does it depend on the true class y. We believe these assumptions can be relaxed without substantial changes to our algorithm.

Classification action k (denoted f_k) is the act of classifying the patient into class $y = k$. The cost of a classification action depends on the true class of the example. Let $MC(f_k, y)$ be the misclassification cost of guessing class k when the true class is y.

Now let us define the states and transition probabilities for the MDP. There is one state for each possible combination of measured attributes. For example, the state $\{\}$ is the start state in which no attributes have been measured. The state $\{\text{age} = \text{old}, \text{insulin} = \text{low}\}$ is a state in which the "age" attribute has been measured to have the value "old" and the "insulin" attribute has been measured to have the value "low". For measurement action x_n executed in state s, the result state s' will be $s' = s \cup \{x_n = v\}$, where v is the measured value of x_n. The probability of this transition will depend on the values of all of the previously-measured attributes: $P_{tr}(s'|s, x_n) = P(x_n = v|s)$. The cost of this transition will be written $C(s, x_n)$, though with our assumptions it is only $C(x_n)$.

There is also a special terminal state s_f. Every classification action makes a transition to s_f with probability 1 (once a classification is made, the task terminates). Because the true class y of an example is not known at classification time, the cost of a classification action (which depends on y) must be viewed as a random variable whose value is $MC(f_k, y)$ with probability $P(y|s)$, which is the probability that the true class is y given the current state s. Fortunately, to compute the optimal policy for the MDP, we only need the expected cost of each action. The expected cost of classification action f_k in state s is $C(s, f_k) = \sum_y P(y|s) \cdot MC(f_k, y)$, which is independent of y.

If $V_n = V, \forall n$, then the total number of states of the MDP is $(V+1)^N + 1$, because an attribute can be either measured (so it will have one of the V values) or not measured. We assume that once an attribute is measured, it cannot be tested again. The set $A(s)$ of actions executable in state s consists of those attributes not yet measured and all the classification actions.

We will say that an example *matches* a state s if the example agrees with the attribute values defining s. Given a training set of labeled examples with no missing attribute values, we can directly learn the MDP's transition probabilities. $P_{tr}(s'|s, x_n)$ is computed as the ratio of the number of training examples that match state s' (where $x_n = v$) divided by the number of training examples that match s. We can also estimate the class probabilities $P(y|s)$ needed to compute the expected costs, $C(s, f_k)$, of the classification actions. $P(y|s)$ is the fraction of training examples matching state s that belong to class y.

A policy π for an MDP is a mapping from states to actions. The value function of a policy, $V^{\pi}(s)$, is the expected total cost of following policy π starting in state s until the terminal state is reached. The terminal state is always reached, because only finitely-many measurement actions can be executed, after which any classification action will cause the MDP to enter the terminal state. The optimal policy π^* minimizes $V^{\pi}(s)$ for all states, and its optimal value is $V^*(s)$. Note that for the terminal state $V^{\pi}(s_f) = 0$ for every policy π.

3. Heuristic Search

Each diagnostic policy can be viewed as a subtree of an AND/OR graph. Each OR node represents an MDP state s in which there is a choice among different possible actions a. Each AND node (s, a) corresponds to performing action a in state s, and it stores a probability distribution over the possible outcomes of the action. The value of the optimal policy for the AND/OR graph can be computed by a bottom-up sweep through the graph. First, assign the terminal state a value of 0. Then iteratively consider any AND or OR node all of whose children have already received values. Compute the value of an AND node (s, a) as the expected value of its action a. Compute the value of an OR node as the minimum of the values of its AND children. Continue this process until a value is assigned to the root OR node. This is the value of the optimal policy. We can view this computation as a single-pass version of value iteration in which each AND node computes $Q(s, a) = C(s, a) + \sum_{s'} P(s'|s, a) \cdot V(s')$, and each OR node computes $V(s) = \min_a Q(s, a)$. This AND/OR graph formulation suggests that we apply AO* to search for the optimal policy.

A similar approach was taken by Hansen (1998) and Washington (1997). They applied AO* search to find the optimal infinite-horizon policy in POMDPs (Partially Observable Markov Decision Processes). Hansen performed the search in the space of finite-state controllers, which is not compatible with our problem, since we can only measure an attribute once for a given example. Washington used the value function of the underlying MDP to define lower and upper bounds.

