To Appear in Artificial Intelligence Journal

Learning Cost-Sensitive Active Classifiers*

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

Adam J. Grove
Netli Inc.
844 E. Charleston Rd
Palo Alto 94303
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 some unspecified attributes, before deciding upon a class label. This can be useful, for instance, when deciding whether to gather information relevant to a medical procedure or experiment. The expected utility of using an active classifier depends on both the cost required to obtain the values of additional attributes 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, this task of learning optimal active classifiers is often intractable.

Keywords: learning cost-sensitive classifiers, decision theory, reinforcement learning, PAC-learnability

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

*This extends the short conference paper [GGR96].

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" in the sentence "`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 interpreter 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 consider *active classifiers*: functions that, given a partially specified instance, return either a class label or a strategy that specifies which test should be performed next (and recur).

To make this more concrete, suppose a "hepatitis" classifier is only initially told 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 this active classifier 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 yet 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, a camera platform on a mobile robot must decide whether it is worth the expense of moving the camera to obtain a better view [BRJ98]; and second, an image analyzer must decide which operators to use in analyzing a single pose [SUB96, CAL93].

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 additional 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, 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. A 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 phase" approach: first learning the underlying concept and then, in a separate phase that does not involve learning, finding the best active classifier.

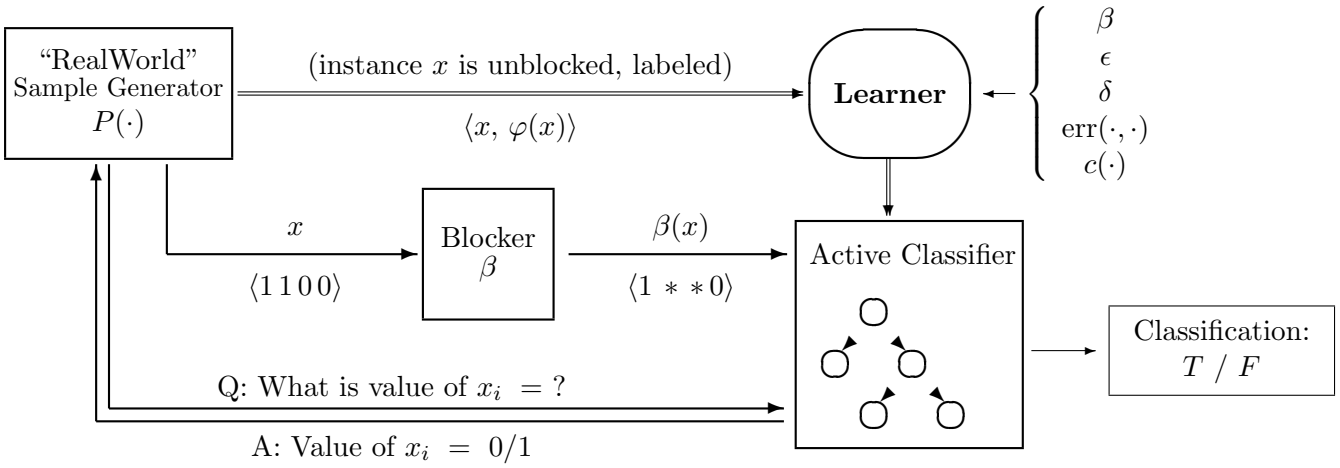

Figure 1: Active Classification Framework

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 results are a mix of good and bad news. 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. Section 6 extends our basic framework. Our current framework charges the classifier for each value it requests and receives, but gives the learner this information for free. This section 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 connects our framework with "episodic reinforcement learning", and 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 assume that all attributes, as well as the classification itself, are binary.[1] Thus we can identify each domain instance with a finite vector of Boolean attributes $x = \langle x_1, \ldots, x_n\rangle$, and let $X = \{0,1\}^n$ be the set of all possible domain instances. As each instance either belongs to the underlying concept or not (*e.g.*, the individual with a particular set of attributes either does, or does not, have hepatitis), we can view this concept φ as an indicator function $\varphi\colon X \mapsto \{T, F\}$, where $x \in X$ is a member of φ iff $\varphi(x) = T$. We

[1]We will later extend our results to non-binary attributes and class labels; see Corollary 8. 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, in addition to T and F.

assume that the learner knows the set of possible concepts, $\mathcal{C} = \{\varphi_i\}$, as well as how each concept is expressed; see discussion below.

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 $P\colon X \mapsto [0, 1]$ 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 "Hepatitis" 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 would test negatively, but whose jaundice and liver tests (x_1 and x_3) would both be positive.[2] 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 suggests that the classifier has to pay for the value of each attribute it sees (see Section 2.1), as the classifier initially sees nothing (read "$\langle *, *, \ldots, *\rangle$"). We will refer to this as *empty blocking*.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 questionnaire, or from a low-level feature extractor that is always run.) In general, we assume there is a separate *blocking process* β that, for each instance, first stochastically selects a subset of the attributes, and then reveals the values of those attributes to the classifier, for free. (See [SG94] for a more general discussion of the blocking models in general.) Under any blocking model, the active classifier is subsequently allowed to obtain (at a price) the values of the remaining blocked attributes. This leads to the framework suggested by Figure 1.

Definition 1 (Active Classifier) *An active classifier is a function*

$$\alpha\colon \{0, 1, *\}^n \mapsto \{T,\ F,\ 1,\ 2,\ \ldots,\ n\}$$

where $\alpha(\langle x_1, x_2, \ldots, x_n\rangle) = T$ (resp., F) means the classifier returns the categorical answer T (resp., F) given partial instance $\langle x_1, x_2, \ldots, x_n\rangle$. 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. We require that if $\alpha(\langle x_1, x_2, \ldots, x_n\rangle) = i$ then $x_i = *$. (Once the value for x_i has been provided, the active classifier then recurs on the now-more-completely instantiated instance.)

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

[2] *N.b.*, we are not committing to performing these tests. We are merely stating the outcomes of these tests on the current patient, if these tests are performed.

classifier then calls itself on the result, say $\langle 1, 0, * \rangle$, perhaps to return $\alpha(\langle 1, 0, * \rangle) = F$. In 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 $\mathcal{A} \subseteq \mathcal{A}^{all}$. Generally, such $\mathcal{A}$ should be more than simply a restricted subset of $\mathcal{A}^{all}$; it should be a "programming language" in that it specifies a representation and a computational model for its members. Thus, we can consider 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 execute quickly. 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 $\mathcal{A}$.[3] In this paper we restrict our 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 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 fewer than 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).

