Submitted to the
Articial Intelligence journal,
August 2000.

Learning Active Classifiers*

Russell Greiner
Department of Computing Science
University of Alberta
Edmonton, Alberta T6G 2H1
greiner@cs.ualberta.ca

Adam J. Grove
44 Murray Place
Princeton NJ 08540
grove@pobox.com

Dan Roth
Department of Computer Science
University of Illinois - Urbana/Champaign
Urbana, Illinois 61801
danr@cs.uiuc.edu

Abstract

Most classification algorithms are "passive", in that they assign a class-label to each instance based only on the description given, even if that description is incomplete. By contrast, an *active* classifier can — at some cost — obtain the values of missing attributes, before deciding upon a class label. This can be useful when considering, for example, whether to extract some information from the web for a critical decision or whether to gather information for a medical test or experiment. The expected utility of using an active classifier depends on both the cost required to obtain the additional attribute values and the penalty incurred if the classifier outputs the wrong classification. This paper analyzes the problem of *learning* optimal active classifiers, using a variant of the probably-approximately-correct (PAC) model. After defining the framework, we show that this task can be achieved efficiently when the active classifier is allowed to perform only (at most) a constant number of tests. We then show that, in more general environments, the task is often intractable.

Keywords: active classification, missing attributes, decision theory, PAC-learnability

*This extends the short conference paper [GGR96].

1 Introduction

A *classifier* is a function that assigns a class label to an instance. For example, given information about a credit-card applicant, a classifier could decide whether the person is a good risk and so should receive a credit card. Similarly, given information about a patient (such as symptoms and test values), a diagnostic classifier might specify the disease; given a visual scene of the world, a visual classifier might decide what object is being depicted; given a sentence, a context sensitive classifier might infer that the word "it" was intended to be "in", as in "`The man it the park`", etc. Most classifiers have no control over how much data they see. A more versatile classifier, however, might first seek additional information about the instance before deciding upon a classification. As obtaining data usually involves costs — *e.g.*, to perform a medical test, to run a specialized image processor or to run some additional processing such as partial parsing on a sentence — a classifier should not necessarily request all possible pieces of information. We therefore define an *active classifier* as a function which, given a partially specified instance, returns either a class-label or a strategy that specifies which test should be performed next (and recurs).

To make this more concrete, suppose that a medical classifier is initially told only that a patient is jaundiced – *i.e.*, her eyes are yellowish. A passive classifier must then return either the diagnosis that the patient has hepatitis, or the diagnosis that she does not. An active classifier could return either of these responses, or it could perhaps follow a strategy: order a blood test, and if that test is positive, return the diagnosis "hepatitis", but if the blood test is negative, then order a liver biopsy, and decide on the diagnosis based on the result of this test. Many other classification situations fit perfectly in this model. For example, a context sensitive text analyzer could make a decision based on the raw information available in the text, use more information such as part-of-speech tags [GR99] or, when the decision seems to be more difficult, choose to perform partial parsing of the sentence [EZR00] to improve its accuracy. Similarly, credit card companies, in deciding whether to give special deals to "accommodate" certain customers, must decide whether it is worth the expense of gathering information about those customers (*e.g.*, by sending out questionnaires, thoroughly investigating their previous spending patterns, etc.). An "active vision" system may also deal with this situation, in two senses: First, an image analyzer must decide which operators to use in analyzing a single pose [SUB96, CAL93]; and second, a camera platform on a mobile robot must decide whether it is worth the expense of moving the camera to obtain a better view [BJ98].

In the standard learning paradigm, a classifier is considered good precisely if it correctly identifies the class label for as many of the instances as possible. This measure is too simplistic for active classifiers. Here, the correct measure must be decision theoretic, balancing the costs of acquiring addition information against the penalties for incorrect classification. For instance, it may not be worth spending \$1,000 to perform an expensive test to distinguish two minor variants of hepatitis, especially if the treatment is the same for both [PP91]; similarly, it is not appropriate to spend \$100 to obtain the information required to win a \$1 bet.

When dealing with any single instance, an active classifier α must pay a *total cost*, defined as the sum of the penalty (if the answer returned is wrong) plus all costs incurred. Ideally,

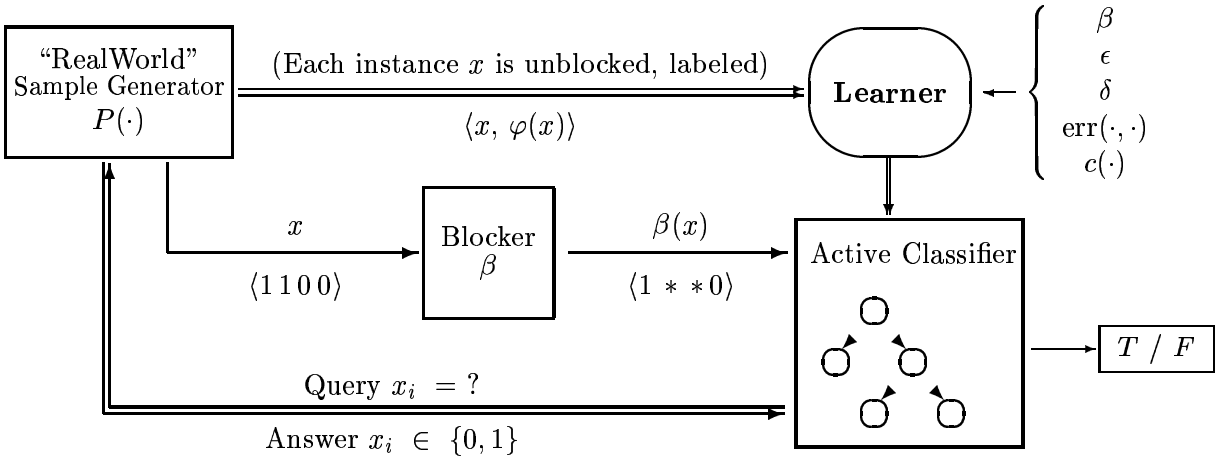

Figure 1: Blocking Model

we would like to find an active classifier whose *expected total cost*, over the distribution of instances that the classifier encounters, is minimum.

This paper investigates the task of learning such active classifiers. The most distinctive aspect of our proposal is that we look at the problem of learning active classifiers in an *integrated* fashion, as opposed to the "two stage" approach, of first learning the underlying concept and then, in a separate phase that does not involve learning, finding the best active classifier. After formally defining our framework in Section 2, we explain this idea and argue, in Section 3, that it has the potential to improve over the two-phase approach. The rest of the paper investigates whether this potential can be realized; the news here is mixed.

Section 4 demonstrates an interesting case in which active classifiers can be learned efficiently; then Section 5 proves that the general problem is very often intractable. The current model charges the classifier for each value it request, but gives the learner this information for free. Section 6 considers the slightly different "on-line" learning model, which charges the *learner* for answering questions during the learning process. Section 7 contrasts our approach with previous related work; in particular, it points out that our results on learning active classifiers is different from "active learning", which deals with learners that actively learn *passive* classifiers. Section 8 concludes with some ideas for future work and some thoughts on the contrast between active learning and passive classifiers. The appendix provides proofs for the theorems presented in this paper. (The text sketches proofs for the propositions.)

2 Framework

To simplify the presentation, we make the fairly standard assumption that all attributes, as well as the classification itself, are binary.[1] Thus we can identify each domain instance with a

[1] Extensions to non-binary attributes and class-labels are straightforward. In particular, in natural problems the appropriate classification can be a member of some arbitrary (finite) set. Note that even if the concept is conceptually "binary", it is often useful to have an "I-don't-know" option available to the classifier,

finite vector of Boolean attributes $x = \langle x_1, \ldots, x_n \rangle$. Let $X = \{0,1\}^n$ be the set of all possible domain instances. We assume the world corresponds to a concept φ, which we view as an indicator function $\varphi : X \mapsto \{T, F\}$, where $x \in X$ is a member of φ iff $\varphi(x) = T$. We assume that the learner knows the set of possible concepts, $\mathcal{C} = \{\varphi_i\}$, as well as a representation scheme for these concepts.

A (labeled) example of a concept $\varphi \in \mathcal{C}$ is a pair $\langle x, \varphi(x) \rangle \in X \times \{T, F\}$. We assume there is a stationary distribution $P : X \mapsto [0,1]$ over the space of domain instances, according to which random instances are drawn independently, both during training and testing of the learning algorithm.

To continue the earlier example, suppose the first attribute x_1 in the instance $x = \langle x_1, x_2, x_3 \rangle$ corresponds to the jaundice test and x_2 and x_3 correspond (respectively) to particular tests of the patient's blood and liver. Then the instance $\langle 1, 0, 1 \rangle$ corresponds to a patient whose blood test was negative, but whose jaundice and liver tests (x_1 and x_3) were both positive. Assume that the concept associated with hepatitis corresponds to any tuple $\langle x_1, x_2, x_3 \rangle$ where $x_1 = 1$ and either $x_2 = 1$ or $x_3 = 1$; *i.e.*, $Hep(\langle x_1, x_2, x_3 \rangle) \equiv x_1 \wedge (x_2 \vee x_3)$. Hence labeled examples of the concept hepatitis include $\langle \langle 1, 0, 1 \rangle, T \rangle$, $\langle \langle 1, 0, 0 \rangle, F \rangle$, and $\langle \langle 0, 1, 1 \rangle, F \rangle$. Further, $P(x)$ specifies the probability of encountering a patient with the particular set of symptoms specified by x; *e.g.*, $P(\langle 1, 0, 1 \rangle) = 0.01$ means 1% of the time we will deal with a patient with positive jaundice and liver tests, but negative blood test. (Notice we are assuming that class assignments are deterministic; *e.g.*, every $\langle 1, 0, 1 \rangle$ patient has hepatitis. It is straightforward to extend our analysis to stochastic assignments — *e.g.*, where say 90% of the patients with this set of symptoms have hepatitis.)

The above description implicitly suggested that the classifier has to "pay for" the value of each attribute it sees, as the classifier initially sees "nothing" (read "$\langle *, *, \ldots, * \rangle$"). In some situations, however, the classifier will initially see some of the attribute values for "free". (*E.g.*, this information may be available from an already completed questionaire, or from a low-level feature extractor that is always run.) In general, we assume there is a separate *blocking process* β, which stochastically selects some element of $X^* = \{0, 1, *\}^n$, to show the classifier for free. We will often consider the special case where the classifier initially sees no attribute values, *i.e.*, when it sees $\langle *, *, \ldots, * \rangle$. We call this *complete blocking*. (See [SG94] for a more general discussion of the blocking model.) Of course, under any blocking model, the classifier is subsequently allowed to obtain (at a price) the values of the remaining blocked attributes.

Definition 1 (Active Classifier) *An active classifier* $\alpha : \{0, 1, *\}^n \mapsto \{T, F, 1, 2, \ldots, n\}$ *is a function that maps a partial instance* $\langle x_1, x_2, \ldots, x_n \rangle \in \{0, 1, *\}^n$ *to one of* $\{T, F, 1, \ldots, n\}$, *where* $\alpha(\langle x_1, x_2, \ldots, x_n \rangle) = T$ *(resp., F) means the classifier returns the categorical answer T (resp., F). Returning* $\alpha(\langle x_1, x_2, \ldots, x_n \rangle) = i \in \{1, \ldots, n\}$ *means the classifier is requesting the value of the* x_i *attribute. (Once that value is provided, the active classifier then recurs.)*

Hence, continuing with the example above, $\alpha(\langle 1, *, * \rangle) = 2$ means the classifier is requesting the value of x_2 — *i.e.*, asking for the results of performing the blood test on the patient. The classifier then calls itself on the result, say $\langle 1, 0, * \rangle$, perhaps to return $\alpha(\langle 1, 0, * \rangle) = F$. In

in addition to T and F.

general, of course, yet other subsequent calls to α might be necessary (requesting the values for several variables) before a final answer is produced.

The learner's task may be to find the optimal active classifier amongst the set of all possible classifiers $\mathcal{A}^{all}$, or to find the best classifier of some particular type $\mathcal{A} \subset \mathcal{A}^{all}$. In general, $\mathcal{A}$ should be viewed as a particular "programming language" — a representation language and computational model for some (possibly restricted) class of active classifiers. We can talk of the size of an active classifier, $|\alpha|$, as well as its running time (*i.e.*, the time α requires, given input $x^* \in \{0,1,*\}^n$, to output its recommendation). Naturally, we are interested in finding classifiers whose size is polynomial in the relevant quantities (such as $|\varphi|$, the size of the true concept). Furthermore, we want active classifiers that are "fast" to execute. As any particular active classifier only has finitely many inputs, we cannot speak of its asymptotic execution time in the sense of standard complexity theory. We can, however, impose the requirement of efficient execution indirectly, as a property of the computational model given by $\mathcal{A}$.[2] In this paper we restrict attention to subclasses $\mathcal{A}$ with the property that

> there is some fixed polynomial $p_{\mathcal{A}}(\cdot)$ such that,
> for all $\alpha \in \mathcal{A}$, the running time of α is at most $p_{\mathcal{A}}(|\alpha|)$.

The question of how best to represent classifiers is a subtle one, and largely beyond the scope of this paper. In the following we occasionally refer to a very simple *lookup-table* representation language. In this, one simply lists the classifier's recommendations for various tuples $x^* \in \{0,1,*\}^n$; if the classifier encounters a tuple that is not on the list, it performs some constant action (perhaps announce the classification "F"). The size of an active classifier thus represented is just the length of the given list, and the run-time complexity of using such a classifier is at most linear in its size.

We will later consider the following class of classifiers that are allowed to be active at most $k-1$ times.

Definition 2 *For any constant $k \in \mathcal{N}$, $\mathcal{A}^k$ is the class of all active classifiers α such that $\alpha(x^*) \in \{T,F\}$ whenever x^* has k specified values (i.e., n-k $*$'s). Hence, each $\alpha \in \mathcal{A}$ has the option of performing additional tests on observing a partial instance that specifies strictly under k attributes. For completeness, we assume α gives a constant response (e.g., F) if there are more than k specified values.*

Notice that the size of any such classifier, in the lookup-table representation, is at most

$$g(k) \quad \triangleq \quad \sum_{i=0}^{k} \binom{n}{i} 2^i \quad = \quad O(\, k(2n)^k \,) \tag{1}$$

as this bounds the number of distinct partial instances in which the classifier must perform an action (either return a truth-value, or perform a test).