3.1 Overview of the AO* Algorithm

The AO* algorithm (Nilsson, 1980) computes the optimal solution graph of an AND/OR graph, given an admissible heuristic. A heuristic is admissible if it never over-estimates the optimal cost of getting from a state s to a terminal state.

During its search, AO* considers partial policies in which not all nodes have been expanded. The AO* algorithm repeats the following steps: in the current best partial policy, it selects an AND node to expand; it expands it; and then it recomputes (bottom-up) the optimal value function and policy of the revised graph. The algorithm terminates when the best decision in all leaf nodes of the current policy is to classify.

To apply AO* to cost-sensitive learning, we need to compute the probabilities in the AND nodes. We do this by "dropping" the training data through the AND/OR graph in the same way that data is "dropped" through a decision tree. The entire data set starts in the root node (the start state). At each OR node, the data is sent to all child AND nodes. At each AND node, the data is partitioned according to the value of the measured attribute. That is, if the AND node measures attribute x_n, then there will be one child OR node for each observed outcome $x_n = v$, and only training examples with $x_n = v$ will be sent to that node. Hence, the probability of the child OR node is the ratio of the number of examples in the child node to the number of examples in its parent node.

In practice, systematic search methods such as AO* are limited by memory rather than CPU time. To make AO* practical, we must reduce the number of nodes it expands. We will do this by introducing an admissible heuristic and a statistical pruning heuristic.

3.1.1 Optimistic Heuristic, Optimistic Values and Optimistic Policy

Our admissible heuristic provides an optimistic estimate, $Q^{opt}(s, a)$, of the expected cost of an unexpanded AND node (s, a). It is based on an incomplete 2-step lookahead search. The first step of the lookahead search is defined as $Q^{opt}(s, a) = C(s, a) + \sum_{s'} P_{tr}(s'|s, a) \cdot \ell^{opt}(s')$. Here s' represents one of the states resulting from action a. The second step of the lookahead is defined by the function $\ell^{opt}(s') = \min_{a' \in A(s')} C(s', a')$, which is the minimum over the cost of each classification action and the cost of each remaining attribute x' in s'. That is, rather than considering the states s'' that would result from measuring x', we only consider the cost of measuring x'. It follows immediately that $\ell^{opt}(s') \leq V^*(s')\ \forall s'$, because $C(s', x') \leq Q^*(s', x') = C(s', x') + \sum_{s''} P_{tr}(s''|s', x') \cdot V^*(s'')$. The key thing to notice is that the cost of measuring a single attribute x' is $\leq$ the cost of any policy that begins by measuring x', because the policy must pay the cost of at least one more action (classification or measurement) before entering s_f.

The definition of the optimistic Q value Q^{opt} can be extended to apply to all AND nodes as follows:

$$Q^{opt}(s, a) = \begin{cases} C(s, a) \\ \quad \text{if } a = f \text{ (a classification action)} \\ C(s, a) + \sum_{s'} P_{tr}(s'|s, a) \cdot \ell^{opt}(s') \\ \quad \text{if } (s, a) \text{ is unexpanded} \\ C(s, a) + \sum_{s'} P_{tr}(s'|s, a) \cdot V^{opt}(s') \\ \quad \text{if } (s, a) \text{ is expanded} \end{cases}$$

where $V^{opt}(s) \stackrel{\text{def}}{=} \min_{a \in A(s)} Q^{opt}(s, a)$. The optimistic policy is $\pi^{opt}(s) = \text{argmin}_{a \in A(s)} Q^{opt}(s, a)$. Every OR node s stores its optimistic value $V^{opt}(s)$, and every AND node (s, a) stores $Q^{opt}(s, a)$. Theorem 1 proves that Q^{opt} and V^{opt} form an admissible heuristic.

Theorem 1 *For all s and all $a \in A(s)$, $Q^{opt}(s,a) \leq Q^*(s,a)$, and $V^{opt}(s) \leq V^*(s)$.*

Proof by induction:

Base case for Q^{opt}
If a is a classification action f, then $Q^{opt}(s,f) = C(s,f) = Q^*(s,f)$. If (s,a) is unexpanded, then $Q^{opt}(s,a) = C(s,a) + \sum_{s'} P_{tr}(s'|s,a) \cdot \ell^{opt}(s') \leq C(s,a) + \sum_{s'} P_{tr}(s'|s,a) \cdot V^*(s') = Q^*(s,a)$, because $\ell^{opt}(s') \leq V^*(s')$.