[3] 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 [Coo90].

It is easy to motivate this $\mathcal{A}^k$ class. 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. Similarly, many text, speech and image processors need to run real-time; here again it is reasonable to limit the amount of time used before making a decision.

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 $\mathrm{err}(T,\ T) = \mathrm{err}(F,\ F) = 0$.[4] To avoid degeneracy, we also assume that $\mathrm{err}(T,\ F) > 0$ and $\mathrm{err}(F,\ T) > 0$, and also $c_i \geq 0$. 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.[5]

Suppose that $x^* \in \{0, 1, *\}^n$ represents the active 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.

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 $tc_\alpha(x, x^*) = c(i) + tc_\alpha(x, x^*_{i \mapsto x[i]})$ where $x^*_{i \mapsto x[i]}$ 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 $x^*_{2 \mapsto 0} = \langle *, 0, *\rangle$, because α sets x^*_2 to $x[2]$, i.e., to 0. If we suppose further that $\alpha(\langle *, 0, *\rangle) = F$, and $\varphi(\langle 1, 0, 1\rangle = T$, then

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

[4] In some situations the classifier may have to pay a positive cost even if the classification is correct. However, as the classifier will eventually have to pay exactly one of $\{\mathrm{err}(F,\ F),\ \mathrm{err}(T,\ F)\}$ for every negative instance, we need only consider the *difference* between these values when deciding on the optimal active classifier. We can therefore shift the values, re-setting $\mathrm{err}(T,\ F)$ to this difference, and re-setting $\mathrm{err}(F,\ F)$ to 0. The same argument shows that we can view $\mathrm{err}(T,\ T) = 0$.

[5] While the condition $c_i \leq \min\{\mathrm{err}(T, F),\ \mathrm{err}(F, T)\ \}$ is sufficient for this situation, we use err_M (based on max) as err_M is later used in several other proofs. Also, to avoid a possible confusion: each c_i quantity is the amount the active classifier will have to pay *if it asks* to see the value of the i^{th} attribute. The classifier is not charged if a "non-empty blocker" reveals the value of this attribute initially.

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

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

(Of course, this "$\beta(x)$" term is an abuse of notation, as in general β is stochastic. Also, to further simplify the notation, we omit β from $EC_P(\alpha)$, as we assume this stochastic blocker is fixed.)

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 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 $\mathcal{A}$:

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

(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*

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

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 L 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 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 the 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?[6].

The class of active classifiers, $\mathcal{A} \subseteq \mathcal{A}^{all}$, 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; $\alpha(\langle x_1, x_2, \ldots, x_n \rangle) = \mathit{argmin}_i\{e_i(\langle x_1, x_2, \ldots, x_n \rangle) = 1\}$. 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 it is beyond the scope of 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 fixed and known.) If we

[6] 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 [Tur95, GO96].) This extension would, however, force a corresponding generalization to our definition of an active 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?).

know all of these components, then we are faced with a very interesting optimization problem [How66, 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.

Proposition 4 *Suppose we know the concept φ and the* product *distribution over instances P, and are looking for classifiers in $\mathcal{A}^k$ (i.e. active classifiers that ask for at most k attribute values; see 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 in $\mathcal{A}^k$ 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:[7]

Theorem 5 *For some $\mathcal{A}$, P and $\mathcal{C}$, it is NP-hard to find the optimal $\alpha^* \in \mathcal{A}$. This result holds even if we further require that that $\mathcal{C}$ is PAC-learnable, that $|\alpha^*|$ is polynomial in $|\varphi|$ (i.e., the complexity is not simply because we need a very large classifier), and that P have support of size $O(n)$, where n is the number of attributes. (That is, $P(x) > 0$ for only $O(n)$ different atomic assignments "x"s.)*

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. (Notice this holds for any blocker β.) ∎

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

[7] Recall that proofs of the theorems appear in the Appendix.

fail (for complexity reasons) in various ways. *This 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 — is conceptually related to the direction developed by Khardon and Roth [KR97] in their *Learning to Reason* framework. In the context of logical reasoning (rather than classification) they show that there are computational advantages in directly learning a representation tailored to the reasoning task (rather than trying to learn the concept itself and then, in a separate phase, perform the reasoning with respect to it. In a similar vein, our work is also consistent with results showing that discriminant learning can be more efficient than generative learning; see [Rip96].

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 need to learn the full concept. For example, suppose we are considering the empty blocking cases (*i.e.*, the classifier initially sees no attribute values), and $\mathrm{err}(\cdot,\cdot)$ 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 completely specified (unblocked) instances. But this is simply irrelevant: as we know that the instances will be presented empty 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 learning framework, and especially the PAC-learning model, usually avoids having to explicitly learn a distribution, because the performance criterion uses the same distribution that one learns under. If one has a (passive) classifier that fits the sample data well, one can often assume 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. (That is, an active classifier that performs well on training data, will do well on future test data.)

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!