[2] This is very similar to the standard PAC requirement that the output representation can be evaluated efficiently. *E.g.*, our active classifier can be a polynomial time circuit, but cannot be an arbitrary polynomial-sized Bayesian network.

Classifiers in $\mathcal{A}^k$ are actually quite realistic. To motivate this situation, consider a time-critical task, where a classification returned after k seconds is useless — perhaps because we know the patient will be dead by then, or the part on the conveyer belt that needs to be classified will be beyond the range of the mechanical sorter. In text, speech and image processing situations, limiting the amount of time or scope of additional features computed before a decision is made is reasonable if one wants to bound processing delay to allow for a perception of real-time behavior.

2.1 Evaluating an Active Classifier

To evaluate the quality of an active classifier, we assume as given a cost function $c_i = c(i) \in \Re$ (for $i = 1 \ldots n$) that specifies the cost of obtaining the value of the i^{th} attribute x_i; and a penalty function $\mathrm{err}(v_1,\ v_2)$, which specifies the penalty for returning $v_1 \in \{T,\ F\}$ when the correct answer is $v_2 \in \{T, F\}$. Without loss of generality, we can assume $\mathrm{err}(T,\ T) = \mathrm{err}(F,\ F) = 0$. To avoid degeneracy, we also assume that $\mathrm{err}(T,\ F) > 0,\ \mathrm{err}(T,\ F) > 0$. Further, as it never makes sense to perform a test whose cost c_i exceeds

$$err_M \quad \stackrel{\Delta}{=} \quad \max\{\ \mathrm{err}(T, F),\ \mathrm{err}(F, T)\ \} \tag{2}$$

we will assume $c_i \leq err_M$ for all i.[3]

Suppose that $x^* \in \{0, 1, *\}^n$ represents the classifier's current knowledge about the instance $x \in X$. We define the "total cost" $tc_\alpha(x, x^*) \in \Re$ to be the amount that α would spend to complete the classification, together with the misclassification penalty, if appropriate. We then define $tc_\alpha(x) = tc_\alpha(x, \langle *, *, \ldots * \rangle)$ as the total cost when the classifier begins with a completely blocked instance.

As suggested by the notation, we assume the $tc_\alpha(x, x^*)$ cost function is "time independent" — *i.e.*, the cost of requesting a value of an attribute is independent of when it is done. This means the value $tc_\alpha(x, x^*)$ can be determined recursively: If $\alpha(x^*) \in \{T, F\}$, then $tc_\alpha(x, x^*) = \mathrm{err}(\alpha(x^*),\ \varphi(x))$ where φ is the target formula. Otherwise, if $\alpha(x^*) = i \in \{1, \ldots, n\}$, then $tc_\alpha(x, x^*) = c(i) + tc_\alpha(x, x^*_{i \mapsto x})$ where $x^*_{i \mapsto x}$ is the result of setting the value of the variable indexed by $i = \alpha(x^*)$ to $x[i]$.

As an example, suppose $x = \langle 1, 0, 1 \rangle$, $x^* = \langle *, *, * \rangle$ (*i.e.*, we have not asked for any attribute's value yet), and $\alpha(x^*) = 2$. Then $x^*_{2 \mapsto 0} = \langle *, 0, * \rangle$, as it sets the x^*_2 value with $x[2] := 0$. If we suppose further that $\alpha(\langle *, 0, * \rangle) = F$, and $\varphi(\langle 1, 0, 1 \rangle = T$, then

$$\begin{array}{rcl}
tc_\alpha(\langle 1, 0, 1 \rangle) & = & tc_\alpha(\langle 1, 0, 1 \rangle,\ \langle *, *, * \rangle) \\
= \ c[\alpha(\langle *, *, * \rangle)] & + & tc_\alpha(\ \langle 1, 0, 1 \rangle,\ \langle *, *, * \rangle_{2 \mapsto 0}\ ) \\
= \qquad c(2) & + & tc_\alpha(\ \langle 1, 0, 1 \rangle,\ \langle *, 0, * \rangle\ ) \\
= \qquad c_2 & + & \mathrm{err}(\ \alpha(\langle *, 0, * \rangle),\ \varphi(\langle 1, 0, 1 \rangle)\ ) \\
= \qquad c_2 & + & \mathrm{err}(\ F,\ T\ )
\end{array}$$

We define the *expected total cost* of the active classifier α as the expected value of the $tc(\ x)$ under the distribution of instances x, $P(\cdot)$, and the blocker β (β is omitted from the

[3] In fact, we may assume that $c_i \leq \min\{\mathrm{err}(T, F),\ \mathrm{err}(F, T)\ \}$; we use max as it is easier in several proofs.

notation):

$$EC_P(\alpha) \quad = \quad E_{x\in P}[\, tc_\alpha(x)\,] \quad = \quad \sum_{x\in X} P(x) \times tc_\alpha(x)\ .$$

We assume, for now, that a learning algorithm L can draw random correctly-labeled completely-specified examples $\langle x,\, \varphi(x)\rangle$ according to the distribution P. One justification for allowing the learner to train on *complete* instances, even though the classifier will see only *partial* instances, is that it can be cost-effective to invest in a relatively expensive training phase, if we expect that the active classifier we learn will be used very often. In this case, the cost of obtaining all attributes while learning, amortized over a much longer performance phase, might be insignificant. In Section 6, we see this is not a serious restriction, by showing that we get similar results, *mutatis mutandis*, when considering the more general case, where the learner must pay to see each attribute.

Here, we evaluate the learner L in terms of the expected total cost of its computation and output, α. For any such $\varphi \in \mathcal{C}$ and $\mathcal{A}$, let $\alpha_{\varphi,\mathcal{A},P} \in \mathcal{A}$ be an active classifier whose expected total cost is minimum among active classifiers in $\mathcal{A}$:

$$\alpha_{\varphi,\mathcal{A},P} \quad = arg\min\{\, EC_P(\alpha) \,|\, \alpha \in \mathcal{A}\,\} \tag{3}$$

(When the dependence on $\mathcal{A}$ and P is clear, we will write α_φ rather than $\alpha_{\varphi,\mathcal{A},P}$.)

We define the following *Probably Approximately aCTive learner*, a variant of the standard "Probably Approximately Correct" (PAC) criterion [Val84, KLPV87] to specify the desired performance of such a learner.

Definition 3 (PACT-Learning) *Given a set of concepts $\mathcal{C}$ defined over X, a probability distribution P over X, a blocker β, a cost function $c(\cdot)$, and a penalty function $err(\cdot,\cdot)$, we say that an algorithm L* PACT-learns *a set of active classifiers $\mathcal{A}$ (with respect to $\mathcal{C}$, P, β, $c(\cdot)$ and $err(\cdot,\cdot)$) if, for some polynomial function $p(\cdots)$, for any target concept $\varphi \in \mathcal{C}$, distribution P and error parameters $\epsilon, \delta > 0$, L runs in time at most $p(\frac{1}{\epsilon}, \frac{1}{\delta}, |\varphi|)$, and outputs an active classifier*

$$\alpha = L(\epsilon,\, \delta,\, \mathcal{C},\, \mathcal{D},\, c(\cdot),\, err(\cdot,\cdot)) \in \mathcal{A},$$

whose expected total cost is, with probability at least $1-\delta$, no more than ϵ over the minimal possible expected total cost; i.e.,

$$\forall \varphi \in \mathcal{C},\ \epsilon, \delta > 0, \quad P(\ \mathrm{EC}_P(\alpha) > \mathrm{EC}_P(\alpha_{\varphi,\mathcal{A},P}) + \epsilon\ ) \quad \leq \quad \delta\ . \qquad \blacksquare$$

Assuming the learning algorithm L and the active classifier α are deterministic, the definition only makes use of a fixed (but unknown) probability distribution P, which governs the occurrences of instances. (If the blocker is stochastic, we need to consider that distribution as well.) Note also that the number of instances drawn by α can be no more than the running time, and thus is polynomial in $\frac{1}{\epsilon}, \frac{1}{\delta}, |\varphi|$. Similarly, the size of the learned classifier, $|\alpha|$, is bounded by the learner's running time, and so is polynomial as well. Using the requirement that $\mathcal{A}$ includes only classifiers whose execution time is time polynomial in their size, we see that α's run-time is also polynomial in $\frac{1}{\epsilon}, \frac{1}{\delta}, |\varphi|$.

One restriction inherent in this model is that the value of requested attributes are reported sequentially (as opposed to requesting a group of tests to be performed concurrently [Tur95]).

A second restriction is that the classifier makes its decisions based only on the *values* of attributes it knows — not on how these values were obtained (*i.e.*, were they visible initially, or did the classifier have to ask for them?)[4].

The class of active classifiers, $\mathcal{A}$, is an important component of the above definition. Not only it is often useful to consider only a subset of the class of all possible active classifiers, perhaps to facilitate learning the classifier, and/or to insure that the classifier produced is computationally efficient, we should also choose a suitable representation language. Explicitly enumerating the actions of a classifier (*i.e.*, viewing it as a function $\alpha: X^* \mapsto \{T,\ F\} \cup X$) requires exponential space, even for such conceptually trivial classifiers such as the degenerate classifier that always returns F. We already mentioned the "lookup-table" encoding, which addresses this problem by specifying a *default* action (*e.g.*, announce that every unexceptional instance has the label F) and only enumerate exceptions to this default. For concreteness, we have this representation language in mind throughout this paper unless we specify otherwise, although none of our results depend critically on this particular choice.

There are a number of more general approaches that might be used to specify $\mathcal{A}$. An active classifier α can be viewed as making a number of binary decisions based on the current input $x^* \in X^*$: *E.g.*, should T be returned?; if not, should F be returned?; if not, should we ask for x_1?; etc. That is, we can write $\alpha \approx \langle e_T, e_F, e_1, \ldots, e_n \rangle$, where each $e_i : X^* \mapsto \{1,\ 0\}$, with the understanding α's action on $\langle x_1, x_2, \ldots, x_n \rangle$ is the index of the first of these e_i which evaluated to 1; $\alpha(\langle x_1, x_2, \ldots, x_n \rangle) = arg\min_i\{e_i(\langle x_1, x_2, \ldots, x_n \rangle) = 1\}$. Using this representation, we can then restrict each e_i to belong to some specified collection of Boolean concepts $\mathcal{E}$ over $\{0, 1, *\}^n$ (*á la* [Rot95]). Although we do not do so in this paper, it would be interesting to investigate the connection between learnability of active classifiers thus specified, and standard PAC-learnability of the classes $\mathcal{E}$.

3 Why Should We *Learn* Active Classifiers?

The optimal active classifier is determined by the concept φ, the set of active classifiers $\mathcal{A}$, and the distribution P. (In general, we will assume that the blocker β is known.) If we know all of these components, then we are faced with a very interesting optimization problem [HBR94] — one which, however, has nothing to do with learning. Sometimes this problem is tractable as, for instance, in the following case involving product distributions (*i.e.*, distributions in which the value of each attribute is determined independently) and classifiers that can only ask a constant number of questions.

[4] Although we do not do so here, the cost model could be generalized to allow "context dependent" costs, where the cost of obtaining attribute i might depend on what other attributes have already been requested. For example, in medical diagnosis there may be a fixed cost associated with drawing blood which should be charged only to the first test requiring a blood sample. (Here, the second, and subsequent, blood tests would be charged only the specific test performed, but not for extracting the blood [GO96].) This extension would, however, force a corresponding generalization to our concept of classifier: If $c(i)$ depends on which tests have previously been performed, it is not sufficient to act based on the values of known attributes — it is also relevant to know *how* we learned about previous attributes (*i.e.*, did we request the test, or was it initially unblocked?).

Proposition 4 *Suppose we know the concept φ and the* product *distribution over instances P, and are considering only the set $\mathcal{A}^k$ of active classifiers that ask for at most k attribute values (Definition 2). Then, for any $\epsilon > 0$, there is an efficient algorithm that runs in time $O(poly(n, \frac{1}{\epsilon}))$ and produces an active classifier whose expected total cost is within ϵ of the optimal active classifier in $\mathcal{A}^k$.*

Proof: Given φ and the product distribution P, the classifier can determine the probability of T versus F for any given (partially specified) instance and thus determine which is the better response. In general, it can also determine whether it should ask for the value of an attribute using straightforward *dynamic programming*; see proof of Theorem 7 (from Section 4). ∎

This suggests an obvious way to learn an active classifier: first learn the optimal underlying (passive) classifier φ and the distribution P, and then combine these to produce the best active classifier $\alpha_{\varphi,\mathcal{A},P}$. While Proposition 4 shows that this "learn then optimize" approach can sometimes works, there are problems. First, and unsurprisingly, the optimization problem can be intractable:[5]

Theorem 5 *There exists choices for $\mathcal{A}$, P, $\mathcal{C}$ such that it is NP-hard to find the optimal $\alpha^* \in \mathcal{A}$. is NP-hard. This result holds even if we further require that $|\alpha^*|$ is polynomial in $|\varphi|$ (i.e., the complexity is not simply because a very large classifier is needed), that $\mathcal{C}$ be PAC-learnable, and that P has support of size $O(n)$, where n is the number of attributes.*

The result above shows that the complexity of classifying actively is, in a sense, "independent" of the complexity of learning. Learning the concept and/or the distribution poses separate problems:

Proposition 6 *There are some concept classes $\mathcal{C}$ (together with $\mathcal{A}$, P, $err(\cdot,\cdot)$, $c(\cdot)$) such that finding the optimal active classifier is trivial if we are given $\varphi \in \mathcal{C}$, but otherwise is not known to be possible.*

Proof: This claim reduces to the fact that not everything is known to be PAC-learnable [Ang92] because, if all costs $c(x_i)$ are zero, the classifier can ask for all attributes and then it will classify optimally if and only if it can identify the concept. ∎

The preceding claims (Proposition 4, Theorem 5 and Proposition 6) show that, while the "learn then optimize" approach is certainly *sufficient* (in principle) to determine α_φ, it can fail (for complexity reasons) in various ways. *The paper's main point, however, is that it may be easier to simply learn the active classifier directly.* In particular, one can sometimes learn a good active classifier *without* having learned (even implicitly) the concept or the distribution. This basic idea — of learning just enough to perform some particular task, rather than trying to learn everything — has been used by Khardon and Roth [KR97] in their *Learning to Reason* framework. They are concerned with the task of logical reasoning (rather than classification), and show that there can be significant complexity advantages in directly learning a representation tailored to a particular reasoning task (rather than

[5] Recall proofs of the theorems appear in the Appendix.

trying to learn the concept itself and then, in a separate phase, perform logical deduction). Our work shares very much the same philosophy (if none of the technical underpinnings) as Learning to Reason.[6]

When might it be a good idea to learn the active classifier directly? Our main positive result, given in Section 4, provides one answer in detail. Below are some of the underlying general issues:

• We do not always have to learn the full concept. For example, suppose $\mathrm{err}(\cdot,\cdot)$ and $c(\cdot)$ are such that it is never worthwhile asking more than one question. Then the optimal active classifier is completely determined once we specify which single attribute we should request, and which classification (T or F) is most likely given each value that this attribute might take(forming "decision-stumps" of the form studied in [Hol93, AHM95]). We can sometimes learn this classifier without knowing the full concept itself. Of course, knowing the full concept *would* be important *if* we were frequently asked to classify complete (unblocked) instances. But this is simply irrelevant: as we know that the instances will be presented completely blocked, we know that such questions will not in fact be asked. We should only care about cases that we actually might encounter (with high enough probability).