Base case for V^{opt}
Let s_u be any state where all its AND nodes (s_u,a), if any, are unexpanded. Then $V^{opt}(s_u) \leq V^*(s_u)$ follows from the base case of Q^{opt}.

Any other state s_e in the graph has at least one AND node (s_e,a) previously expanded. Let s_c be any of the result states of (s_e,a).

Induction hypothesis
If $V^{opt}(s_c) \leq V^*(s_c), \forall s_c$, then $Q^{opt}(s_e,a) \leq Q^*(s_e,a), \forall a \in A(s_e)$, and $V^{opt}(s_e) \leq V^*(s_e)$.

We only need to consider the case of an expanded action $a \in A(s_e)$, because the other situations are covered by the base case. By definition, $Q^{opt}(s_e,a) = C(s_e,a) + \sum_{s'} P_{tr}(s'|s_e,a) \cdot V^{opt}(s')$. Because s' is one of the result states of (s_e,a), we can apply the induction hypothesis with $s_c = s'$, so $V^{opt}(s') \leq V^*(s')$, hence $Q^{opt}(s_e,a) \leq Q^*(s_e,a)$. It follows that $Q^{opt}(s_e,a) \leq Q^*(s_e,a), \forall a \in A(s_e)$, and $V^{opt}(s_e) \leq V^*(s_e)$. **Q.E.D.**

3.1.2 Realistic Values and Realistic Policy

We also introduce an upper bound, $V^{real}(s)$, which is an overestimate of the value of the optimal policy rooted at s. Every OR node s stores a realistic value $V^{real}(s)$, and every AND node (s,a) stores a realistic Q value, $Q^{real}(s,a)$. For $a \in A(s)$ define

$$Q^{real}(s,a) = \begin{cases} C(s,a) \\ \quad \text{if } a = f \text{ (a classification action)} \\ C(s,a) + \sum_{s'} P_{tr}(s'|s,a) \cdot V^{real}(s') \\ \quad \text{if } (s,a) \text{ is expanded} \\ \text{ignore} \quad \text{if } (s,a) \text{ is unexpanded} \end{cases}$$

and $V^{real}(s) = \min_{a \in A'(s)} Q^{real}(s,a)$, where the set $A'(s)$ is $A(s)$ without the unexpanded actions. The realistic policy is $\pi^{real}(s) = \mathrm{argmin}_{a \in A'(s)} Q^{real}(s,a)$.

In the current graph expanded by AO*, assume that we ignore all unexpanded AND nodes (s,a). We call this graph the *realistic* graph. The current realistic policy is the best policy (according to the training data) from this realistic graph.

Theorem 2 *The realistic value V^{real} is an upper bound: $V^*(s) \leq V^{real}(s), \forall s$.*

Proof: Base case: By definition, a leaf node s in the realistic graph has $\pi^{real}(s) = f$, where f is the best classification action in s. Therefore $V^{real}(s) = C(s,f) \geq V^*(s)$, because we ignore unexpanded AND nodes (s,x_n), and it could well be that $C(s,f) \geq Q^*(s,x_n)$. Induction step: The internal nodes in the graph compute their realistic values using a one step Bellman backup based on realistic values of the next states, $V^{real}(s) = \min\limits_{a \in A'(s)} [C(s,a) + \sum\limits_{s'} P_{tr}(s'|s,a) \cdot V^{real}(s')]$. **Q.E.D.**

Corollary 1 *If a is an expanded action in state s or a classification action, then $Q^*(s,a) \leq Q^{real}(s,a)$.*

Proof: If a is a classification action f then $Q^{real}(s,f) = C(s,f) = Q^*(s,f)$. If a is an expanded action, the proof is immediate from the definition of $Q^{real}(s,a)$ based on one step lookahead, and using Theorem 2 in the result states s'. **Q.E.D.**

3.1.3 Node Expansion

In the current optimistic policy π^{opt}, we choose to expand the unexpanded AND node $(s, \pi^{opt}(s))$ with the largest impact on the root, defined as

$$\operatorname*{argmax}_s \, (V^{real}(s) - V^{opt}(s)) \cdot P_{reach}(s|\pi^{opt}),$$

where $P_{reach}(s|\pi^{opt})$ is the probability of reaching state s from the root, following policy π^{opt}. The difference $V^{real}(s) - V^{opt}(s)$ tells how much we expect the value of state s to change if we expand $\pi^{opt}(s)$.