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

We start the presentation of the main result by considering first the case of "empty 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 empty 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 partially-specified 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 $P^{i \mapsto 0}_{x^*} = P(\, x^*_{i \mapsto 0} \,|\, x^* \,)$, 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 $P(\, \varphi(x^*) = T \,|\, x^* \,)$, and $\widehat{P^{i \mapsto 0}_{x^*}} = \#[x^*_{i \mapsto 0}] \,/\, \#[x^*]$ is the empirical estimate of $P(\, x^*_{i \mapsto 0} \,|\, x^* \,)$. 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 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 $C(\, x^*, T \,)$ (here $\widehat{C}(\,\cdot\,)$ 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.

```
Algorithm L^(k)( ε ∈ ℜ+, δ ∈ (0,1) ) : α ∈ A^k
    % Returns an active classifier α from A^k whose expected cost is, with probability at least 1 − δ,
    %          within ε of optimal
    % Uses oracle for drawing complete labeled instances, and knows cost model, err(·,·), c(·)
  Let g(k) = Σ_{i=0}^{k} (n choose i) 2^i        % See Equation 1
  Draw  M(k) = (err_M^2 4^{2k+3} / 9ε^2) log (4 g(k) / δ)   completely-specified, labeled instances S
  Return Helper_L^(k)( S, ⟨∗,...,∗⟩ )
End Algorithm L^(k)

Algorithm Helper_L^(k)( S : sample; x^start : instance) : α ∈ A^k
  % Uses { After[·] : a list of g(k) real-numbers,
         { Op[·]    : a list of g(k) "operations"   (each Op[j] ∈ {T, F, 1, ..., n}),
  %   whose elements are "indexed" by partial assignments from X*_{0..k}
  % See text for definitions of P̂^φ_{x*} and P̂^{i↦0}_{x*}
  For each ℓ = k..0 do
    For each x* ∈ X*_ℓ do                                                          [L0]
      Ĉ(x*, T) := (1 − P̂^φ_{x*}) × err(T, F)                                       [L1]
      Ĉ(x*, F) := P̂^φ_{x*} × err(F, T)                                             [L2]
      If ℓ < k, then  For each i = 1..n where x*_i = ∗ do
        Ĉ(x*, i) := c_i + P̂^{i↦0}_{x*} × After[x*_{i↦0}] + (1 − P̂^{i↦0}_{x*}) × After[x*_{i↦1}]   [L3]
      Op[x*] := argmin{ Ĉ(x*, z) | z ∈ {T, F, 1, ..., n} }
      After[x*] := min{ Ĉ(x*, z) | z ∈ {T, F, 1, ..., n} }
  Return BuildTree( Op[·], ⟨∗,...,∗⟩ )
End Algorithm Helper_L^(k)

Algorithm BuildTree( Op[·]: list_of_operations; x* ∈ 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 ∈ {1, ..., n}
    Label n.Action := "Test x_i"
    Let n.ifTrue := BuildTree( Op[·], x*_{i↦1} )
    Let n.ifFalse := BuildTree( Op[·], x*_{i↦0} )
  return( n )
End Algorithm BuildTree
```

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

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 in 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 cost.

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 specified (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 true 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 estimates 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 *For any fixed k, the algorithm $L^{(k)}$ (Figure 2) PACT-learns active classifiers in the set $\mathcal{A}^k$ given empty blocking for* any *concept φ and* any *distribution P. Moreover, its run complexity is $O(\left(\frac{err_M}{\epsilon}\right)^2 n^k \left[\log n + \log \frac{1}{\delta}\right])$.*

While this specific $L^{(k)}$ algorithm is geared to binary classification over a set of binary attributes, these ideas can be extended to handle active classifiers that deal with finite classifications, over a space of finite-domain attributes. In particular, assume each attribute ranges over s values, and each instance is labeled by one of r values $\{\ell_1, \ldots, \ell_r\}$. We will need, as input, a general $r \times r$ "error matrix", whose $\langle i, j \rangle$ entries — a.k.a. $\mathrm{err}(\ell_i, \ell_j)$ — define the penalty for returning the label ℓ_i when the correct label should be ℓ_j. Let $\mathrm{err}_M = \max_{i,j}\{\mathrm{err}(\ell_i, \ell_j)\}$ be the maximum of these r^2 values. We now define ${\mathcal{A}^k}_{r,s}$ to be the class of classifiers that can ask at most k questions (starting from empty blocking) over these active classifiers — which in general can ask for the value of any of n attributes, or return any of r possible values.

We can still PACT-learn in this situation:

Corollary 8 *For any fixed k, it is possible to PACT-learns active classifiers in the set ${\mathcal{A}^k}_{r,s}$ given empty blocking for* any *concept φ and* any *distribution P. Moreover, its run complexity is $O(\left(\frac{err_M}{\epsilon}\right)^2 (s\,n)^k \left[\log(sn) + \log \frac{r}{\delta}\right])$.*

Comparing the computational complexity here to Theorem 7, we see that the complexity scales as $s^k \log s$ with respect to the number of attribute values s, but only as $\log r$ with respect to the number of classes. (This is because the obvious dynamic program needs to consider all of the partial instances that specify at most k attributes.)

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

$$X^*_\ell(x^{start}) \;=\; \{\, x^* \in X^* \mid x^* \text{ extends } x^{start}, \text{ specifying } \ell \text{ additional attributes} \,\}$$

which denotes the set of all $O(2^\ell \binom{n}{\ell})$ instances formed by starting with the starting instance $x^{start} \in X^*$ and specifying the values of exactly ℓ of its initially-uninstantiated variables. (Hence $X^*_\ell(\langle *, \ldots * \rangle) = X^*_\ell$.) (Actually, our bound is slightly tighter, as we need only consider the $O(2^\ell \binom{(n-|x^{start}|)}{\ell})$ possible patterns formed by instantiating the $n - |x^{start}|$ attributes not specified initially.)

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 phase, 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}. Applying the technique from the proof of Theorem 7 requires that we be able to estimate these $2\sum_{\ell=0}^{k} |X^*_\ell(x^{start})| \leq 2g(k)$ values, associated with *any* possible starting instance $x^{start} \in X^*$. The only challenge is collecting a sufficiently large sample to achieve this. In fact, the short proof below goes further and shows that it is possible to collect a sample large enough to estimate $P(\,\varphi(x^*) = T \,|\, x^*\,)$ and $P(\,x^*_{i\mapsto 0} \,|\, x^*\,)$ 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

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

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 9 *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 the proof of Theorem 7) that the classifier is effective whenever it can reliably (*i.e.*, with collective probability at least $1-\delta$) estimate the values of $P(\,\varphi(x^*) = T \,|\, x^*\,)$ and $P(\,x^*_{i\mapsto 0} \,|\, x^*\,)$ for each $x^* \in X^*_\ell(x^{start})$ to within $\lambda = \frac{3\epsilon}{8\, err_M\, 4^k}$. 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, a classifier in $\mathcal{A}^k$ only requires the algorithm to estimate $O(n^k)$ probabilities; this means computational complexity is not a problem. The second issue deals with the sample size. Unlike the case of

a fixed starting point, here we may need to estimate an *exponential* number of probabilities, which means the sample size required to guarantee good estimates will be larger. However, since the sample size depends only logarithmically on the number of probabilities we need to estimate, the sample size remains polynomial.