• We do not always need to learn the complete distribution P. The same example shows that, in some cases, only a few aspects of the distribution may be relevant: here we only need to know correlations between single attributes and the class label. Higher order correlations (*i.e.*, involving more than one attribute) do not affect the optimal active classifier.

• There is a second reason why we might not need to learn the distribution. The standard PAC-learning framework usually avoids having to learn distributions, because the performance criterion uses the same distribution that one learns under. If one has a (passive) classifier that fits the sample data well enough, one may hope that it will perform well on other data from the same distribution. We do not necessarily need to know *what* that distribution is; only that it has not changed since the learning phase. As our definition of PACT-learning is similar to the standard PAC formulation in this respect, it too might avoid the need to learn distributions.

Of course, these arguments are only suggestive. Section 5 below will show several significant limitations on what can be achieved. However, we first present a fairly simple, yet worthwhile, positive result.

4 Learning $\mathcal{A}^k$

This section presents an expressive class of problems for which we can efficiently learn the optimal active classifiers. The results depend on restricting the set of active classifiers considered to $\mathcal{A}^k$, those classifiers that request at most a constant, k, attribute values (Definition 2).

We show that it is possible to PACT-learn *any* concept class $\mathcal{C}$ under *any* distribution. In particular (in this situation), we can learn to actively classify with respect to concepts and distributions that are not learnable in the pure PAC-learning sense!

[6] See also the "Learning accurate belief nets" work [GGS97].

Algorithm $L^{(k)}(\ \epsilon\ \in \Re^+,\ \delta\ \in (0,1)\ ):\ \ \alpha\ \in \mathcal{A}^k$

% Returns an active classifier α from $\mathcal{A}^k$ whose expected cost is, with probability at least $1-\delta$,
% within ϵ of optimal
% Uses oracle for drawing complete labeled instances, and knows cost model, $err(\cdot,\cdot)$, $c(\cdot)$

Draw $\frac{err_M^2 4^{2k+3}}{9\epsilon^2} \ln \frac{4\,g(k)}{\delta}$ completely-specified, labeled instances S
Return Helper_$L^{(k)}(\ S,\ \langle *, \ldots, * \rangle\ )$
End Algorithm $L^{(k)}$

Algorithm Helper_$L^{(k)}(\ S$: sample; x^{start} : instance$) : \alpha\ \in \mathcal{A}^k$

% Uses $\begin{cases} \widehat{\text{AFTER}}[\,\cdot\,]: & \text{a list of } g(k) \text{ real-numbers,} \\ Op[\cdot]: & \text{a list of } g(k) \text{ "operations"} \quad (\text{each } Op[j] \in \{T, F, 1, \ldots, n\}), \end{cases}$
% whose elements are "indexed" by partial assignments from $X^*_{0..k}$
% See text for definitions of $\widehat{P^{\varphi}_{x^*}}$ and $\widehat{P^{i\mapsto 0}_{x^*}}$

For each $\ell = k..0$ do
  For each $x^* \in X^*_\ell$ do *[L0]*
    $\widehat{C}(\,x^*, T\,) := (1 - \widehat{P^{\varphi}_{x^*}}) \times \texttt{err}(T,F)$ *[L1]*
    $\widehat{C}(\,x^*, F\,) := \widehat{P^{\varphi}_{x^*}} \times \texttt{err}(F,T)$ *[L2]*
    If $\ell < k$, then For each $i = 1..n$ where $x^*_i = *$ do
      $\widehat{C}(\,x^*, i\,) := c_i + \widehat{P^{i\mapsto 0}_{x^*}} \times \widehat{\text{AFTER}}[\,x^*_{i\mapsto 0}\,] + (1 - \widehat{P^{i\mapsto 0}_{x^*}}) \times \widehat{\text{AFTER}}[\,x^*_{i\mapsto 1}\,]$ *[L3]*
    $Op[x^*] := arg\min\{\ \widehat{C}(\,x^*, z\,) \mid z \in \{T, F, 1, \ldots, n\}\ \}$
    $\widehat{\text{AFTER}}[\,x^*\,] := \min\{\ \widehat{C}(\,x^*, z\,) \mid z \in \{T, F, 1, \ldots, n\}\ \}$
Return BuildTree($Op[\cdot]$, $\langle *, \ldots, * \rangle$)
End Algorithm Helper_$L^{(k)}$

Algorithm BuildTree($Op[\cdot]$: list_of_operations; $x^* \in X^*$): Decision_Tree
Let n be a new node
if $Op[x^*] = T$
  Label $n.Action$:= "Return True"
elseif $Op[x^*] = F$
  Label $n.Action$:= "Return False"
else *% Here, $Op[x^*] = i$*
  Label $n.Action$:= "Test x_i"
  Let $n.ifTrue$:= BuildTree($Op[\cdot]$, $x^*_{i\mapsto 1}$)
  Let $n.ifFalse$:= BuildTree($Op[\cdot]$, $x^*_{i\mapsto 0}$)
return(n)
End Algorithm BuildTree

Figure 2: The $L^{(k)}$ learning algorithm, for PACT-learning $\mathcal{A}^k$ under complete blocking

4.1 Learning $\mathcal{A}^k$ Under Complete Blocking

We start the presentation of the main result by considering first the case of "complete blocking" — *i.e.*, the classifier only sees the features it explicitly requests. The $L^{(k)}$ algorithm, shown in Figure 2, is capable of PACT-learning active classifiers in the set $\mathcal{A}^k$, given complete blocking, for any concept class and under any distribution.

We let $X^*_{0..k} = \cup_{i=0}^{k} X^*_i$, where X^*_m is the set of all blocked n-tuples with exactly m specified attributes (n-m $*$'s). Also, for any $x^* \in X^*_\ell$ whose i^{th} attribute has not been specified (*i.e.*, $x^*_i = *$), $y^* = x^*_{i \mapsto 0}$ is a partially-specified tuple with $\ell + 1$ specified values that extends x^* by setting $y^*_i = 0$. (*E.g.*, $\langle 1, *, * \rangle_{3 \mapsto 0} = \langle 1, *, 0 \rangle$.)

$L^{(k)}$ first draws a number of instances, which it uses to obtain estimates:

- $\widehat{P^{\varphi}_{x^*}}$, to estimate $P^{\varphi}_{x^*} = P(\, \varphi(x) = T \mid x \textit{ extends } x^* \,)$, which is the conditional probability that an instance drawn according to P and which coincides with x^* on its specified attributes, will be labeled T; and

- $\widehat{P^{i \mapsto 0}_{x^*}}$, to estimate $P^{i \mapsto 0}_{x^*} = P(\, x^*_{i \mapsto 0} \mid x^* \,)$, which is the conditional probability that an instance drawn according to P and which coincides with x^* on its specified attributes, will have its i^{th} attribute equal to 0.

(Stated more precisely: given the set of complete instances S, for each $x^* \in X^*_\ell$, let $\#[x^*] = \|\{x \in S \mid x \text{ extends } x^*\}\|$ be the number of instances in S that extend x^*, and $\#[\varphi(x^*) = T] = \|\{x \in S \mid x \text{ extends } x^* \;\&\; \varphi(x) = T\}\|$ be the number of instances in S that extend x^* and are labeled T. Then $\widehat{P^{\varphi}_{x^*}} = \#[\varphi(x^*) = T] \,/\, \#[x^*]$ is the empirical estimate of $P(\, \varphi(x^*) = T \mid x^* \,)$, and $\widehat{P^{i \mapsto 0}_{x^*}} = \#[x^*_{i \mapsto 0}] \,/\, \#[x^*]$ is the empirical estimate of $P(\, x^*_{i \mapsto 0} \mid x^* \,)$. If $\#[x^*] = 0$, then set $\widehat{P^{\varphi}_{x^*}}$ and $\widehat{P^{i \mapsto 0}_{x^*}}$ to $1/2$.)

Before dealing with sampling error, we first prove that, if these estimates are exactly correct (*i.e.*, $\widehat{P^{\varphi}_{x^*}} = P^{\varphi}_{x^*}$ and $\widehat{P^{i \mapsto 0}_{x^*}} = P^{i \mapsto 0}_{x^*}$), then the $L^{(k)}$ algorithm will in fact produce the optimal active classifier $\alpha^{opt} = \alpha_{\varphi, \mathcal{A}, P}$ as defined in Equation 3. This follows by observing that $L^{(k)}$ is just a straightforward *dynamic program*. Given our assumptions, we have to classify $x^* \in X^*_k$ after the classifier has already asked k questions, meaning $\alpha(\, x^* \,) \in \{T, F\}$. Now observe that $C(\, x^*, T \,)$ ($\widehat{C}(\,\cdot\,)$ is defined on line *[L1]*, based on $\widehat{P^{\varphi}_{\cdot}}$ values; $C(\,\cdot\,)$ is corresponding value based on $P^{\varphi}_{\cdot}$) is simply the cost paid for returning T for x^*; similarly for $C(\, x^*, F \,)$. Clearly the optimal α^{opt}, on encountering x^*, should take the smaller of these values.

Having decided what the α^{opt} classifier should do for $x^* \in X^*_k$, $L^{(k)}$ must then determine the correct actions for each element in X^*_{k-1} and then decide how to deal with each element is X^*_{k-2}, and so on, until reaching $X^*_0 = \{\langle *, *, *, \ldots, * \rangle\}$, thus completing the specification of the learned classifier α^{opt}.

To explain each step, suppose α^{opt} has decided what to do for all $x^* \in X^*_{k-i}$ ($i \geq 0$), and is considering some particular $y^* \in X^*_{k-(i-1)}$, with one more "$*$". Let $\textsc{Before}(\, y^* \,)$ be α^{opt}'s costs already incurred in reaching y^* (starting from $\langle *, \ldots, * \rangle$), and let $\textsc{After}(\, y^* \,)$ be the remaining costs; hence, if α^{opt} eventual strategy involves y^*, its cost will be $\textsc{Before}(\, y^* \,) + \textsc{After}(\, y^* \,)$. Here, α^{opt}'s possible actions are to announce a classification (*i.e.*, T or F) or

ask about a variable whose value is not yet known. The cost of announcing either T or F is the same as it was for $x_k^* \in X_k^*$. The expected cost of testing attribute x_i is:

$$\begin{array}{rcl} C(\, y^*, i\,) & = & \\ c_i & + & P(\, x_i = 1 \,|\, x \textit{ extends } y^*\,) \;\times\; \textsc{After}(\, y^*_{i\mapsto 1}\,) \\ & + & P(\, x_i = 0 \,|\, x \textit{ extends } y^*\,) \;\times\; \textsc{After}(\, y^*_{i\mapsto 0}\,) \end{array} \tag{4}$$

(See line *[L3].*) Note that the expected costs required by the last equation ($\textsc{After}(\, y^*_{i\mapsto 1}\,)$ and $\textsc{After}(\, y^*_{i\mapsto 0}\,)$), have been computed in the previous phase of the algorithm. As shown, the $L^{(k)}$ algorithm then simply assigns to α^{opt} the action ("`Return True`", "`Return False`", or "`Test` x_i") with the lowest expected costs.

To understand why this algorithm works, note that these $\textsc{After}(\, y^*_{i\mapsto 1}\,)$ costs depend only on the instance $y^*_{i\mapsto 1}$, and not on how (or even, if) α^{opt} would reach this instance. As such, this value is completely independent of $\textsc{Before}(\, y^*\,)$. This means that $L^{(k)}$ can compute the values of $\textsc{After}(\, y^*\,)$ in one sweep, from the most immediate (in X_k^*) back to the least (in X_0^*).

Of course, our $L^{(k)}$ algorithm does not have access to the actual probabilities. Here, it will use empirical estimates of $P^{\varphi}_{x^*}$, called $\widehat{P^{\varphi}_{x^*}}$ in Figure 2, and so produce a not-necessarily-optimal classifier α. What happens if these estimates are inexact — which they typically will be, due to statistical fluctuations? Suppose first the distribution is uniform, P_{uniform}, which means the proportion of training instances matching any $x^* \in X_i^*$ will be about $1/2^i$. Thus, in a reasonable number of instances, we can obtain good estimates of these quantities — *i.e.*, we expect $\widehat{P^{\varphi}_{x^*}} \approx P^{\varphi}_{x^*}$. If the basic "argmin decisions" are clear cut — *i.e.*, if $(1-P^{\varphi}_{x^*}) \times \text{err}(T,F)$ is far from $P^{\varphi}_{x^*} \times \text{err}(F,T)$ — then using $\widehat{P^{\varphi}_{x^*}}$ rather than $P^{\varphi}_{x^*}$ should not matter, as here $(1-P^{\varphi}_{x^*}) \times \text{err}(T,F)$ will be bigger than $P^{\varphi}_{x^*} \times \text{err}(F,T)$ iff $(1-\widehat{P^{\varphi}_{x^*}}) \times \text{err}(T,F)$ is bigger than $\widehat{P^{\varphi}_{x^*}} \times \text{err}(F,T)$. The only potential problems arise if $P^{\varphi}_{x^*}$ is near a threshold — *i.e.*, if $(1 - P^{\varphi}_{x^*}) \times \text{err}(T,F) \;\approx\; P^{\varphi}_{x^*} \times \text{err}(F,T)$ — as this could cause $L^{(k)}$ to make the wrong decision. But this is precisely when it does not matter much which decision we make, because the expected costs are nearly the same.