3.1.4 AO* Implementation

Our implementation of AO* is the following:

repeat
  select an AND node (s,a) to expand (using π^{opt}, V^{opt}, and V^{real}).
  expand (s,a).
  do bottom-up updates of $Q^{opt}, V^{opt}, \pi^{opt}$ and of $Q^{real}, V^{real}, \pi^{real}$.
until there are no unexpanded nodes reachable by π^{opt}.

As more nodes are expanded, the optimistic values V^{opt} increase, becoming tighter lower bounds to the optimal values V^*, and the realistic values V^{real} decrease, becoming tighter upper bounds. They converge to the value of the optimal policy on the training data: $V^{opt}(s) = V^{real}(s) = V^*(s)$, $\forall s$ reachable by π^*.

For an unexpanded AND node (s, a), if $V^{real}(s) < Q^{opt}(s, a)$, then action a does not need to be expanded, and therefore our heuristic provides a cut-off in node expansions. In this case, $V^{opt}(s) \leq V^*(s) \leq V^{real}(s) < Q^{opt}(s, a) \leq Q^*(s, a)$. So $V^{opt}(s) < Q^{opt}(s, a) \Rightarrow \pi^{opt}(s) \neq a$, and $V^*(s) < Q^*(s, a) \Rightarrow \pi^*(s) \neq a$.

The AO* algorithm can be viewed as an "anytime" algorithm. If we want to stop the algorithm after a certain number of nodes have been expanded or if we run out of memory, then we can return the realistic policy π^{real} computed up to that point.

3.2 Statistical Pruning

For problems with many low-cost, weakly-informative attributes, our admissible heuristic is unable to prune many nodes, and hence, the search space will explode. If we consider this explosion from the perspective of machine learning, we are led to ask whether it is really necessary to consider all of these different policies. Given a small training sample, there are many pairs of policies that are statistically indistinguishable. We would like to find a way that we can prune subgraphs from the AND/OR graph while ensuring that AO* still finds a policy statistically indistinguishable from the optimal policy. Let us formalize this observation as the following indifference principle:

Indifference Principle. *Given two policies π_1 and π_2 whose values are statistically indistinguishable based on the training data set, a learning algorithm can choose arbitrarily between them.*

Strictly speaking, we cannot apply this principle until AO* has generated two complete policies π_1 and π_2, and hence, we could not apply the principle to avoid computation. However, we can apply the principle heuristically by identifying an AND node (s, a) with the following properties: (i) action a has not been expanded, (ii) with high probability, when a is expanded, the resulting optimal policy rooted at a will be statistically indistinguishable from the current realistic policy rooted at s. If such an AND node (s, a) can be found, then we can prune it from the graph. We call this heuristic *statistical pruning* (abbreviated SP).

To decide whether an unexpanded AND node (s, a) has property (ii), we perform a statistical test to determine whether $V^{real}(s)$ is statistically distinguishable from $Q^{opt}(s, a)$. If the test cannot reject the null hypothesis that these two values are equal, then we prune the AND node from the graph. The statistical test is a paired test for the difference of two means. For each training example j that matches state s, we compute its actual cost $c^{real}(j)$ according to π^{real} and its optimistic cost $c^{opt}(j)$ according to π^{opt}. These costs are based on the observed values of j and its true class y_j. We then compute a normal confidence interval for the mean of the difference $c^{real}(j) - c^{opt}(j)$, and if this interval contains zero, then we cannot reject the null hypothesis that $V^{real}(s) = Q^{opt}(s, a)$.

Statistical pruning is easily incorporated into AO*. When a node (s, a) is selected to be expanded, we apply the paired-difference test. If the test decides to prune this action, then we delete it from $A(s)$ and update $Q^{opt}, V^{opt}, \pi^{opt}$. The realistic graph does not need to be updated, because (s, a) was unexpanded, so it did not and will not influence the realistic policy.