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 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.[8] 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 [BCJ93]. However, given that some very good heuristics exist for learning decision trees, it could be interesting to investigate modifying those to produce 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 for the time complexity of the algorithm (at least in the algorithmic approach we suggested). 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 10 *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 9, 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 $\log n$, and hence of size $O(2^{\log n}) = O(n)$. (It is binary as each internal node asks for the value of a binary attribute.) ∎

That is, the learner uses a reasonable number of instances and (eventually!) outputs a small, and 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, as that effort will be well rewarded. See also Section 6.

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

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 11 *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 any active classifier, which can ask for an arbitrary number of attribute values;* $\mathcal{A}^{(n)} \equiv \mathcal{A}^{all}$*.)*

The greedy algorithm L^G for learning conjunctions is shown in 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. This results in a "linear" active classifier, on the form shown in Figure 4. 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 $(x_1 \oplus x_2) \wedge x_3 \wedge x_4 \wedge \cdots \wedge x_n$. The dependencies introduced by the exclusive-or "$\oplus$" mean the optimal active classifier will not be a simple linear-tree (Figure 4), which means the greedy heuristic shown in Figure 3 will not produce an optimal 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 $\mathcal{A}^{\approx k}$

Our earlier results consider 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 place an upper bound on the *average* number of queries needed or allowed. For example, imagine we need to classify 100,000 text documents within a total of 200,000 seconds. While the "classify each document within 2 seconds" requirement is sufficient, the weaker requirement that the classifier takes *on average* 2 seconds per document is a more accurate reflection of the given constraint.

Definition 12 *Given any instance distribution* $P(\cdot)$*, we define* $\mathcal{A}_P^{\approx k}$ *as the subset of active classifiers (in any representation language) that ask,* on average*, at most* k *questions for samples drawn from* $P(\cdot)$*.*

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 $M_{L^G} = 2\left(\frac{\mathtt{err}(F,T)}{\epsilon}\right)^2\left(n\log(3n) + \frac{2}{\delta}\right)$ 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 $q_i = \hat{P}(\, x_i = 0\,) = \frac{1}{|S|}|\{x \in S | x_i = 0\}|$ *% ...= estimate of success probability $P(\, x_i = 1\,)$*
Re-number s.t. $\frac{c_1}{q_1} \le \frac{c_2}{q_2} \le \ldots \le \frac{c_k}{q_k}$ *% ...Set to ∞ if $q_i = 0$*
Let $\ell = argmax\{\frac{c_{\ell'}}{q_{\ell'}} \le err_M \,|\, \ell' = 1..k\}$ *% ...= 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

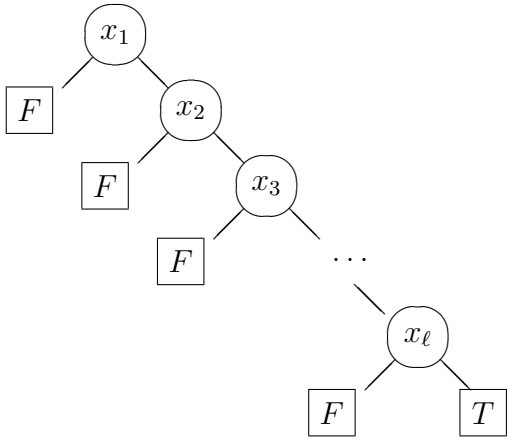

Figure 4: "Linear" Active Classifier (Decision Tree)

In general, we can bound the average number of queries asked by an optimal classifier as follows.

Definition 13 *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.

Then it is easy to see that:

Proposition 14 *The optimal active classifier from $\mathcal{A}^{all}$ should not ask* on average *more than $k(err, c)$ questions — i.e., it is in $\mathcal{A}_P^{\approx k(\mathrm{err},c)}$.*

Proof: Whenever the classifier asks more than $k(\mathrm{err}, c)$ questions, its cost exceeds err_M; hence if it averages more than $k(\mathrm{err}, c)$ questions, its average cost must exceed err_M. This cannot be optimal, as it is inferior to the trivial "just say F" classifier, whose average cost is at most err_M. ∎

However this does not mean that we can *bound* the number of questions by $k(\mathrm{err}, c)$ — *i.e.*, we cannot restrict ourselves to $\mathcal{A}^{k(\mathrm{err},c)}$.[9] 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.

Theorem 7 shows that the problem of learning in $\mathcal{A}^k$ is tractable for any concept and for any distribution. It may seem plausible that there is a similar result given a bound on the *expected* number of queries used by a good classifier.

When might we have such a bound? It could arise as an inherent property of the chosen classifier *language* $\mathcal{A}$, although the dependence of expectation bounds on the instance distribution $P(\ )$ means that this is not especially useful. More plausibly, such a bound may arise as an additional problem-specific constraint limiting the class of acceptable classifiers. Recall the earlier example, where we need to classify 100,000 text documents within a total of 200,000 seconds, and suppose each query takes a second. This places a global constraint on the classifier class—i.e., that any learned classifier must be in $\mathcal{A}_P^2$. Finally, we may be able to place a bound on the expected number of queries needed using Proposition 14 or similar considerations. Unfortunately, however we come by such a bound, we are faced with the following negative result:

Theorem 15 *PACT-learn is NP-hard even given the additional constraint that there is an optimal classifier in $\mathcal{A}^{\approx 1+\gamma}$, for any fixed $\gamma > 0$.*

[9] On the other hand, it is sufficient to consider classifiers in $\mathcal{A}^{k(\mathrm{err},c) \times err_M/\epsilon}$; see Proposition 18 in the Appendix. But then our dynamic-programming algorithm will be exponential in err_M/ϵ.

Class	sample size	computation	representation size
$\mathcal{A}^k$	poly	poly	poly
$\mathcal{A}^{\log n}$	poly	$\equiv$ PAC-learn $\log n$-depth DTs	poly
$\alpha^{opt} \in \mathcal{A}_P^{\approx k}$	poly	NP-hard ($k > 1$)	poly

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

(Note that this result does not apply when $\gamma = 0$, since $\mathcal{A}^{\approx 1}$ is equivalent to $\mathcal{A}^1$ and so 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 10?