For distributions other than P_{uniform}, there may be some probabilities whose estimate will be wildly inaccurate, because the sample will include so few matching instances. But, by our PAC-like performance criterion, it does not matter much if we do badly on these extremely unlikely cases. We make these arguments precise in the appendix (when we give the proof for the algorithm, Theorem 7), but this is the basic idea underlying $L^{(k)}$'s correctness: Estimated payoffs are *good enough* in this setting, and although they may lead to a classifier whose recommendations differ from the optimal classifier, this only happens when the disagreement does not affect costs by much.

Theorem 7 *The algorithm $L^{(k)}$ PACT-learns active classifiers in the set $\mathcal{A}^k$ given complete blocking for* any *concept φ and* any *distribution P.*

4.2 Learning $\mathcal{A}^k$ under Arbitrary Blocking

While the $L^{(k)}$ algorithm shown only takes $\langle *, *, *, \ldots, * \rangle$ as its "starting pattern", it is easy to define a related algorithm that starts from *any* fixed pattern. Here, at each step, the

active classifier may ask the value of any currently-unspecified attribute; we continue to consider only classifiers that ask for the values of at most k additional attributes. To do this, we replace the X^*_ℓ on line *[L0]* with $X^*_\ell(x^{start})$, which denotes the set of all $O(2^\ell \binom{n}{\ell})$ instances formed by starting with starting instance $x^{start} \in X^*$ and specifying the values of exactly ℓ of its initially-uninstantiated variables. (Hence $X^*_\ell(\langle *, \ldots * \rangle) = X^*_\ell$.)

This basic approach similarly works whenever the classifier can encounter a fixed (or even polynomial) number of starting patterns, by just building a different "subclassifier" for each starting pattern. Next we investigate several more significant weakenings.

In general, there can be an arbitrarily large number of initial instances x^{start} — *e.g.*, we can consider blockers that can reveal completely arbitrary sets of attribute values initially, for free. The simple extension cannot handle this, as this would mean dealing with perhaps 3^n initial instances, and so require building an exponential number of different sub-classifiers (one for each starting instance).

However, a fairly simple modification of the $L^{(k)}$ algorithm will work. The main difference is that we will not use the explicit (lookup-table) representation scheme, as that would not be poly-size in this case. Instead, the learning algorithm will be a "lazy" learning algorithm (reminiscent of [Aha97]). In the learning stage, this learner simply records the instances seen during training, S. The resulting classifier would take the result of the learning (read "the sample S"), together with the specified starting instance x^{start}. It would then call Helper_$L^{(k)}(S, x^{start})$, to compute the appropriate actions to take, then begin performing the specific actions.

Notice we only estimate the relevant probabilities — the values of $P(\, \varphi(x^*) = T \,|\, x^* \,)$ and $P(\, x^*_{i \mapsto 0} \,|\, x^* \,)$, for each $x^* \in X^*_\ell(x^{start})$, $\ell = 0..k$ — after we know the current value of x^{start}. The only challenge is collecting a large enough sample, to insure that we can estimate these $2 \sum_{\ell=0}^{k} |X^*_\ell(x^{start})| = 2g(k)$ values, for any possible starting instance $x^{start} \in X^*$. Of course, it is sufficient to simply estimate these $P(\, \varphi(x^*) = T \,|\, x^* \,)$ and $P(\, x^*_{i \mapsto 0} \,|\, x^* \,)$ values for *every* possible $x^* \in X^*$ — all 3^n possible partially-specified instances.

This leads to the "lazy variant of $L^{(k)}$": The lazy-$L^{(k)}$ learner initially collects

$$M_{all} \quad = \quad \frac{err_M^2 4^{2k+3}}{9\epsilon^2} \ln \frac{4 \times 3^n}{\delta} \quad = \quad O\left(\frac{err_M^2}{\epsilon^2} [n + \ln \frac{1}{\delta}] \right) \tag{5}$$

instances, S. (Recall that k, and hence 4^{2k+3}, is a constant.) The subsequent classifier will then use this to actively-classify, starting with the starting instance x^{start}. It simply calls Helper_$L^{(k)}(S, x^{start})$ to find the appropriate active-classification-tree, then executes this tree.

Corollary 8 *The lazy-$L^{(k)}$ system PACT-learns active classifiers in the set $\mathcal{A}^k$* given any blocker *for* any *concept φ and* any *distribution P.*

Proof: Just observe (from proof of Theorem 7) that the classifier is effective whenever it can reliably (*i.e.*, with collectively probability at least $1 - \delta$) estimate the values of $P(\, \varphi(x^*) = T \,|\, x^* \,)$ and $P(\, x^*_{i \mapsto 0} \,|\, x^* \,)$ to within $\lambda = \frac{3\epsilon}{8\, err_M\, 4^k}$, for each $x^* \in X^*_\ell(x^{start})$. Using Hoeffding's Inequality (Equation 9), M_{all} instances is sufficient to estimate these quantities over *all* 3^n possible $x^* \in X^*$. ∎

To summarize, the extension to general blocking relies on two issues. First, the observation that a classifier in $\mathcal{A}^k$ only requires the algorithm to keep $O(n^k)$ probabilities implies that when dealing with classifier in $\mathcal{A}^k$ computational complexity is not a problem. The second issue deals with the sample size. Unlike the case of a fixed starting point there is, potentially, an exponential number of probabilities we might need to estimate, and the sample size required to guarantee good estimates would be larger. The crucial observation is that the increase in the sample size is only polynomial since the sample size depends only logarithmically on the number of probabilities required.

5 Further Extensions

5.1 Allowing the ActiveClassifier to Ask More Questions

The results in the previous section show that we can PACT-learn in general, provided that the active classifier is allowed to inquire about no more than a constant k additional queries. We might also hope to be able to weaken this restriction, by allowing the active classifier to ask, say, $O(\log n)$ attributes; this corresponds to learning the class $\mathcal{A}^{\log n}$. However, if this was possible, then we could PAC-learn $\log n$-depth decision trees in the standard (passive learning) model.[7] But even the simpler problem:

> learning Boolean functions that depend on only $\log n$ variables, even under the uniform distribution,

is regarded as a challenging open problem [Blu94, BCJ93]. Given that some very good heuristics exist for learning decision trees, it could be interesting to investigate modifying those to a practical algorithm that would apply in the more general situation of active classifiers.

On the other hand, the news is not all bad here. As suggested above, the number of attributes requested by the learner is responsible (at least in the algorithmic approach we suggested) for the time complexity of the algorithm. The sample complexity is determined by the number of starting points the blocking allows and, as we have shown, scales well in our case. Therefore, the difficulty here concerns *computational* complexity, and neither sample complexity nor the nonexistence of a good small classifier.

Proposition 9 *It is possible to learn $\mathcal{A}^{\log n}$ in the sense of Definition 3, still using only polynomially many instances and producing a polynomial-size table-lookup classifier, except that the learner may not run in polynomial time.*

Proof: As noted in Corollary 8, the sample size remains poly-sized even if we need to estimate all $O(3^n)$ possible $P(\,\varphi(x^*) = T \,|\, x^*\,)$ and $P(\,x^*_{i\mapsto 0} \,|\, x^*\,)$ values. To see that the output is small, note that the resulting active classifier will correspond to a *binary* decision tree of depth $\ln n$, and hence of size $O(2^{\ln n}) = O(n)$. (It is binary as each internal node asks for the value of a binary attribute.) ∎

[7] Just assume uniform cost for each query, and a very large penalty for incorrect responses.

That is, the learner uses a reasonable number of instances and (eventually!) outputs a small (hence efficient) active classifier. This can be useful; if the performance phase is much longer than the training phase, it may well be worth spending whatever time is necessary to find a good classifier; see Section 6.

5.2 Restricted Distributions and Underlying Concepts

So far we have discussed active learning in a very general setting, without any restriction on the underlying distribution or the underlying concept. Here, we consider whether the probability distribution and the concept class can have a significant effect on learnability, as it does in standard passive PAC-learning. We do this in the context of the class of "product distributions", in which each attribute value is chosen independently. The uniform distribution is a further restriction of this class.

The following discussion shows that sometimes the underlying concept class could be significant to learning.

Theorem 10 *The class of conjunctions can be PACT-learned under the product distribution, under any blocking model, and with any cost structure) using a greedy strategy. (Note that here we allow arbitrary number of queries; $\mathcal{A}^{(n)}$.)*

The greedy algorithm L^G for learning conjunctions is shown Figure 3. This classifier always asks first for the attribute that promises the highest immediate information gain about the classification, balanced by cost; then recurs. The appendix provides the complete proof that the L^G algorithm can PACT-learn conjunctions.

Although this is a simple observation, we note that the algorithm does not restrict the number of attributes requested by the classifier and thus shows that the earlier negative result depends crucially on having "difficult" distributions or concepts. To further understand this, note that even under product distributions, greedy active classification is not guaranteed to work in general, beyond the class of conjunctions. As a simple counterexample, consider the function $(x_1 \oplus x_2) \wedge y_1 \wedge y_2 \wedge \cdots \wedge y_n$. It is clear that the dependencies introduced by the $\oplus$ prevent the greedy heuristics from learning an active classifier. However, variants of the greedy strategy might be very useful heuristics and this, too, is worth further investigation.

5.3 Bounded *Expected* Number of Queries $\mathcal{A}^{\approx k}$

So far, we have considered the situation where there is a hard upper bound on the number of queries that can be used to classify each individual instance. Imagine, instead, that we had a hard upper bound on the *average* number of queries, for classifying a large number of instances. For example, imagine we need to classify 100,000 text documents within a total of 100,000 seconds. While the "classify each document within 1 second" requirement is guaranteed to be sufficient here, the weaker requirement, that our classifier takes *on average* 1 second per document, will probably work as well. That is, we restrict our learner to only consider classifiers in the following class of active classifiers.

Definition 11 *$\mathcal{A}^{\approx k}$ is the class of active classifiers whose members $\alpha \in \mathcal{A}^{\approx k}$ each ask, on average, at most k questions.*

Algorithm $L^G(\ \epsilon\ \in \Re^+,\ \delta\ \in (0,1)\ ):\ \alpha^G\ \in \mathcal{A}$

% Returns an active classifier α from $\mathcal{A}$ whose expected cost is, with probability at least $1-\delta$,
% within ϵ of optimal
% Uses oracle for drawing complete labeled instances, and knows cost model, $err(\cdot,\cdot),\ c(\cdot)$

Draw $M_{LG} = 2\left(\frac{\mathtt{err}(F,T)}{\epsilon}\right)^2\left(n\ln(3n) + \frac{2}{\delta}\right)$ completely-specified, labeled instances S
each $x \in S$ represented as SET *% $x = [x_1\ \neg x_2\ x_3] \approx \{1, \overline{2}, 3\}$*
Let S^+ be positive examples in S
Let $\varphi = \bigcap_{x \in S^+} x$ *[R1]*
% Re-number, flip-parity s.t. $\varphi = x_1 \wedge x_2 \wedge \cdots \wedge x_k$
For $i = 1..k$
Let $\hat{P}(x_i = 0) = \frac{1}{|S|}|\{x \in S | x_i = 0\}|$ *% ...= estimate of failure probability $P(x_i = 0)$*
Re-number s.t. $\frac{c_1}{q_1} \leq \frac{c_2}{q_2} \leq \ldots \leq \frac{c_k}{q_k}$
Let $\ell = arg\max\{\frac{c_{\ell'}}{q_{\ell'}} \leq err_M \,|\, \ell' = 1..\ell\}$ *% ...= largest index for which $\frac{c_{\ell'}}{q_{\ell'}}$ is under err_M*
% Build ℓ-node "linear" decision tree; see Figure 4:
For $i = 1..\ell$:
Label node n_i with "Perform x_i"
Connect "0"-labeled arc from n_i to: "Return F"
If $i < \ell$ THEN Connect "1"-labeled arc from n_i to: n_{i+1}
ELSE Connect "1"-labeled arc from n_ℓ to: "Return T"
Return an active classifier α^G based on this decision tree
End Algorithm L^G

Figure 3: L^G Algorithm for PACT-learning Conjunctions

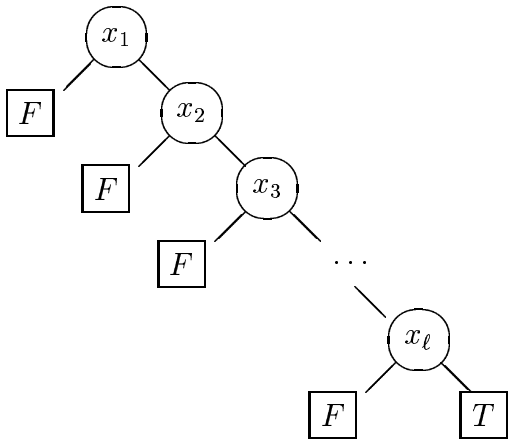

Figure 4: "Linear" Decision Tree

Note that the class $\mathcal{A}^{\approx k}$ depends on the instance distribution. As a further reason to consider this class, notice the optimal active classifier, from all of $\mathcal{A}^{all}$, will be in $\mathcal{A}^{\approx k}$, for some k. This is most relevant when the costs are "high" relative to the potential penalties.

Definition 12 *Without loss of generality, assume $c_1 \leq \ldots \leq c_n$; i.e., attributes are sorted by increasing cost. We then define:*

$$k(err, c) \quad = \quad \textit{largest } k' \textit{ such that } \left\{ \sum_{i=1}^{k'} c_i \ \leq \ err_M \right\}$$

using the err_M from Equation 2.