Statistical pruning is a heuristic that approximately implements the Indifference Principle. Most of the time, the statistical test correctly identifies AND nodes whose expansions cannot lead to a policy that is statistically distinguishable from π^{real}. This is because $Q^{opt}(s, a) \leq Q^*(s, a)$, so expanding AND node (s, a) can only cause the estimated value $Q^{opt}(s, a)$ to increase. Hence, if $Q^{opt}(s, a)$ is indistinguishable from $V^{real}(s)$, then $Q^*(s, a)$ is likely to be even closer to $V^{real}(s)$. The heuristic can fail if $Q^*(s, a)$ has very low variance on the training set so that even though it is closer to $V^{real}(s)$, it is also statistically distinguishable. We believe such cases will arise rarely.

4. Experiments

To test the effectiveness of the admissible heuristic and the SP heuristic, we implemented AO* with the admissible heuristic (algorithm AO*), and AO* with both heuristics (algorithm AO*+SP). We tested them on two synthetic problems and we measured the amount of memory required by both algorithms to converge. The memory used is computed as the total number of OR nodes and AND nodes. We also measured the quality of the learned policy to see whether the memory savings achieved by statistical pruning caused a significant change in the quality of the policy.

4.1 Naive Bayes Synthetic Problem

Our first set of experiments is conducted on a synthetic problem generated from a simple Naive Bayes network. The network has the following structure. There are two classes $y = 0$ and $y = 1$ with equal probability $P(y = 0) = P(y = 1) = 0.5$. There are 8 binary attributes. Attribute x_1 has probabilities $P(x_1 = 0|y = 0) = P(x_1 = 0|y = 1) = 0.5$, so it provides no information about the class. The remaining 7 attributes share the same symmetric distribution

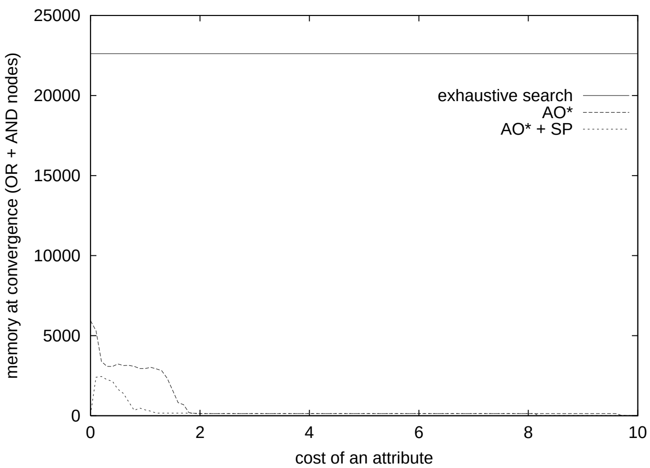

Figure 1. Naive Bayes problem, AO* memory and AO*+SP memory at convergence for different costs of an attribute, for $p = 0.1$, 1000 training examples.

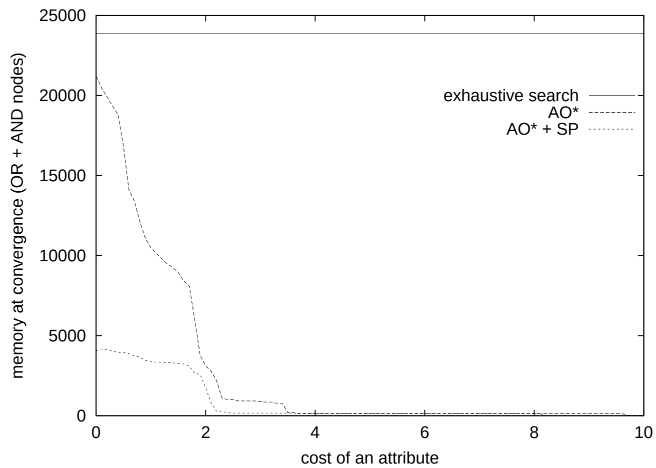

Figure 2. Naive Bayes, $p = 0.2$, 1000 training examples.