5.4 Summary

We have shown that

- one *can* learn active classifiers in $\mathcal{A}^k$ in general (for any concept class, any blocker, any distribution)
- learning classifiers in $\mathcal{A}^{\log n}$ subsumes a hard computational problem (but neither sample complexity nor representation size are problematic)
- for certain classes of classifiers and distributions, learning classifiers in $\mathcal{A}^n$ can be tractable
- Given just the constraint that it suffices to look for classifiers in $\mathcal{A}^{\approx k}$, for $k > 1$, the problem of learning active classifiers remains NP-hard.

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. An suitable framework is "on line" learning, where a learn+active-classify (LAC) system would pay for each attribute it sees, from the very beginning. To evaluate such a system, we would compute a "loss function" comparing some system LAC against the perfect classifier α^{opt}, (i.e. against using the optimal active classification process from the beginning). The loss function measures the average difference between LAC and 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 empty 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

Algorithm $LAC^*()$

% *Continuously draws and processes instances*

% *Uses known cost model,* $err(\cdot,\cdot),\ c(\cdot)$

% *oracle for drawing empty-blocked labeled instances*

% $h(\,\epsilon,\,\delta\,) \quad = \quad M_n(\,\epsilon,\,\delta\,) \quad = \quad \frac{err_M^2\, 2^{4n+6}}{9\epsilon^2} \log \frac{4\, g(n)}{\delta}$ *(sample size for* $L^{(n)}$*)*

% $W = err_{min} + \sum_i c_i$

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

% r^{th} *exploration phase*

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

Let $h_r \;=\; h(\,\frac{1}{2^r},\,\frac{1}{2^r}\,)$

For $i = 1..h_r$ do

"Draw" an empty blocked instance x^*

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

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

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

$S_r \;=\; S_r \;\cup\; \langle x,\, \varphi(x)\rangle$

Use $L^{(n)}$ (with completely-specified, labeled instances S_r) to produce α_r

% r^{th} *exploitation phase*

Let $\quad Y_r \;=\; [2^r W - 1] \times \left[\sum_{i=1}^{r} h_i \;+\; \sum_{i=1}^{r-1} Y_i \right]$

Use α_r to process next Y_r instances

End For

End Algorithm LAC^*

Figure 5: LAC^* Learn+Classify Algorithm

penalty/cost that LAC must pay. Given a sequence of m such instances $X_m = \langle x^{(i)*} \rangle_{i=1}^{m}$, we define $tc(LAC, X_m) = \frac{1}{m} \sum_{x^{(i)*} \in X_m} tc(LAC, x^{(i)*})$ as the average cost. For a given (stationary) distribution over instances $P(\cdot)$ and target concept φ (and given the empty blocker), 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, $\mathcal{A} \subseteq \mathcal{A}^{all}$.) As this is clearly the best that any learn+classify systems can achieve, we will consider the difference

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

We would like an on-line learn+classify system that can do essentially as well as this optimal classifier, in that $\mathit{diff}(LAC^*, m)$ 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 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 16 *With probability* 1, $\mathrm{diff}(LAC^*, m)$ *goes to* 0 *as* m *increases.*

While this algorithm deals with empty blocking, it is straightforward to see how to extend this result to arbitrary blocking models and subclasses $\mathcal{A} \subset \mathcal{A}^{all}$.

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 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.

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. In contrast, we assume Other research [BDD93, GKS97] (resp., [KR95, KR99]) considers the problem of learning from partially specified instances, but with the goal of later classifying *complete* instances, (resp., later reasoning with respect to the learned concept). *N.b.*, these other systems do not consider ways for the classifier to gather more information. Also, we assume that we can obtain the value of an attribute, once it is requested.

Many researchers are concerned with "relevance". Littlestone [Lit88], John *et al.* [JKP94], and others consider the situation where only a subset of the variables are "relevant" — *i.e.*, 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. (By contrast, the Greiner *et al.* [GGK97] relevance model considers the case where the learner knows it will only see 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; there is a rich history of ideas here, dating back (at least) to the seminal work by Howard [How66] on "value of information". Many diverse areas use related ideas, including planning, diagnosis and decision theory.

As just one illustrative example, Heckerman *et al.* [HBR94] describe how to translate a certain class of decision nets (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, BRJ98]. 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.

Other Comparisons: Our results are related to "behavioral cloning" [SHKM92, SMB95,

SGD97], where the learner sees a (hopefully good) active classifier in action, and produces a classifier (or in general, a performance system) that tries to duplicate its performance [Kha99]. In our model, however, the learner must use the cost structure to discover its own classification strategy, rather than simply imitate the observed teacher's strategy.

Finally, our framework shares much in common with reinforcement learning (RL) [SB98]; especially *episodic* undiscounted RL. In each framework, 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"). As such, our active classifiers can be viewed as policies, as they each map states to actions (here, from $\{t, f, *\}^n$ to one of $\{A_1, \ldots, A_n, t, f\}$). Moreover, our basic learning algorithm is a variant of dynamic programming, just like many reinforcement learners. Our results show that, while this special case of episodic reinforcement learning (with fixed known depth) is "easy", it is hard to extend this to more general situations — *e.g.*, where there is a bound on the *average* number of steps in an episode.[10]

8 Conclusions and Future Work

In this paper, we have proposed a framework for addressing *learning* and *active classification* together. We anticipated that we might obtain 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 the distribution. Our results support this thesis. We show that we can efficiently learn active classifiers in cases in which we do not know how to learn the underlying concept and distribution separately. There are several possible directions that may yield further positive results, including (1) other restrictions on the type of active classifiers allowed; (2) approximation techniques; and (3) combinations of restrictions on both the concept class and the distribution.

We have also explored an "on-line" version of this framework, where the *learner* incurs costs while it is 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 the concept class.* For example, while Theorem 7 and its corollary may be restrictive in some respects, they work 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 [KS94] 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

[10] While the "fixed known depth" situation does corresponds to a (episodic) MDP, the harder "bounded *average* number of steps" situation does not.

that are relevant to the classification of a complete instance might become irrelevant to an active classifier dealing with blocked instances, when the cost distribution structure imply that only a small number of the attributes can be tested. 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, or at least not to the extent that they matter in ordinary passive classifier learning theory.