It is easy to see that:

Proposition 13 *The optimal active classifier from $\mathcal{A}^{all}$ should not ask* on average *more than $k(err, c)$ questions — i.e., it is in $\mathcal{A}^{\approx k(err,c)}$.*

Proof: Whenever the classifier asks more than $k(\text{err}, c)$ questions, its cost exceeds err_M; hence if it averages more than $k(\text{err}, c)$ questions, its average cost must exceed err_M. This cannot be optimal, as it is inferior to the trivial "just say No" classifier, whose average cost is at most err_M. ∎

Unfortunately, this does not mean that we can *bound* the number of questions by $k(\text{err}, c)$ — *i.e.*, we cannot restrict ourselves to $\mathcal{A}^{k(\text{err},c)}$.[8] The problem is that a classifier may reach a point where it should ask yet more questions, even after it has spent more than any possible payoff. This is because earlier costs are *sunk costs* and even if, in retrospect, they turn out to be useless, they must still be paid for. However the optimal classifier should not expect *a priori* to get into this situation very often, as a classifier that often throws good money after bad cannot be optimal.

While it may seem plausible that a variant of the dynamic programming can still PACT-learn active classifiers under the assumption of constant $k(\text{err}, c)$, this is wrong:

Theorem 14 *Unless $P = NP$, for any $\gamma > 0$, no efficient algorithm can PACT-learn $\mathcal{A}^{\approx 1+\gamma}$.*

(Note that $\mathcal{A}^{\approx 1}$ is the same as $\mathcal{A}^1$ and so it is learnable.) This hardness result only deals with computational complexity. It is open as to whether this is the only difficulty — *i.e.*, is there a result analogous to Proposition 9?

5.4 Summary

We have shown that

- one *can* learn active classifiers in $\mathcal{A}^k$ in this general setting,
- learning classifiers in $\mathcal{A}^{\log n}$ subsumes a hard computational problem, (but neither sample complexity nor representation size are problematic)

[8] On the other hand, it *is* easy to see that it is sufficient to consider classifiers in $\mathcal{A}^{k(\text{err},c)/\epsilon}$. But then our dynamic-programming algorithm will be exponential in $1/\epsilon$.

Class	sample size	computation	rep'n size
$\mathcal{A}^k$	poly	poly	poly
$\mathcal{A}^{(\ln n)}$	poly	$\equiv$ PAC-learn $\log n$-depth DTs	poly
$\mathcal{A}^{\approx k}$	poly	NP-hard	poly

Table 1: Complexity of PACT-learning various classes of Active Classifiers

- for certain classes of classifiers and distributions, learning classifiers in $\mathcal{A}^n$ can be tractable
- learning classifiers in $\mathcal{A}^{\approx k}$ is NP-hard for $k > 1$.

See Table 1.

6 On-Line Learning

As noted earlier, we allow the learner to see *complete*, unblocked instances, which the eventual classifier must pay to see. An arguably more natural model would charge the *learner* for each attribute it views, just as it charges the classifier. One natural framework, here, is "on line" learning, where a learn+active-classify (LAC) system would pay for each attribute it sees, from the very beginning. To evaluate a LAC system, we would compute a "loss function", which compares our learning $\widehat{LAC}$ with the perfect such system α^{opt}, which uses the optimal active classification process from the beginning.[9] The loss function measures the average difference between our learned $\widehat{LAC}$ with this α^{opt} (averaged over the number of instances seen); our goal here is to find a learn+classify system whose average difference goes to 0 as the number of instances increases.

To state this more precisely, we must first present our protocol: Complete instances $x^{(i)}$ are drawn sequentially, and completely blocked to produce $x^{(i)*}$. These unlabeled, blocked instances $x^{(i)*}$ are then presented to the LAC system, one by one. For each $x^{(i)*}$, LAC attempts to determine the class, asking questions as appropriate; we let $tc(LAC, x^{(i)*})$ be the penalty/cost that LAC must pay. Given a sequence of m such instances $X_m = \langle x^{(i)*} \rangle_{i=1}^m$, we define $tc(LAC, X_m) = \frac{1}{m} \sum_{x^* \in X_m} tc(LAC, x^{(i)*})$ as the average cost. For a given (stationary) distribution over instances $P(\cdot)$, target concept φ and blocker β, we can compute the expected value of $E[\, tc(LAC, X_m)\,]$ of the learn+classify system LAC, where the expectation is averaged over all sequences of m blocked instances. Our goal is to minimize this $E[\, tc(LAC, X_m)\,]$, as $m \rightsquigarrow \infty$.

Of course, for this $P(\cdot)$, φ and β, there is a best possible active classifier, which has the minimum cost $\alpha^{opt} = \alpha_{\varphi,\mathcal{A},P}$ as defined in Equation 3. (Note that this can be relative to a subclass of active classifiers, $\mathcal{A} \subset \mathcal{A}^{all}$.) As this is clearly the best that any learn+classify systems can achieve, we will consider the difference

$$\mathit{diff}(LAC, m) \;=\; \mathit{diff}_{\varphi,\mathcal{A},P}(LAC, m) \quad \triangleq \quad E[\, tc(LAC, X_m)\,] - EC_P(\alpha_{\varphi,\mathcal{A},P})$$

[9] Note that the optimal "learn+active-classify system" would simply be an active-classification system, as there is no need for the optimal system to learn.

Algorithm $LAC^*()$

% Continuously draws and processes instances

% Uses oracle for drawing complete labeled instances, and knows cost model, $err(\cdot,\cdot)$, $c(\cdot)$

For $k = 1, 2, \ldots$ do

% k^{th} exploration phase

Let $S_k = \{\}$ *% To hold set of examples*

Let $h_k = h(n, \frac{1}{2^k}, \frac{1}{2^k})$

For $i = 1..h_k$ do

"Draw" a completely blocked instance x^*

Pay $\sum_j c_j$ to see values of all attributes x

Return least-risk guess: $\begin{cases} T & \text{if } \mathtt{err}(T,F) < \mathtt{err}(T,F) \\ F & \text{otherwise} \end{cases}$

Get label $\varphi(x)$ (pay err_{min} if required)

$S_k = S_k \cup \langle x, \varphi(x)\rangle$

Use $L^{(n)}$ (with completely-specified, labeled S_k) to produce α_k

% k^{th} exploitation phase

Let $Y_k = \left[2^k W - 1\right] \times \left[\sum_{i=1}^{k} h_i + \sum_{i=1}^{k-1} Y_i\right]$

Use α_k to process next Y_k instances

End For

End Algorithm LAC^*

Figure 5: LAC^* Learn+Classify Algorithm

We would like an on-line learn+classify system that can do essentially as well as this optimal classifier, in that *diff*(LAC^*, m) goes to 0 as m increases. One standard way to do this is by a series of "explore then exploit" stages. That is, the algorithm first gathers information ("explore"), paying whatever it costs; it then uses this information to build a reasonable classifier, α_1 The algorithm will next exploit this α_1 classifier, using it to actively classify for a number of instances, with the hope that α_1 will do well enough to compensate for the cost required to learn it. This constitutes one "explore+exploit" stage. The algorithm performs a series of these stages — each time spending a bit longer in the information-gathering phase, to help produce increasingly better classifiers $\langle \alpha_1, \alpha_2, \ldots \rangle$; after learning each, it spends yet longer in the "exploit" phase, to recover the cost.

This is the basis for the LAC^* system, shown in Figure 5. In the appendix we prove that it works effectively:

Theorem 15 *With probability* 1, diff(LAC^*, m) *goes to* 0 *as* m *increases.*

While this algorithm deals with complete blocking, it is straightforward to see how to extend this result to arbitrary blocking models.

7 Related Work

Our framework is based on the "standard" learning model [BMSJ78], in which a learner receives a set of labeled (*i.e.*, correctly classified) training examples as input, and must output a good classifier. Furthermore, the notion of "good" we use is a derivative of the popular probably-approximately-correct (PAC) model [Val84]. However, we differ from the usual model in (at least) the following respects. First, our classifier (and in Section 6, our learner) receives only *partially specified* instances, which can omit the values of some or all attributes. Second, our classifier is able to *actively* request attribute values. Third, the quality of such a classifier depends on its *expected cost of obtaining attributes*, as well as its classification accuracy.

In the following we refer to some of the new aspects of our framework and relate it to previous work.

Missing attribute values: Several other learning algorithms produce classifiers that can deal with partially specified instances; *cf.*, [DLR77, LR87, Qui89, SG94]. However, these classifiers are not able to actively obtain missing information. Other research [BDD93, KR99, GKS97, KR95] considers the problem of learning from partially specified instances, but with the goal of (resp.) later classifying *complete* instances, or later reasoning with respect to the learned concept; *n.b.*, these other systems do not consider ways for the classifier to gather more information.

Littlestone [Lit88], John *et al.* [JKP94], and others consider the situation where only a subset of the variables are needed to perform the classification (for each instance); those systems, however, assume that the values of all variables, both relevant and irrelevant, are given. (The Greiner *et al.* [GGK97] model is similar, but here the learner sees only the values of the relevant variables.) To connect this to our model, note that an active classifier would never request the value of any irrelevant variable and would, moreover, seek the "minimal cost" set of relevant attributes.

Active-ness: Of course, "active" classification is not a novel concept; many diverse areas use related ideas (including planning, diagnosis, decision theory, and so on). As just one illustrative example, Heckerman *et al.* [HBR94] describe how to translate any given decision net (which satisfies certain properties) into an effective "active classifier" — one that both isolates and repairs the fault, taking account of costs and the probability of various diagnoses being correct. Other examples of studies of active classification exist in the vision community [SUB96, CAL93, BJ98]. However those frameworks do not address the challenge of *learning* such classifiers. One possible reason is that the tasks of learning and classifying can often be decoupled. For instance, [HBR94] could appeal to standard Bayesian-network learning techniques to learn the necessary distributions. While conceding that such a decoupling is possible in many cases, the basic question examined in this paper is whether there can be any advantage in studying *learning and active classification together*; see Section 3.

Our task, of learning active classifiers, is also distinct from the task of *actively learning (passive) classifiers*. For example, [Ang87, Ang88], [KMT93], [FSST97], consider the "learning with membership queries" model, in which the *learner* can request *labels* of examples as it is learning. Recall however that our *learner* is seeking optimally inexpensive active classifiers, rather than optimally accurate passive ones; moreover, we focus primarily on a

passive learner (until Section 6).

Utility: There are several learning projects that attempt to learn classifiers that are sensitive to test costs. For example, Turney [Tur95] (and others; see references therein) uses heuristic methods to build decision trees that minimize classification and test costs; by contrast, we are seeking provably optimal active classifiers, of any representation. Haussler [Hau92] studies a decision-theoretic generalization of the PAC model, in which the learner may output a classification or a decision rule with the goal of minimizing a given loss function. However, his classifier always receives complete instances, and so is not active in our sense.

Finally, our framework shares much in common with reinforcement learning (RL) [SB98], In both frameworks, the performance system (in our case, the active classifier) is expected to act in a way that maximizes its reward, which often involves acquiring new information before making an important decision (for us, "labeling the instance"). Our basic model, however, involves separate phases: first learning, then classifying. (Note, however, that our learner does exploit information about the classification environment; see Section 3.) The learn+classify model we present in Section 6 is more like RL, as the learner is combined with the classifier, and so must deal with partial information at each point. Moreover, our basic algorithm can also be viewed as trading-off exploring versus exploiting. Our model differs, however, by providing fairly immediately feedback, after each instance.

8 Conclusions and Future Work

In this paper, we have proposed a framework for studying *learning* and *active classification* together. It seems plausible that this might entail some "Learning-to-Reason"-style advantages [KR97] in that learning a particular classification strategy (with respect to a particular cost structure and blocker) might be easier than learning the full concept (and perhaps distribution). We have presented results which support this thesis, under assumption of the amount of activity the learner is allowed to undertake. Three research directions that offer hope for further positive results are (1) other restrictions on the type of active classifiers allowed; (2) approximation techniques; and (3) the combination of stronger restrictions on both the concept class and the distribution.

We have also explored the obvious "on-line" version of this framework, where the *learner* incurs costs while it was learning, as it must pay for any attribute it sees, and has to predict each instance's classification, risking penalty. Here, our goal is to minimize total cost over the learner's lifetime. We show the (unsurprising) result that a learner can converge to the optimal classifier, by employing a sequence of exploration steps, to produce successively better classifiers, each followed by an exploitation phase, to recoup the cost of producing that classifier.

We close by noting an interesting contrast between our results and standard PAC concept learning: *Few of our results depend, in any critical way, on the identity of concept class.* Theorem 7 and its corollary, while rather restrictive in many other respects, works for every possible class of concepts. To explain this difference, first note that when one does not see all the attributes then the induced probabilistic concept [KS90] over the visible attributes can, in general, be quite complex, even if the real concept is a simple one. A second issue

is that the distribution appears to matter more. For example, attributes which are logically irrelevant to the concept might nevertheless be very important to an active classifier, if their values are somehow correlated with the correct label. Both of these reasons suggest that, to whatever extent that active classifiers can be learned at all, we might expect to find results that do not distinguish between concept classes to the extent that ordinary passive classifier learning theory does.

Acknowledgments

We gratefully acknowledge receiving helpful comments from Mukesh Dalal, Sanjeev Kulkarni, Nick Littlestone, and Dale Schuurmans. Some of this work was performed while the first author (RG) worked at Siemens Corporate Research, NJ; the second author (AG) worked at NEC Research Institute, NJ, and the third author (DR) worked at the Weizmann Institute of Science. Russ Greiner is supported by an NSERC operating grant, and Dan Roth is supported by NSF grants IIS-9801638 and IIS-9984168.

A Proofs

Proof of Theorem 5: We reduce our problem to the NP-complete problem

> **Definition:** EXACT COVER BY 3-SETS (X3C) **Decision Problem** [GJ79, p221]**:** Given set of elements $X = \{x_1, \ldots, x_{3r}\}$ and a collection $S = \{s_1, \ldots, s_m\}$ of 3-element subsets of X, does S contain an exact cover of X; *i.e.*, is there a subcollection $S' \subset S$ such that each $x \in X$ is in exactly one element $s_x \in S'$?