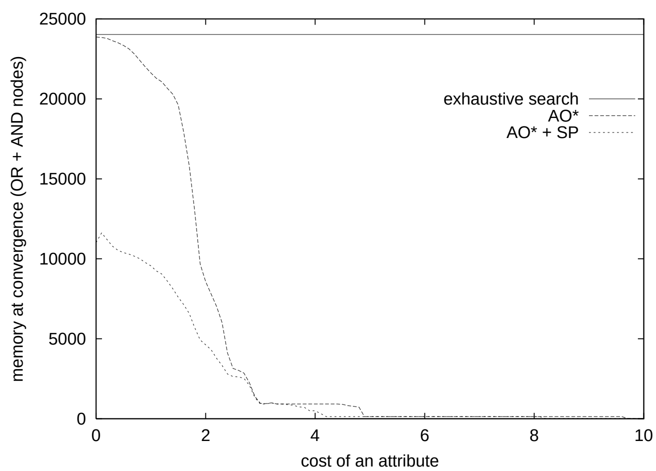

Figure 3. Naive Bayes, $p = 0.3$, 1000 training examples.

$P(x_i = 0|y = 0) = P(x_i = 1|y = 1) = p$, $\forall i > 1$. In the experiments, we vary p from 0.1 to 0.4 in steps of 0.1. As p approaches 0.5, the attributes become useless; as p approaches 0, the attributes become perfect predictors of the class. The cost of misclassification (false positive or false negative) is fixed at 20, the cost of measuring x_1 is fixed at 0.1, and the remaining attributes have identical cost c, which we vary in the experiments from 0 to 10 in steps of 0.1. This upper limit of 10 was chosen to be half the misclassification cost, because, for more expensive attributes, the optimal diagnostic policy would be to classify immediately without measuring any attributes.

For each value of p, we generated 1000 training examples from the Naive Bayes network. For each attribute cost c, we run the AO* and AO*+SP algorithms. We employed a confidence level of 99.9999% for the statistical tests in the SP heuristic. After each node expansion, we measured the performance of the realistic policy on the true model. We also computed the amount of memory consumed by each algorithm.

Figures 1, 2, 3 and 4 show the results for $p = 0.1$, $p = 0.2$, $p = 0.3$, and $p = 0.4$. Each figure compares the amount of memory consumed by exhaustive search, by AO*, and by AO*+SP. All methods are run to completion. The horizontal axis gives the cost of measuring an attribute. We see that the admissible heuristic gives a massive reduction in memory, particularly when the cost c is large. This is because the optimal policy tests few or no attributes, and the admissible heuristic prunes most of the search space. In such cases, statistical pruning is not needed. The difficult cases for AO* are when the test cost c is small, because this is when the optimal policy can afford to test many attributes, and hence, the AND/OR graph grows large. Here, we see that SP gives important additional memory reductions. When $c < 2$, the reductions range from 17-fold ($p = 0.1$, $c = 1.3$) to 5-fold ($p = 0.2, c = 0.1$) to 1-fold ($p = 0.4, c = 1.9$).

To determine whether SP caused an increase or decrease in the total expected cost of the policy, we compared the values of the policies learned using equal amounts of memory. Specifically, for each combination of p and c, we trained both AO* and AO*+SP to convergence. We then determined which method (usually AO*+SP) used the smallest amount of memory at

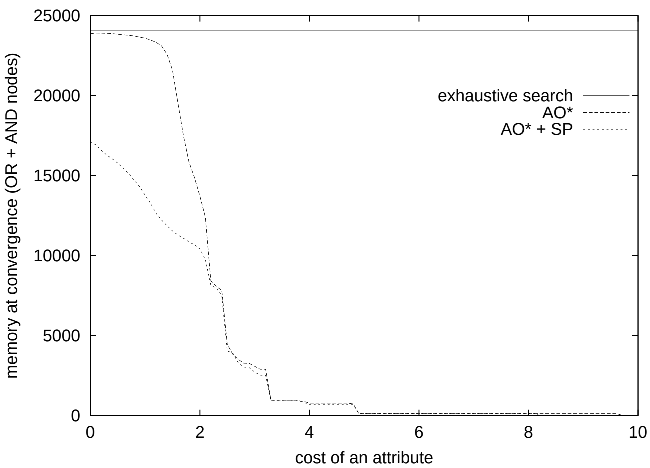

Figure 4. Naive Bayes, $p = 0.4$, 1000 training examples.