Acknowledgments

We gratefully acknowledge receiving helpful comments from Mukesh Dalal, Sanjeev Kulkarni, Nick Littlestone, and Dale Schuurmans, as well as the superb, and extremely thorough, remarks from the reviewers. 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 a 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 a distribution over the binary variables $\{s_1, ..., s_m, x_1, ...x_{3r}\}$ where the x_i's are independent of each other, and each is 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 $\gamma = (1 - p^{3r})/(1 - p^3) = 8 \times (1 - 1/8^r)/7$.

$\Longrightarrow$**:** Assume there is an exact cover — w.l.o.g. call it $\{s_1, \ldots, s_r\}$. Now consider the active classifier α^* that simply asks 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") — see the decision tree in Figure 4.

To compute the expected cost: Note that

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

where Equation 6 uses the fact that, as this is an exact cover, s_j involves variables different from $s_1, \ldots, s_{j-1}$; 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

$$EC_P(\alpha^*) \quad = \quad (r \times (p^3)^r) + \sum_{i=1..r} i \times (1-p^3) \times (p^3)^{i-1} \quad = \quad (1-(p^3)^r)/(1-p^3)$$

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 the "$x_{jk} = 1$" branch of α: 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 structures of s_j's, as finding any $s_j = F$ immediately tells us that 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 an *always correct* classifier will have "length" $> r$. We first show that any such "always correct" active classifier will have cost strictly greater than γ.

We can assume, w.l.o.g., 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, we 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 probability at least p^{3r-1}, which means the probability of performing $r+1$ tests is p^{3r}; this adds a cost of $(r+1)p^{3r-1}$. Hence, the total cost of α 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 γ.

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 γ. As there is 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 active classifier 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 the classifier will return the wrong answer. The cost of this mistake, therefore, is at least $p^{3r-1} \times (1-p) \times \mathrm{err}(F, T) = \frac{2}{8^r} \frac{1}{2} \times 2\frac{1-1/8^r}{(7/8)\times 1/8^r} = 2\gamma > \gamma$. (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 $(\bigwedge_i s_i) \wedge \left(\bigwedge_j x_j\right)$) and hence is clearly PAC-learnable, and that the distribution has a linear representation since it is is just a product distribution. ∎

Theorem 7 uses the following lemma:

Lemma 17 *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 $a^{opt} = \text{argmin}_{a_i}\{\, c(a_i)\,\}$, and the "estimation-based" agent, who takes action $\hat{a} = \text{argmin}_{a_i}\{\, \hat{c}(a_i)\,\}$, is bounded by $2\beta_{max}$, where $\beta_{max} = \max\{\beta_i\}$.*

Proof:

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

∎

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^*.

Let $P_{x^*}^{i \mapsto 1} = P(\, Ext(x^*_{i \mapsto 1}) \,|\, Ext(x^*)\,)$ be the probability that attribute x_i has the value 1, given the partial instance x^* — *e.g.*, $P_{\langle 0,*,*\rangle}^{3 \mapsto 1} = P(\, \langle 0,*,1\rangle \,|\, \langle 0,*,*\rangle\,)$ is the probability that attribute x_3 will have value 1, given that we know attribute x_1 had value 0 — and $P_{x^*}^{\varphi} = P(\, \varphi(x) = T \,|\, x \text{ extends } x^*\,) = \sum_{x \in Ext(x^*)} P(\, \varphi(x) = T \,|\, x^*\,)$. Notice both $P_{x^*}^{i \mapsto 1}$ and $P_{x^*}^{\varphi}$ are *conditional probabilities* — each relative to the conditioning event that x^* occurs (with probability $P(\, x^*\,) = \sum_{x \in Ext(x^*)} P(\, x\,)$). The $\widehat{P_{x^*}^{\varphi}}$ and $\widehat{P_{x^*}^{i \mapsto 1}}$ 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

$$M \quad = \quad M(k, \epsilon, \delta) \quad = \quad \frac{err_M^2 2^{4k+6}}{9\epsilon^2} \log \frac{4\, g(k)}{\delta} \tag{8}$$

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 $X_i \in [0,1]$ with common mean μ, will be far from μ:

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

Now consider a fixed $x^* \in X^*_{0..k}$, and let the $X_i^{x^*}$ variable be 1 if a randomly drawn instance will extend x^*, and 0 otherwise. 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 more than $\frac{3\epsilon}{8\, err_M\, 4^k}$ off is

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

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.*,

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

Now let α^{opt} be the 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

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

holds for all $x^*_\ell \in X^*_\ell$ (*i.e.*, for all partial instances that include exactly ℓ specified values), as this means, in particular, $|\, \textsc{After}(\, \widehat{\alpha},\, x^*_0\,) \ -\ \textsc{After}(\, \alpha^{opt},\, x^*_0\,)\, | \ =\ |\, \hat{P}(\, x^*_0\,) \times \textsc{After}(\, \widehat{\alpha},\, x^*_0\,) \ -\ P(\, x^*_0\,) \times \textsc{After}(\, \alpha^{opt},\, x^*_0\,)\, | \ \leq\ \frac{\epsilon\ (4^{k+1-0}-1)}{4^{k+1}} \ <\ \epsilon$ as desired.

We prove Equation 11 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 17 (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

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

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 17, 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 desired.

For the inductive step, we need to bound Δ_ℓ, 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 17, 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

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

(This uses the inductive assumption, our unproblematic assumption that $c_i \leq err_M$, our assumption (Equation 10) that $|\; \hat{P}(\, x^* \text{ occurs} \,) \;-\; P(\, x^* \text{ occurs} \,) \;| \leq \lambda$, and the fact that $\hat{P}(\, x^*_{i\mapsto 0}, x^* \,)$ is simply $\hat{P}(\, x^*_{i\mapsto 0} \,)$.)

Hence, from Lemma 17, we know that

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

as desired.