Now given any instance $\langle S, X \rangle$, form distribution over binary variables $\{s_1, ..., s_m, x_1, ...x_{3r}\}$ where the x_i's are independent of each other, and are true with probability $P(\, x_i = 1\,) = p = 1/2$ Also each $s_j \equiv x_{j1} \& x_{j2} \& x_{j3}$ — *i.e.*, $P(\, s_j \,|\, x_{j1}, x_{j2}, x_{j3}\,) = 1$ and $P(\, s_j \,|\, \neg x_{jk}\,) = 0$ for each $k = 1, 2, 3$.

Now let $\varphi \equiv x_1 \& \ldots \& x_{3r}$ and the cost of each s_j be $c(s_j) = 1$ and of each x_i be $c(x_i) = r$. Finally, set the penalty for being wrong $\mathrm{err}(T, F) = \mathrm{err}(F, T) = 2\frac{1-p^{3r}}{(1-p^3)\times p^{3r}}$.

We now show that,

> There is an exact cover iff there is an active classifier, in this situation, whose expected cost is at most $\alpha = (1 - p^{3r})/(1 - p^3) = 8 \times (1 - 1/8^r)/7$.

$\Longrightarrow$**:** Assume there is exact cover — w.l.o.g. call it $\{s_1, \ldots, s_r\}$. Now consider the active classifier α^* that simply ask these queries in order $\langle s_1, ..., s_r \rangle$, until one fails (in which case, return "No") or if all pass (here say "Yes") — *i.e.*, (See the decision tree in Figure 4).

To compute expected cost: Note

$$\begin{aligned} P(\, s_j = T \,|\, s_1 = T,\ s_2 = T,\ \ldots,\ s_{j-1} = T\,) &= P(\, s_j = T\,) && (6)\\ &= P(\, x_{j1},\ x_{j2},\ x_{j3}\,) \\ &= P(\, x_{j1}\,)\, P(\, x_{j2} \,|\, x_{j1}\,)\, P(\, x_{j3} \,|\, x_{j2}, x_{j1}\,) \\ &= P(\, x_{j1}\,)\, P(\, x_{j2}\,)\, P(\, x_{j3}\,) \quad = \quad p^3 && (7) \end{aligned}$$

where Equation 6 uses that fact that, as this is an exact cover, s_j involves different variables than $s_1, \ldots, s_{j-1}$; and Equation 7 uses that fact that the x_i's are independent.

Notice this classifier never returns the wrong prediction, hence its expected cost is simply the expected number of evaluations, which is

$$EC_P(\alpha^*) \quad = \quad (r \times w^r) + \sum_{i=1..r} i \times (1-w) \times w^{i-1} \quad = \quad (1-w^r)/(1-w)$$

as claimed.

$\Longleftarrow$: Observe

- there is an optimal active classifier that uses only s_j's rather than x_i's.

  (Given any purportedly optimal classifier α that uses a x_{jk}, form a new classifier α' that differs from α only by replacing that x_{jk} with s_j. Observe that α' will be as correct as α: Suppose α reaches this x_{jk}-labeled node. If $x_{jk} = 0$, then the value of $s_j = 0$, and the correct answer is "False". On αs "$x_{jk} = 1$" branch: here the $s_j = 1$ test will perform even more appropriate tests. Moreover, s_j costs less — $c(s_j) < c(x_{jk})$.

- we need only consider linear structure of s_j's, as finding any $s_j = F$ immediately tells us the answer is "F").

This means the optimal active classifier α can be viewed as $\alpha \equiv \langle s_1, ..., s_m \rangle$.

Moreover, we may assume that $m > r$: As here there is no exact cover, we know that a an *always correct* classifier will have "length" $> r$. We first show that any such "always correct" active classifier will have cost strictly greater than α.

We can assume, wlog, that each of the s_i's on the path $\alpha \equiv \langle s_1, ..., s_m \rangle$ will include at least one x_{ik} that did not appear on any of $\langle s_1, \ldots, s_{i-1} \rangle$. (Otherwise, can get a less expensive and equally correct classifier by deleting that useless s_i.) This means the cost of dealing with the first r s_i's is at least $\sum_{i=1}^{r} i \times P($ "reaching the F under this node" $) = \sum_{i=1}^{r} i \times (1-p) \times p^{i-1} = (1 + rp^{r+1} - (r+1)p^r)/(1-p)$. In addition, we know that at least one x_i was not covered by any of $\{s_1, \ldots, s_r\}$. This means we will follow the 1-branch from all r of these s_i's with probably at least p^{3r-1}, which means the probability of performing $r+1$ tests is p^{3r-1}; this adds on an additional cost of $(r+1)p^{3r-1}$. Hence, α's total cost is at least $(1 + rp^{r+1} - (r+1)p^r)/(1-p) + (r+1)p^{3r-1} = 2 + r/2^r - 2(r+1)/2^r + 2(r+1)/8^r$; for $r > 3$, this is strictly larger than α.

Of course, we might also consider classifiers that were not completely correct, but instead were "truncated", at say depth q and simply announced a class (either "T" or "F"), However, stopping at depth $q \leq r$ will again cause the active classifier to cost more than α. As there is a no exact cover, stopping before $r+1$ means at least one x_i will NOT be included in the tests. It will reach this final node with probability at least p^{3r-1}. Now suppose the α returns T. Then with probability at least $p^{3r-1} \times (1-p)$, the value of this untested x_i was "false", which means this is the wrong answer. The cost of this mistake, therefore, is at least $p^{3r-1} \times (1-p) \times \text{err}(F, T) = \frac{2}{8^r} \frac{1}{2} \times 2 \frac{1-1/8^r}{(7/8) \times 1/8^r} = 2\alpha > \alpha$. (Similarly for the case where α returns F here.)

To complete the proof, we need to observe that the classifier is small (just linear in the size of the concept $\varphi \equiv x_1 \& \ldots x_{3r}$), that the class of classifiers $\mathcal{C}$ contains just one classifier

(corresponding to $\bigwedge_i c_i \wedge \bigwedge_j x_j$) and hence is clearly PAC-learnable, and that the distribution use is just a product distribution, and hence is of linear size. ∎

Theorem 7 uses the following lemma:

Lemma 16 *An agent must take one of r possible actions $\{a_1, \ldots, a_r\}$, whose true costs are $c(a_i) \in \Re$. However, due to sampling error, the agent perceives these costs as $\hat{c}(a_i)$, where $|\hat{c}(a_i) - c(a_i)| \leq \beta_i$. Then the difference in cost between the optimal agent, who takes action $a^{opt} = \arg\min\{ c(a_i) \}$, and the "estimation-based" agent, who takes action $\hat{a} = \arg\min\{ \hat{c}(a_i) \}$, is bounded by $2\beta_{max}$, where $\beta_{max} = \max\{\beta_i\}$.*

Proof:

$$\begin{array}{rcccccccc} c(\hat{a}) - c(a^{opt}) & \leq & c(\hat{a}) - \hat{c}(\hat{a}) & + & \hat{c}(\hat{a}) - \hat{c}(a^{opt}) & + & \hat{c}(a^{opt}) - c(a^{opt}) \\ & \leq & \beta_{\hat{a}} & + & 0 & + & \beta_{a^{opt}} \\ & \leq & \beta_{max} + \beta_{max} & = & 2\,\beta_{max} & & \end{array}$$

∎

Proof of Theorem 7: We first need some definitions. Given any $x^* \in \{0, 1, *\}^n$, let $Ext(x^*) \subset \{0,1\}^n$ be the set of complete tuples that extend x^* — *i.e.*, each $x \in Ext(x^*)$ is a complete tuple that agrees with each attribute value specified in x^*.

Define $P^{i \mapsto 0}_{x^*} = P(\, Ext(x^*_{i \mapsto 0}) \,|\, Ext(x^*)\,)$ to be the probability that attribute x_i has the value 0, given the partial instance x^* — *e.g.*, $P^{3 \mapsto 0}_{\langle 1,*,*\rangle} = P(\, \langle 1,*,0\rangle \,|\, \langle 1,*,*\rangle\,)$ is the probability that attribute x_3 will have value 1, given that we know attribute x_1 had value 1 — and $P^{\varphi}_{x^*} = P(\, \varphi(x) = T \,|\, x \text{ extends } x^*\,) = \sum_{x \in Ext(x^*)} P(\, \varphi(x) = T \,|\, x^*\,)$. Notice both $P^{i \mapsto 1}_{x^*}$ and $P^{\varphi}_{x^*}$ are *conditional probabilities* — each relative to the conditioning event, $P(\, x^*\,) = \sum_{x \in Ext(x^*)} P(\, x\,)$.[10] The $\widehat{P^{\varphi}_{x^*}}$ and $\widehat{P^{i \mapsto 1}_{x^*}}$ quantities shown in Figure 2 are empirical estimates of these quantities. There is one such number for each of the $g(k) = \sum_{i=0}^{k} \binom{n}{i} 2^i \leq k(2n)^k$ partial instances (in $X^*_{0..k}$) that can occur.

As shown in Figure 2, the $L^{(k)}$ learner first draws

$$M \quad = \quad M_k(\epsilon, \delta) \quad = \quad \frac{err_M^2 2^{4k+5}}{9\epsilon^2} \ln \frac{4\, g(k)}{\delta} \tag{8}$$

instances. (Recall from Equation 2 that $err_M = \max\{\text{err}(T,F), \text{err}(F,T)\}$ is the largest error for giving the wrong response.)

We will use these instances to produce estimates $\hat{P}(\, x^* \text{ occurs}\,)$ of $P(\, x^* \text{ occurs}\,) = P(\, x^*\,)$ and $\hat{P}(\, \varphi(x) = T,\ x \text{ extends } x^*\,)$ of $P(\, \varphi(x) = T,\ x \text{ extends } x^*\,)$, for each $x^* \in X^*_{0..k}$. Notice these are each *unconditional* probabilities.

We first bound the probability that *any* of these $2\, g(k)$ estimates is more than $\frac{3\epsilon}{8\, err_M\, 4^k}$ from the correct value. Here, we use Hoeffding's Inequality [Hoe63, Che52], which bounds the probability that the empirical average of m iid (independent and identically distributed) instances $X_i \in [0, r]$ with common mean μ, will be far from μ:

$$P\left(\left| \frac{1}{m} \sum_{i=1..m} X_i - \mu \right| \ > \ \lambda \right) \quad \leq \quad 2e^{-2m\left(\frac{\lambda}{r}\right)^2} \tag{9}$$

[10] Note this is *not* measuring the probability that the classifier will see such an instance — as that quantity is 0 for many partial instances that DO occur.

Now consider a fixed $x^* \in X^*_{0..k}$, and let the $X_i^{x^*}$ variable be 1 if a randomly drawn sample will extend x^*, and 0 otherwise. (Hence, $r = 1$.) After M instances, the chance that the empirical estimate $\hat{P}(\, x^* \text{ occurs}\,) = \frac{1}{M} \sum_{i=1..M} X_i^{x^*}$ will be more than $\lambda = \frac{3\epsilon}{8\, err_M\, 4^k}$ away from $\mu = P(\, x^* \text{ occurs}\,)$ will be under $2\exp(-2M \left(\frac{3\epsilon}{8\, err_M\, 4^k}\right)^2) \leq \frac{\delta}{2\, g(k)}$. Hence, the probability that *any* of the $g(k)$ possible partial instances will be $\frac{3\epsilon}{8\, err_M\, 4^k}$ off is

$$\begin{array}{rcl} & & P(\, \exists x^* \in X^*_{0..k}\, |\hat{P}(\, x^* \text{ occurs}\,) - P(\, x^* \text{ occurs}\,)| > \lambda\,) \\ & \leq & \sum_{x^* \in X^*_{0..k}} P(\, |\hat{P}(\, x^* \text{ occurs}\,) - P(\, x^* \text{ occurs}\,)| > \lambda\,) \\ & \leq & g(k) \times \frac{\delta}{2\, g(k)} \quad = \quad \frac{\delta}{2} \end{array}$$

Similarly,

$$\begin{array}{l} P(\, \exists x^* \in X^*_{0..k}\, |\, \hat{P}(\, \varphi(x) = T,\ x \text{ extends } x^*\,) \ -\ P(\, \varphi(x) = T,\ x \text{ extends } x^*\,) \,|\ \geq\ \frac{3\epsilon}{8\, err_M\, 4^k}\,) \\ \qquad \leq \quad \frac{\delta}{2} \end{array}$$

Therefore, with probability at least $1 - \delta$, we can assume that all of the estimates are within $\lambda = \frac{3\epsilon}{8\, err_M\, 4^k}$ of correct — *i.e.*,

$$\begin{array}{rcl} |\, \hat{P}(\, x^* \text{ occurs}\,) \ -\ P(\, x^* \text{ occurs}\,)\,| & \leq & \lambda \\ |\, \hat{P}(\, \varphi(x) = T,\, x \text{ extends } x^*\,) \ -\ P(\, \varphi(x) = T,\, x \text{ extends } x^*\,)\,| & \leq & \lambda \end{array} \qquad (10)$$

Now let α^{opt} be optimal active classifier, $\widehat{\alpha}$ be the classifier returned by our $L^{(k)}$ learner, which uses the estimates shown above, and $\textsc{After}(\, \alpha,\, x_0^*\,)$ be total expected cost of using the active classifier α. As $P(\, x_0^*\,) = \hat{P}(\, x_0^*\,) = 1$ for $x_0^* = \langle *, \ldots, * \rangle \in X_0^*$, we need only show that

$$\textsc{After}(\, \widehat{\alpha},\, x_0^*\,) \ -\ \textsc{After}(\, \alpha^{opt},\, x_0^*\,) \quad \leq \quad \epsilon$$

when our estimates satisfy Equation 10. Given that $P(\, x_0^*\,) = \hat{P}(\, x_0^*\,) = 1$, it suffices to prove that

$$|\, \hat{P}(\, x_\ell^*\,) \times \textsc{After}(\, \widehat{\alpha},\, x_\ell^*\,) \ -\ P(\, x_\ell^*\,) \times \textsc{After}(\, \alpha^{opt},\, x_\ell^*\,)\,| \quad \leq \quad \frac{\epsilon\ (4^{k+1-\ell} - 1)}{4^{k+1}} \qquad (11)$$

holds for all $x_\ell^* \in X_\ell^*$ (*i.e.*, for all partial instances that include exactly ℓ specified values). This we prove by induction.