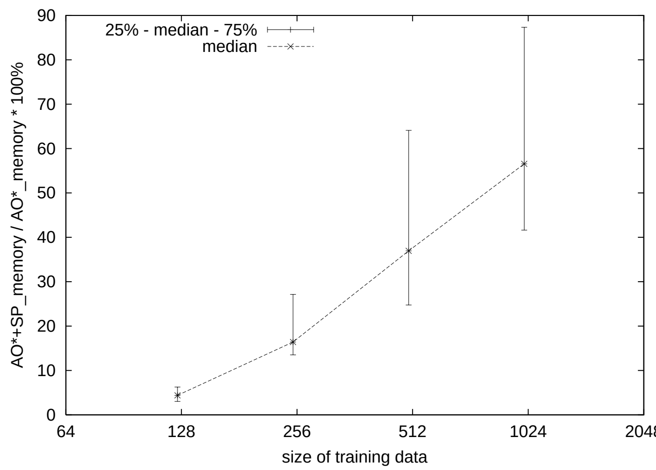

Figure 5. Naive Bayes. Percentage of AO* memory used by AO*+SP for all (p, c) combinations.

convergence. Let π_1 be the final policy computed by this method. We then found the point where the other method (usually AO*) used the same amount of memory. Let π_2 be this policy. We then computed the values of π_1 and π_2 on the true model and compared them. There are 404 combinations of p and c values. Using the sign test (with significance level of 0.05), we cannot reject the null hypothesis that the policies computed by AO* have the same values as the policies computed by AO*+SP. Hence, the reductions in memory were obtained without any statistically significant change in performance of the learned policies.

We also ran experiments with smaller training sets to determine how the size of the training set affects statistical pruning. Figure 5 shows the percentage of AO* memory used by AO*+SP (ignoring the easy cases where AO*+SP memory $<$ 200 nodes). This shows that SP becomes more and more effective as the size of the training set becomes smaller, because more and more policies become statistically indistinguishable and can be pruned. For all of these cases, the sign test found that there was no significant difference between the quality of the policies learned by AO* and learned by AO*+SP.

We performed a scaling-up experiment in which we increased the number of binary attributes to 20 with three different training set sizes and three different test costs c (and a memory limit of 10^6 nodes). For each of the 9 settings, we had 10 training sets. In 89 of the 90 runs, the policy constructed by AO*+SP was equal to or better than AO*. We believe that the AO*+SP policies were close to optimal (although the problems are too large to verify this by systematic search).

Table 1. Diabetes data. Confidence interval for the difference in (AO* $-$ SP) values on the test set.

# ex	95% CI	result
537	[0.228, 0.855]	AO*+SP better than AO*
268	[–0.515, 0.070]	AO*+SP and AO* the same
134	[–0.399, 0.511]	AO*+SP and AO* the same

4.2 Diabetes Synthetic Problem

Our second problem is based on the Pima Indians Diabetes dataset, donated by Vincent Sigillito, with attribute costs donated by Peter Turney, which can be found at the UCI repository. The data consists of 768 examples with no missing values, 8 attributes, and two classes (healthy or diabetic). 500 of the examples (65.10%) are labeled as healthy (class 0). Test costs are 1, 17.61, and 22.78. We assigned a cost of 100 to false negative diagnoses, and a cost of 80 to false positive diagnoses. Because our current implementation works for binary attributes, we first discretized the diabetes data into two levels by computing, for each attribute, the threshold value that maximizes the mutual information with the class variable.

Since the dataset is quite small, we measured performance by generating 100 random splits of the data into 70% training and 30% test examples (with sampling stratified by class). For each split of the diabetes data, the training set had 537 examples. To generate learning curves, we further subdivided each training set to produce training sets of size 268 and 134.

Both AO* and AO*+SP were trained on each of these training sets, and their performance was measured on the corresponding test set. The confidence level for

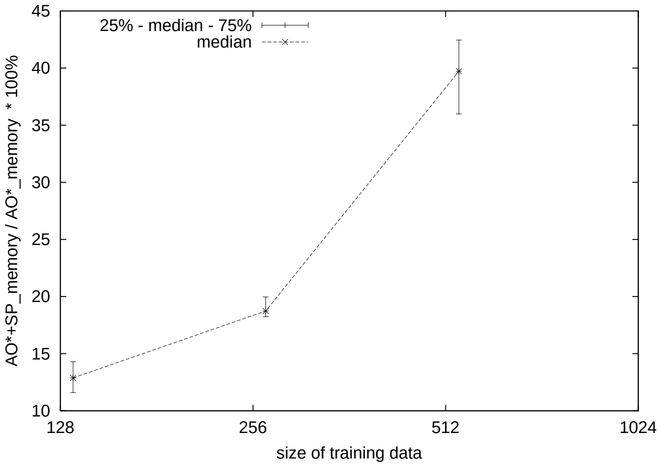

Figure 6. Diabetes data. Percentage of AO* memory used by AO*+SP.