All that remains is to show that the $L^{(k)}$ algorithm is computationally efficient. $L^{(k)}$ needs to collect only a polynomial number, M, of instances to estimate the values of a polynomial number $2 \times g(k)$ of probability values, for $\widehat{P^{\varphi}_{x^*}}$ and $\widehat{P^{i \mapsto 0}_{x^*}}$. Given these estimates, Helper_$L^{(k)}$ needs to compute the $(2+n) \times 2g(k)$ values of $\widehat{C}(\, x^*, i\,)$, as well as the $2g(k)$ values of $Op[x^*]$ and AFTER$[\, x^*\,]$; each of these computations requiring constant time. Finally, BUILDTREE requires $O(2^k)$ time to build the binary tree of depth k. Hence, the total run time is $O(M(k) \times 2g(k) \; + \; (2+n) * g(k) \; + \; 2^k)$. As $g(k) \geq 2^k$, and $M(k) \geq n$, we have a run time of $O(g(k) * M(k)) = O(\left(\frac{err_M}{\epsilon}\right)^2 n^k [\ln n + \log 1/\delta])$, which is polynomial in the relevant quantities. (Here we view k as a constant.) ∎

Proof of Corollary 8: To accommodate r class labels and s values for each attribute, our $L^{(k)}_{r,s}$ algorithm (and subroutines) must compute, for each x^*, the r values $\widehat{P^{\phi=\ell_i}_{x^*}}$ (each an estimate of $P^{\phi=i}_{x^*} = P(\, \varphi(x) = \ell_i \,|\, x \; \textit{extends} \; x^*\,)$); and the $n \times s$ values $\widehat{P^{i \mapsto j}_{x^*}}$ to estimate $P^{i \mapsto j}_{x^*} = P(\, x^*_{j \mapsto i} \,|\, x^*\,)$

As shown in the proof of Theorem 7, these quantities will be estimated well enough if we can estimate the $r+1$ quantities $\hat{P}(\, x^* \text{ occurs}\,)$ and $\hat{P}(\, \varphi(x) = \text{class\#j}, \; x \text{ extends } x^*\,)$, for $j = 1..r$, for each of the

$$g_{r,s}(k) = \sum_{i=0}^{k} \binom{n}{i} s^i \;\; \leq \;\; k(s\,n)^k$$

partially-specified instances x^*; *i.e.*, if we can guarantee the chance that any one of these $(r+1)\, g_{r,s}(k)$ unconditional quantities is greater than $\lambda = \frac{3\epsilon}{8\, err_M\, 4^k}$ away from the correct value, is at most δ.

This requires at most

$$M_{r,s}(k) = \frac{err_M^2 4^{2k+3}}{9\epsilon^2} \log \frac{2\,(r+1)\, g_{r,s}(k)}{\delta} \;\; = \;\; O(16^k \left(\frac{err_M}{\epsilon}\right)^2 [k \log(sn) + \log \frac{r}{\delta}])$$

instances. Using the same arguments given above, the overall $L^{(k)}_{r,s}$ algorithm has computational complexity

$$O(M_{r,s}(k) \times g_{r,s}(k)) \;\; = \;\; O((s \times n)^k \left(\frac{err_M}{\epsilon}\right)^2 [\log(sn) + \log \frac{r}{\delta}])$$

as claimed. ∎

Proof of Theorem 11: 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. If (2) holds, then we know that

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

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 $q_i = \hat{P}(\, x_i = 0\,)$) will be more that $\epsilon/2$ off is under

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

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, with respect to 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, (where c_i is the cost of acquiring the value of x_i), consider a classifier α that does not — *e.g.*, that asks for x_{r+1} before x_r, where

$$\frac{c_r}{q_r} \quad < \quad \frac{c_{r+1}}{q_{r+1}} \tag{12}$$

For now, assume there is only a single violation. Here, using $p_i = \hat{P}(\, x_1 = 1\,)$, α's cost is $A + \ Q \times (c_{r+1} + p_{r+1}c_r) \ + Q \times p_{r+1} \times p_r \times B$, where A is the cost of dealing with the first portion of the classifier before reaching x_{r+1}, involving tests whose collective success probability is $Q = \prod_{i=1}^{r-1} p_i$, and B is the cost associated with the remainder of the classifier, after x_r. To show that α cannot be optimal, let α' be a classifier that differs from α only by exchanging these variables, placing x_r before x_{r+1}. Given Equation 12, the cost of α', which is $A + \ Q \times (c_r + p_r c_{r+1}) \ + Q \times p_r \times p_{r+1} \times B$, is under the cost of α. Note that the cost of classifier with more violations would be yet less.

Now let α_m be the tree-structured classifier that includes m variables, in this c_i/q_i order; we need only show that α_ℓ has the minimal expected cost — *i.e.*, that $m = \ell$. Consider sequentially growing the classifier tree. The classifier α_r is better than α_{r-1} when the difference between their costs $EC_P(\alpha_r) - EC_P(\alpha_{r-1}) = (\prod_{i=1}^{r-1} p_i) \times [c_r \ - \ q_r \mathrm{err}(F,T)]$ is negative — *i.e.*, when $c_r/q_r < \mathrm{err}(F,T)$. This means the optimal classifier will have all-and-only the first ℓ of the terms, as shown in Figure 3. ∎

Proposition 18 *Using $k = k(err, c)$ defined in Definition 13, there is an active classifier α_k^* in $\mathcal{A}^{k(\mathrm{err},c)\times err_M/\epsilon}$ whose expected cost is within ϵ of optimal; i.e., if α^* is a classifier with minimal expected cost,* $\mathrm{EC}_P(\alpha^*) < \mathrm{EC}_P(\alpha_k^*) + \epsilon$.

Proof: Let the random variable N represent the number of actions taken by an optimal strategy α^* on a specific example. Proposition 14 bounded the expected value of N: $E[N] \leq k$. Using the Markov Inequality, $P(\, N > k \times err_M/\epsilon\,) \leq E[N]/(k \times err_M/\epsilon) \leq \epsilon/err_M$. Now let α^*_k be the classifier formed by truncating α^* after $k \times err_M/\epsilon$ actions, and just returning, say, F here. Note that α^*_k is in $\mathcal{A}^{k(\mathrm{err},c)\times err_M/\epsilon}$. This α^*_k will be slightly worse than α^*: at most ϵ/err_M of the time, α^*_k will produce an error that is at most err_M. Hence, $EC_P(\alpha^*) < EC_P(\alpha^*_k) + \epsilon$, as claimed. ∎

Proof of Theorem 15: 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 $\langle Y, S\rangle$, with $|Y| = |3r|$ and $|S| = m$, we produce a PACT-learning instance whose m variables x_j correspond to the subsets $S = \{s_j\}_{j=1}^m$, and whose training instances (basically) correspond to the elements in Y. The concept to be learned is $\varphi \equiv x_1 \wedge \ldots \wedge x_m$.