We deal first with the base $\ell = k$ case. We will use $C(\, x_\ell^*,\ \chi\,)$ to refer to the cost of the action $\chi \in \{T, F, 1, \ldots, n\}$, given the partial instance x_ℓ^* ; and $\widehat{C}(\, x_\ell^*,\ \chi\,)$ to refer to our estimate of this cost. By Lemma 1 (shown above), we need only bound the difference between $P(\, x_k^*\,) \times C(\, x_k^*,\ \chi\,)$ and $\hat{P}(\, x_k^*\,) \times \widehat{C}(\, x_k^*,\ \chi\,)$, for the two options — $\chi = T$ and $\chi = F$ — as $|\hat{P}(\, x_k^*\,) \times \textsc{After}(\, \alpha,\, x_k^*\,) - P(\, x_k^*\,) \times \textsc{After}(\, \alpha^{opt},\, x_k^*\,)|$ is at most twice the larger of these differences.

Now observe that

$$\begin{array}{rcl} P(\, x_k^*\,) \times C(\, x_k^*,\ F\,) & = & P(\, x_k^*\,) \times P^{\varphi}_{x^*} \times \text{err}(F, T) \\ & = & P(\, x \text{ extends } x_k^*\,) \times P(\, \varphi(x) = T \,|\, x \text{ extends } x_k^*\,) \times \text{err}(F, T) \\ & = & P(\, \varphi(x) = T,\ x \text{ extends } x_k^*\,) \times \text{err}(F, T) \end{array}$$

and similarly

$$\hat{P}(\, x^*_k \,) \;\times\; \widehat{C}(\, x^*_k,\; F \,) \;\;=\;\; \hat{P}(\, \varphi(x) = T,\; x \text{ extends } x^*_k \,) \;\times\; \mathrm{err}(F,T)$$

Hence the difference between the true and estimated values

$$\begin{array}{ll} & |P(\, x^*_k \,) \times C(\, x^*_k,\; F \,) \;-\; \hat{P}(\, x^*_k \,) \times \widehat{C}(\, x^*_k,\; F \,)| \\ = & |P(\, \varphi(x) = T,\; x \text{ extends } x^*_k \,) \;\times\; \mathrm{err}(F,T) \;-\; \hat{P}(\, x \text{ extends } x^*_k,\; x \text{ extends } x^*_k \,) \times \mathrm{err}(F,T)| \\ = & \mathrm{err}(F,T) \times |P(\, \varphi(x) = T,\; x \text{ extends } x^*_k \,) \;-\; \hat{P}(\, \varphi(x) = T,\; x \text{ extends } x^*_k \,)| \\ \leq & \mathrm{err}(F,T) \times \lambda \;\;\leq\;\; \lambda\, err_M \end{array}$$

as $\mathrm{err}(F,T) \leq err_M$. This is also the error bound for $|P(\, x^*_k \,) \times C(\, x^*_k,\; T \,) \;-\; \hat{P}(\, x^*_k \,) \times \widehat{C}(\, x^*_k,\; T \,)|$. Hence, by Lemma 1, the difference $|\hat{P}(\, x^*_k \,) \times \textsc{After}(\, \alpha,\, x^*_k \,) - P(\, x^*_k \,) \times \textsc{After}(\, \alpha^{opt},\, x^*_k \,)|$ is at most $2\,\lambda\, err_M \;=\; 2\,\frac{3\epsilon}{8 \times 4^k} = \frac{\epsilon}{4^{k+1}}(4^1 - 1)$, as appropriate.

For the inductive step, we need to bound $\Delta_\ell = |\hat{P}(\, x^*_\ell \,) \times \textsc{After}(\, \alpha,\, x^*_\ell \,) - P(\, x^*_\ell \,) \times \textsc{After}(\, \alpha^{opt},\, x^*_\ell \,)|$, given that $\Delta_{\ell+1} = |\hat{P}(\, x^*_{\ell+1} \,) \times \textsc{After}(\, \alpha,\, x^*_{\ell+1} \,) - P(\, x^*_{\ell+1} \,) \times \textsc{After}(\, \alpha^{opt},\, x^*_{\ell+1} \,)| \leq \frac{\epsilon}{4^{k+1}}(4^{k-\ell} - 1)$ holds for all $x^*_{\ell+1} \in X^*_{\ell+1}$. Again, using Lemma 1, we need only bound the largest error of any of the $n+2$ options, at x^*_ℓ.

For $\chi \in \{T,\; F\}$, the error $|P(\, x^*_\ell \,) \times C(\, x^*_\ell,\; \chi \,) \;-\; \hat{P}(\, x^*_\ell \,) \times \widehat{C}(\, x^*_\ell,\; \chi \,)|$ remains bounded by $\lambda\, err_M$ using the same proof as for x^*_k. For the other actions $i \in \{1, \ldots, n\}$, we use Equation 4 and the fact that $\textsc{After}(\, \alpha,\, x^*_{\ell+1} \,)$ is the value of $C[x^*_{\ell+1}]$ given the probability values used ($\hat{P}(\,\cdot\,)$ for $\textsc{After}(\, \alpha,\, x^*_{\ell+1} \,)$, and $P(\,\cdot\,)$ for $\textsc{After}(\, \alpha^{opt},\, x^*_{\ell+1} \,)$), we see

$$\begin{array}{ll} & |P(\, x^*_\ell \,) \times C(\, x^*_\ell,\; i \,) \;-\; \hat{P}(\, x^*_\ell \,) \times \widehat{C}(\, x^*_\ell,\; i \,)| \\ = & |\; P(\, x^*_\ell \,) \times [c_i + \\ & \quad P(\, x_i = 1 \,|\, x \text{ extends } x^* \,)\; \textsc{After}(\, \alpha^{opt},\, x^*_{i \mapsto 1} \,) \;+\; P(\, x_i = 0 \,|\, x \text{ extends } x^* \,)\; \textsc{After}(\, \alpha^{opt},\, x^*_{i \mapsto 0} \,)] \\ & - \;\; \hat{P}(\, x^*_\ell \,) \times [c_i + \\ & \quad \hat{P}(\, x_i = 1 \,|\, x \text{ extends } x^* \,)\; \widehat{\textsc{After}}[\, \alpha,\, x^*_{i \mapsto 1} \,] \;+\; \hat{P}(\, x_i = 0 \,|\, x \text{ extends } x^* \,)\; \widehat{\textsc{After}}[\, \alpha^{opt},\, x^*_{i \mapsto 0} \,]] \;| \\ \leq & c_i \;|\, P(\, x^*_\ell \,) - \hat{P}(\, x^*_\ell \,) \,| \;\;+ \\ & \quad |[P(\, x^*_{i \mapsto 1}, x^* \,)\; \textsc{After}(\, \alpha^{opt},\, x^*_{i \mapsto 1} \,) \;+\; P(\, x^*_{i \mapsto 0}, x^* \,)\; \textsc{After}(\, \alpha^{opt},\, x^*_{i \mapsto 0} \,)] \\ & \quad - [\hat{P}(\, x^*_{i \mapsto 1}, x^* \,)\; \widehat{\textsc{After}}[\, \alpha,\, x^*_{i \mapsto 1} \,] \;+\; \hat{P}(\, x^*_{i \mapsto 0}, x^* \,)\; \widehat{\textsc{After}}[\, \alpha,\, x^*_{i \mapsto 0} \,]]| \\ \leq & c_i\, \lambda \\ & \quad + \;\; |P(\, x^*_{i \mapsto 1}, x^* \,)\; \textsc{After}(\, \alpha^{opt},\, x^*_{i \mapsto 1} \,) \;-\; \hat{P}(\, x^*_{i \mapsto 1}, x^* \,)\; \widehat{\textsc{After}}[\, \alpha,\, x^*_{i \mapsto 1} \,]| \\ & \quad + \;\; |P(\, x^*_{i \mapsto 0}, x^* \,)\; \textsc{After}(\, \alpha^{opt},\, x^*_{i \mapsto 0} \,) \;-\; \hat{P}(\, x^*_{i \mapsto 0}, x^* \,)\; \widehat{\textsc{After}}[\, \alpha,\, x^*_{i \mapsto 0} \,]]| \\ \leq & err_M \lambda + \;\Delta_{\ell+1} \;+\; \Delta_{\ell+1} \\ \leq & \lambda err_M + \;2\frac{\epsilon}{4^{k+1}}(4^{k-\ell} - 1) \end{array}$$

(This uses the inductive assumption, our unproblematic assumption that $c_i \leq err_M$, and the fact that $\hat{P}(\, x^*_{i \mapsto 0}, x^* \,)$ is simply $\hat{P}(\, x^*_{i \mapsto 0} \,)$.)

Hence, from Lemma 1, we know that

$$\begin{array}{rcl} \Delta_\ell & \leq & 2\, \max\{\; \lambda err_M,\; \lambda err_M,\; \lambda err_M \;+\; 2\frac{\epsilon}{4^{k+1}}(4^{k-\ell} - 1)\} \\ & = & 2[\frac{3\epsilon}{8 \times 4^k} \;+\; 2\frac{\epsilon}{4^{k+1}}(4^{k-\ell} - 1)] \\ & = & \frac{\epsilon}{4^{k+1}}\,[3 + \;4 \times 4^{k-\ell} \;- 4] \;\;=\;\; \frac{\epsilon}{4^{k+1}}\,(4^{k+1-\ell} - 1) \end{array}$$

as desired. ∎

Proof of Theorem 10: As shown in Figure 3, the L^G algorithm first collects a set of instances, then produces an active classifier α^G using these instances. We need only prove (1) if this sample is truly representative (*i.e.*, if $\hat{P}(\, x_i = 0\,) = \; P(\, x_i = 0\,)$), then the resulting α^G is optimal; and (2) the sample size is sufficient to simultaneously estimate the costs of all possible active classifiers (for conjunctions) to within $\epsilon/2$, with probability at least $1 - \delta$. In particular, let α^{opt} be the optimal classifier. Given (2), we know that

$$\begin{array}{ll} & EC_P(\alpha^G) - EC_P(\alpha^{opt}) \\ = & [EC_P(\alpha^G) - \widehat{EC}_P(\alpha^{opt})] \; + \; [\widehat{EC}_P(\alpha^{opt}) - \widehat{EC}_P(\alpha^G)] \; + \; [\widehat{EC}_P(\alpha^G) - EC_P(\alpha^{opt})] \\ \leq & |EC_P(\alpha^G) - \widehat{EC}_P(\alpha^{opt})| \; + \; 0 + \; |\widehat{EC}_P(\alpha^G) - EC_P(\alpha^{opt})| \\ \leq & \epsilon/2 \; + \; 0 \; + \; \epsilon/2 \quad = \quad \epsilon \end{array}$$

To prove (2), we note that there are at most $3^n\, n!$ possible active classifiers for conjunctions. (This requires observing that we need only consider classifiers that correspond to linear decision trees (see Figure 4), and there are only $n!$ orderings of the variables, and each variable can occur either positively (as "x_i") or negatively (as "$\neg x_i$"), or be omitted.) Moreover, we need only consider classifiers whose costs range from 0 to $\mathrm{err}(F, T)$, as our space of classifiers trivially includes the degenerate classifier that simply returns F, whose error is at most $\mathrm{err}(F, T)$. Now realize that each instance in the training sample is providing an estimate of the expected cost of each classifier. We can then use Hoeffding's Inequality (Equation 9) to bound our estimates of the quality of the classifiers: After M_{L^G} instances, the probability that our empirical estimate of any classifier, based on this sample (and the induced values $\hat{P}(\, x_i = 0\,)$) will be more that $\epsilon/2$ off is under

$$2 \exp\left(-2 M_{L^G} \left(\frac{\epsilon/2}{\mathrm{err}(F, T)} \right)^2 \right) \quad \leq \quad \frac{\delta}{3^n n!}$$

Hence, the chance that our estimates of any of the $3^n n!$ classifiers will be off by more than $\epsilon/2$ is under the $3^n n! \frac{\delta}{3^n n!} = \delta$, as desired.

We therefore need only prove (1): that L^G produces the active classifier that is optimal, given the sample. It is trivial to see that the "intersection" step (line *[R1]*) PAC-learns the target concept; *cf.*, [Val84]. To show that the minimal-cost active classifier will consider these variables in order of increasing values of c_i/q_i, consider a classifier α that does not — *e.g.*, that asks for x_2 before x_1, where

$$\frac{c_1}{q_1} \quad < \quad \frac{c_2}{q_2} \tag{12}$$

(In general, assume this is the first violation.) Hence, α's cost, on reaching x_1, will be $c_2 + q_2 c_1$, where (in general) $q_i = \hat{P}(\, x_i = 0\,)$ and c_i is the cost of acquiring the value of x_i. To show that α cannot be optimal, let α' be a classifier that differs from α only by exchanging these variables, placing x_1 before x_2. Given Equation 12, α's cost $c_1 + q_1 c_2$ is under αs cost.

To show that the optimal classifier should include all-and-only the first ℓ variables, just observe that the cost of such a classify is less than a classifier that stops after $m < \ell$ variables, or one that continues for $n > \ell$ variables. ∎

Proof of Theorem 14: We again reduce our problem to the NP-complete problem EXACT COVER BY 3-SETS (X3C) (shown in proof of Theorem 5).