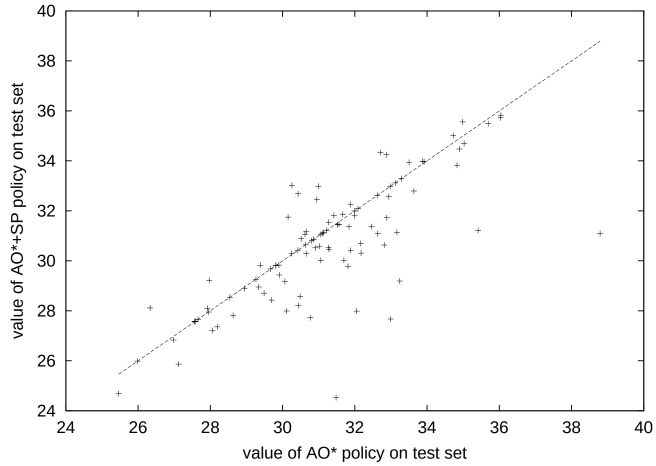

Figure 7. Diabetes. Scatter plot of the values of AO* and AO*+SP policies for 100 splits, each with 537 examples.

the statistical pruning heuristic was set at 99.9999%.

Figure 6 shows the results. In all cases, AO*+SP uses much less memory than AO*. Again, we observe that statistical pruning is especially effective for small training sets, but even with the large training set, it requires half the memory of AO*. The average CPU times are 16.45s for AO* and 29.78s for AO*+SP.

We applied a paired-difference test to determine whether the policy learned by AO*+SP was worse than that learned by AO*. Table 1 shows that for training sets of size 134 and 268, there was no significant difference. However, for the large training set, AO*+SP produced a policy significantly better than the AO* policy (given the same amount of memory). For this case, Figure 7 shows a scatter plot of the values on the test set of the AO* and AO*+SP policies for the 100 data splits of diabetes. Since half of the points are under the diagonal $y = x$ (AO*+SP is better than AO*), a fourth are on the diagonal (ties) and the rest are above the diagonal (AO*+SP is worse than AO*, but not markedly worse), this graph illustrates that AO*+SP performs better than AO* more often than not. By reducing the search space, AO*+SP overfits less than AO* and this leads to a reduction in cost.

5. Conclusions

Systematic search of the space of decision trees is infeasible in ordinary supervised learning. This paper has shown that in cost-sensitive learning, systematic search (using AO*) has the potential to be feasible when combined with an admissible heuristic and the statistical pruning heuristic. These heuristics give massive reductions in the search space without hurting the quality of the resulting diagnostic policy. Furthermore, statistical pruning can sometimes improve the diagnostic policy by reducing overfitting.

Acknowledgements. This research was supported by NSF IIS-0083292. We also thank Peter Turney for helpful discussions.

References

Greiner, R., Grove, A., & Roth, D. (2002). Learning cost-sensitive active classifiers. *Art. Int. Journal*, *in press.*

Hansen, E. (1998). Solving POMDPs by searching in policy space. *UAI-98* (pp. 211–219). Morgan Kaufmann.

Margineantu, D. (2001). *Methods for cost-sensitive learning.* Dissertation, Oregon State Univ.

Nilsson, N. (1980). *Principles of artificial intelligence.* Palo Alto: Tioga Publishing Co.

Norton, S. W. (1989). Generating better decision trees. *IJCAI-89* (pp. 800–805). Morgan Kaufmann.

Nunez, M. (1991). The use of background knowledge in decision tree induction. *Mach. Learn., 6*, 231–250.

Tan, M. (1993). Cost-sensitive learning of classification knowledge and its applications in robotics. *Mach. Learn., 13*, 1–33.

Turney, P. D. (1995). Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *JAIR, 2*, 369–409.

Washington, R. (1997). BI-POMDP: Bounded, incremental partially-observable Markov-model planning. *Proc. 4th Eur. Conf. Plan.*