Given any X3C instance $\langle X, S \rangle$, produce a PACT-learn instance whose variables correspond to subsets $S = \{s_j\}_{j=1}^{m}$, and whose training instances (basically) correspond to the elements $X = \{x_i\}_{i=1}^{3r}$. We will use a total of $\frac{9\,r\,(r-1)}{2\gamma}$ training instances, formed from $3r+2$ distinct instances. Using $f = \frac{4\,\gamma}{3(r-1)}$,

- $3r$ negative instances, $\{x^{(1)}, \ldots, x^{(3r)}\}$ where each $x^{(i)} = \langle x_1^{(i)}, \ldots, x_m^{(i)} \rangle$, where $x_j^{(i)} = 0$ iff $x_i \in s_j$ (and $= 1$ otherwise). We include each 1 time (so its empirical probability is $f/(6r)$);
- one positive instance $x^{(+)} = \langle 1, 1, \ldots, 1 \rangle$ and include it $3r$ times (so its empirical probability is $f/2$);
- one negative instance $x^{(-)} = \langle 0, 0, \ldots, 0 \rangle$, and include it $3r[3(r-1) - 4\gamma]/(2\gamma)$ times (so its empirical probability is $1 - f$);

Let the cost of obtaining the value of any attribute is $c_i = 1$, and the penalty for being wrong is $R = \frac{9r(r-1)}{\gamma}$. We are also assuming complete blocking.

Claim: There is an active classifier with expected total cost $1 + \gamma$ iff there is an exact cover.

$\Leftarrow$: If there is an exact cover $C = \{s_1, ..., s_r\} \subset S$, then form an active classifier α^{EC} whose associated decision tree is "linear" (see Figure 4), with nodes labeled by the sets $s \in C$, going to "F" if $s_i = 0$, and further down if $s_i = 1$. The final internal node is s_r; its "1"-labeled arc leads to "T". The total cost of this active classifier is

$$\begin{array}{lll} (1-f) \times 1 & & \textit{;; to deal with } x^{(-)} \\ + \quad f/2 \times r & & \textit{;; to deal with } x^{(+)} \\ + \quad f \times (r+1)/4 & & \textit{;; to deal with all } 3r\ x^{(i)}\textit{'s} \\ = \quad 1 + f[3(r-1)/4] & = & 1 + \gamma \end{array}$$

(As this classifier always returns the correct answer, this is the entire cost.) The first line requires the observation that $x^{(-)}$, which occurs with probability $P(\,x^{(-)}\,) = 1 - f$, will be answered (correctly) after examining the first node at cost $c_? = 1$; and the second line requires observing that $x^{(+)}$ requires examining all r nodes in the decision tree, at cost r. The third requires noting that exactly 3 of the $x^{(i)}$s will reach the "false"-branch of each of the nodes in the tree; as the cost of reaching the j^{th} node is j, this means the total cost of handing all $3r$ of the $x^{(i)}$s is $P(\,x^{(i)}\,) \times 3 \sum_{j=1}^{r} j \;=\; \frac{f}{6r}\, 3\, r(r+1)/2$.

Notice this cost also corresponds to the expected number of tests that will be performed.

$\Rightarrow$: Observe first that the optimal tree is always a linear tree: as every instance that follows the $s_i = 0$ branch is labeled false, there is no reason to ever have more than the single node $\boxed{\text{F}}$ under any $s_i = 0$ branch.

Hence, the only issue is: how deep is the tree? Suppose a classifier's decision tree stops early, after requesting only j of the r sets. If s_j's 1-labeled arc goes to F, then it is better to

just say "F" without running any tests; here the penalty is $P(\,x^{(+)}\,)\times R \;>\; \frac{1}{2}\frac{4\gamma}{3(r-1)}\times\frac{9r(r-1)}{\gamma} \;=\; 6r > 12$.

Otherwise, if s_j's 1-labeled arc goes to T, then the total cost is at least the penalty for mislabeling at least one $x^{(i)}$, which is $f/(6r)\times R \;=\; \frac{1}{6r}\frac{4\gamma}{3(r-1)}\times\frac{9r(r-1)}{\gamma} \;=\; 2$.

Now suppose tree is too deep, including at least $r+1$ internal nodes, where the first r are NOT a cover. Then at least one $x^{(i)}$ reaches this $r+1$'s internal node, rather than stopping at (at worst) the r^{th} internal node, meaning the *additional* cost of this tree, over the minimal-possible tree, is at least the cost of this additional test: $[(r+1)-r]\times f/(6r) \;>\; 0$. (To be completely rigorous, we should note that (1) while some individual $x^{(i)}$'s might cost less, the *average* cost, over all $3r$ of the $x^{(i)}$'s, must be more than $3\times(r+1)/2$; and (2) the cost for $x^{(+)}$ is now at least $r+1$, which is more than r; and the cost for $x^{(-)}$ remains 1.) ∎

Proof of Theorem 15: We know that the $L^{(k)}$ learning algorithm (Figure 2) can PACT-learn-learn $\mathcal{A}^k$ under complete blocking, using

$$h(\,k,\,\epsilon,\,\delta\,) \quad = \quad \frac{err_M^2\,2^{4k+5}}{9\epsilon^2}\ln\frac{4\,g(k)}{\delta} \tag{13}$$

completely-specified, labeled instances. Our LAC^* on-line learning algorithm uses this algorithm:

As shown in Figure 5, LAC^* first draws $h_1 = h(n,\,1/2,\,1/2)$ instances, and for each, pays the cost $\sum_i c_i$ to fill-in all of their attribute values. It then guesses T if $\mathrm{err}(T,F) < \mathrm{err}(F,T)$, and F otherwise; and so pays at most a penalty of

$$err_{min} \quad = \quad \min\{\mathrm{err}(T,F),\ \mathrm{err}(F,T)\}$$

This phase, therefore, costs at most $h_1 \;\times\; W$ where $W = err_{min} + \sum_i c_i$. It then uses these now completely-specified and labeled instances to produce an active classifier α_1 (using $L^{(n)}$); by construction, with probability at least $1/2$, α_1's cost will be within $1/2$ of the best — *i.e.*, $EC_P(\alpha_1) - EC_P(\alpha^{opt}) \;\leq\; 1/2$. We then exploit this pretty-good classifier, using it to deal with the next batch of

$$Y_1 = h_1[2^1 W - 1]$$

instances. After these $m_1 = h_1 + Y_1$ instances, we are 50% confident that the average difference is only

$$\mathit{diff}(LAC^*,\ m_1) \;=\; \frac{h_1\times[W - EC_P(\alpha^{opt})] \;+\; Y_1\times 1/2}{h_1 \;+\; Y_1} \;\leq\; 1$$

The LAC^* algorithm then repeats this "explore then exploit" cycle, but with tighter bounds: Here, the exploration phase draws $h_2 = h(n,\,1/4,\,1/4)$ instances, pays $\sum_i c_i$ to get the values of all attributes, and then accepts a err_{min} penalty for guessing the "safer" option. It then uses this sample to produce the α_2 active classifier, then exploits this α_2 for the next $Y_2 = [2^2 W - 1]\times(h_1 + h_2 + Y_1)$ instances. Here, we are $1 - 1/4$ confident that this final batch of instances will have expected error at most $1/4$. As we are not assuming that the first batch was within $1/2$ of optimal, we see that, after $m_2 = h_1 + Y_1 + h_2 + Y_2$ instances, the difference is

$$\mathit{diff}(LAC^*, m_2) \;\leq\; \frac{(h_1 + Y_1 + h_2)\times[W - EC_P(\alpha^{opt})] \;+\; Y_2\times 1/4}{h_1 \;+\; Y_1 + h_2 + Y_2} \quad\leq\quad \frac{1}{2}$$

In general, our LAC continues with this explore-exploit loop — on the k^{th} cycle, it draws $h_k = h(n, 1/2^k, 1/2^k)$ instances, pays to see the values of all of the attribute and guesses the least-risk label, then uses these instances to produce an active classifier α_k whose cost is, with probability at least $1 - 1/2^k$, within $1/2^k$ of optimal. It then exploits this α_k for the next $Y_k = [2^k W - 1] \times [\sum_{i=1}^{k} h_i + \sum_{i=1}^{k-1} Y_i]$ instances. It is easy to confirm that, after $m_k = \sum_{i=1}^{k} h_i + \sum_{i=1}^{k} Y_i$ instances, the average difference is, with probability at least $1 - 2^k$,

$$diff(LAC^*, m_k) \quad \leq \quad \frac{1}{2^{k-1}}$$

Note that, as k grows, we become increasingly confident that the resulting α_k will be better, at a rate that insures that the running average difference is also going to 0.

Notes: Of course, in practice there are several things we could do to produce a more effective on-line learning algorithm. For example, rather than just return the least risk label (T or F) in the exploration phase of stage $k+1$, we could instead use α_k to produce a label.

Also, we don't have to use $h(\underline{n}, 1/2, 1/2)$ on the first round; we could instead grow the depth of the tree, in parallel with decreasing the ϵ and δ terms; *i.e.*, use $h'_1 = h(1, 1/2, 1/2)$, then $h'_2 = h(2, 1/4, 1/4)$, $\ldots h'_k = h(k, 1/2^k, 1/2^k)$, $\ldots$ until reaching $h'_n = h(n, 1/2^n, 1/2^n)$, and then after leaving the tree depth at n and only updating ϵ and δ. ∎

References

[Aha97] D. Aha. Special issue on "Lazy Learning". *Artificial Intelligence Review*, 11(1–5), February 1997.

[AHM95] P. Auer, R. C. Holte, and W. Maass. Theory and applications of agnostic PAC-learning with small decision trees. In *ICML-95*, pages 21–29, 1995.

[Ang87] D. Angluin. Learning regular sets from queries and counterexamples. *Inform. Comput.*, 75(2):87–106, 1987.

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

[Ang92] D. Angluin. Computational learning theory: survey and selected bibliography. In *STOC-92*, pages 351–369, 1992.

[BCJ93] A. Blum, P. Chalasani, and J. Jackson. On learning embedded symmetric concepts. In *Proc. 6th Annu. Workshop on Comput. Learning Theory*, pages 337–346. ACM Press, New York, NY, 1993.

[BDD93] S. Ben-David and E. Dichterman. Learning with restricted focus of attention. In *COLT93*, pages 287–296, 1993.

[BJ98] D. Basri, R. Roth and D. Jacobs. Clustering appearances of 3d objects. In *CVPR'98, The IEEE Conference on Computer Vision and Pattern Recognition*, pages 414–420, 1998.

[Blu94] A. Blum. Relevant examples and relevant features: Thoughts from computational learning theory. In *AAAI Fall Symposium on 'Relevance'*, 1994.

[BMSJ78] B. G. Buchanan, T. M. Mitchell, R. G. Smith, and C. R. Johnson, Jr. Models of learning systems. In *Encyclopedia of Computer Science and Technology*, volume 11. Dekker, 1978.

[CAL93] D. Cohn, L. Atlas, and R. Ladner. Improved generalization with active learning. *Machine Learning*, 15(2):201–221, 1993.

[Che52] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sums of observations. *Annals of Mathematical Statistics*, 23:493–507, 1952.

[DLR77] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistics Society, B*, 39:1–38, 1977.

[EZR00] Y. Even-Zohar and D. Roth. A classification approach to word prediction. In *NAACL-2000, The 1st North American Conference on Computational Linguistics*, pages xx–yy, 2000.

[FSST97] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28:133, 1997.

[GGK97] Russell Greiner, Adam Grove, and Alex Kogan. Knowing what doesn't matter: Exploiting the omission of irrelevant data. *Artificial Intelligence*, December 1997. http://www.cs.ualberta.ca/~greiner/PAPERS/superfluous-journal.ps.

[GGR96] R. Greiner, A. Grove, and D. Roth. Learning active classifiers. In *ICML-96*, Bari, Italy, 1996. Morgan Kaufmann.

[GGS97] Russell Greiner, Adam Grove, and Dale Schuurmans. Learning Bayesian nets that perform well. In *Uncertainty in Artificial Intelligence*, 1997.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, 1979.

[GKS97] S. A. Goldman, S. Kwek, and S. D. Scott. Learning from examples with unspecified attribute values. In *Proc. 10th Annu. Conf. on Comput. Learning Theory*, pages 231–242. ACM Press, New York, NY, 1997.

[GO96] R. Greiner and P. Orponen. Probably approximately optimal satisficing strategies. *Artificial Intelligence*, 83(1), May 1996.

[GR99] A. R. Golding and D. Roth. A Winnow based approach to context-sensitive spelling correction. *Machine Learning*, 34(1-3):107–130, 1999.

[Hau92] D. Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1):78–150, 1992.

[HBR94] D. Heckerman, J. Breese, and K. Rommelse. Troubleshooting under uncertainty. In *International Workshop on Principles of Diagnosis*, 1994.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.

[Hol93] R. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63–91, 1993.

[JKP94] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *ICML-94*, pages 121–129, 1994.

[KLPV87] M. Kearns, M. Li, L. Pitt, and L. G. Valiant. On the learnability of boolean formulae. In *STOC-87*, pages 285–295, 1987.

[KMT93] S. R. Kulkarni, S. K. Mitter, and J. N. Tsitsiklis. Active learning using arbitrary binary valued queries. *Machine Learning*, 11:23–35, 1993.

[KR95] R. Khardon and D. Roth. Learning to reason with a restricted view. In *Proc. 8th Annu. Conf. on Comput. Learning Theory*, pages 301–310, 1995.

[KR97] R. Khardon and D. Roth. Learning to reason. *Journal of the ACM*, 44(5):697–725, Sept. 1997.

[KR99] R. Khardon and D. Roth. Learning to reason with a restricted view. *Machine Learning*, 35(2):95–117, 1999.

[KS90] M. Kearns and R. Shapire. Efficient distribution-free learning of probabilistic concepts. In *FOCS-90*, 1990.

[Lit88] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[LR87] J. Little and D. Rubin. *Statistical Analysis with Missing Data.* Wiley, New York, 1987.

[PP91] G. Provan and D. Poole. The utility of consistency-based diagnostic techniques. In *KR-91*, pages 461–72, 1991.

[Qui89] J. R. Quinlan. Unknown attribute values in induction. In *ICML-89*, pages 164–168, 1989.

[Rot95] D. Roth. Learning to reason: The non-monotonic case. In *IJCAI-95*, pages 1178–1184, 1995.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press (A Bradford Book), Cambridge, MA, 1998.

[SG94] D. Schuurmans and R. Greiner. Learning default concepts. In *CSCSI-94*, pages 519–523, 1994.

[SUB96] M. Salganicoff, L. H. Ungar, and R. Bajcsy. Active learning for vision-based robot grasping. *Machine Learning*, 23:251–278, 1996.

[Tur95] P. D. Turney. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of AI Research*, 2:369–409, 1995.

[Val84] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–42, 1984.