We will use a total of $\frac{9\,r\,(r-1)}{2\gamma}$ training instances (including some duplicates; there are $3r+2$ distinct instances). Using $f = \frac{4\,\gamma}{3(r-1)}$, these instances are:

- $3r$ negative instances, $\{x^{(1)}, \ldots, x^{(3r)}\}$ where each $x^{(i)} = \langle x^{(i)}_1, \ldots, x^{(i)}_m\rangle$, where $x^{(i)}_j = 0$ iff $y_i \in s_j$ (and $= 1$ otherwise). (We assume that no y_i is in every s_j — for otherwise, the nonexistence of an exact cover would be immediate.) We include each of these samples one time, so the empirical probability of each is $f/(6r)$;
- one positive instance $x^{(+)} = \langle 1, 1, \ldots, 1\rangle$ included $3r$ times, so its empirical probability is $f/2$;
- one negative instance $x^{(-)} = \langle 0, 0, \ldots, 0\rangle$, included $3r[3(r-1) - 4\gamma]/(2\gamma)$ times, so its empirical probability is $1 - f$.

Let the cost of obtaining the value of attribute x_i be $c_i \geq 1$, and the penalty for being wrong be $R = \frac{9r(r-1)}{\gamma}$. We also assume empty blocking.

Claim: For every exact cover, there corresponds (in a sense discussed below) an active classifier in $\mathcal{A}^{\approx 1+\gamma}$ which classifies every training instance correctly. All other classifiers (i.e. that do not correspond to any exact cover in this sense) have average cost $\geq \min(2, 1 + \gamma + f/6r)$ on the training set.

⇐: If there is an exact cover $C = \{s_{i_1}, ..., s_{i_r}\} \subset S$, then form an active classifier α^C whose associated decision tree is "linear" (see Figure 4), with nodes labeled by the sets $s \in C$, going to "F" if $s_{i_j} = 0$, and further down if $s_{i_j} = 1$. The final internal node is s_{i_r}; its "1"-labeled arc leads to "T". The expected number of queries used by α^C is:

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

The first line uses the observation that $x^{(-)}$, which occurs with probability $P(\, x^{(-)}\,) = 1 - f$, will be answered (correctly) after examining the first node. The second line uses the fact

that $x^{(+)}$ requires examining all r nodes in the decision tree. The third line requires noting first that exactly 3 of the $x^{(i)}$s will reach the "false"-branch of each of the nodes in the tree. This means the total number of queries involved in handling all $3r$ of the $x^{(i)}$s is $P(\, x^{(i)}\,) \times 3 \sum_{j=1}^{r} j \;=\; \frac{f}{6r}\, 3\, r(r+1)/2$. Clearly this classifier always returns the correct answer.

$\Rightarrow$**:** First observe that if a classifier misclassifies even one sample point, then its cost is at least $f/(6r) \times R \;=\; \frac{1}{6r}\frac{4\gamma}{3(r-1)} \times \frac{9r(r-1)}{\gamma} \;=\; 2$, as required.

Thus we restrict attention to classifiers that classify all sample points correctly. We can further restrict attention to such classifiers that are equivalent to linear trees, i.e. that test some sequence $s_{i_1}, s_{i_2}, \ldots, s_{i_k}$ in order, stop and announce F if any $s_{i_j} = 0$, and announce T if all $s_{i_j} = 1$. If we are given any other classifier (e.g. that makes further tests after seeing some $s_{i_j} = 0$) we can modify it to construct a linear classifier that is as least as good, and so it suffices to prove the result for such linear trees. Note that, necessarily, $k > r$. For if $k < r$, the classifier must make an error on at least one $x^{(i)}$. And $k = r$, the classifier must either correspond to an exact cover, or must also make at least one misclassification.

Since $k > r$ and the first r tests do not correspond to an exact cover, then there must be at least one $x^{(i)}$ that reaches the $r+1$'th internal node (i.e. takes more than r tests to classify). In fact, the total number of tests to classify all $3r$ of the $x^{(i)}$ must be at least $P(\, x^{(i)}\,) \times ((3 \sum_{j=1}^{r-1} j) + 2r + (r+1)) \;=\; \frac{f}{6r}\,(1 + 3\, r(r+1)/2)$. Thus, the total number of tests used by this classifier is at least $1 + \gamma + f/6r$. Since each test costs at least $c_i \geq 1$, this verifies the claim.

To complete the proof of the theorem, consider X3C instances $\langle Y, S \rangle$ such that $s_i = \{y_{3i+1}, y_{3i+2}, y_{3i+3}\}$ for $i = 1, ..., r$. Thus, it is known that there is at least one exact cover. Clearly, given this knowledge, the problem of determining whether there exists any *other* exact cover remains NP-complete. Let the cost of querying s_i for $i \leq r$ be $1 + \frac{f}{12r(1+\gamma)}$, and the cost of querying any other s_i be 1. From the claim, we know that there is an optimal classifier in $\mathcal{A}^{\approx 1+\gamma}$ (because there is at least one exact cover). In fact, the cost of this classifier is $1 + \gamma + f/12r$. By the claim, this is better than any classifier *not* based on an exact cover. However, if there is a classifier based on any other exact cover (i.e. other than $\{s_1, ..., s_r\}$) then at least one query in this other classifier costs only $c_i = 1$. In the worst case, this cheaper query is the last query, but even then the expected cost will be $1 + \gamma + f/12r - c$ where $c = \frac{f^2}{24r^2(1+\gamma)}$. Thus, the ability to PACT-learn under these conditions, taking $\epsilon = c/2$, implies the ability to tell if there is more than one exact cover. This proves the result. ∎

Proof of Theorem 16: We know that the $L^{(k)}$ learning algorithm (Figure 2) can PACT-learn $\mathcal{A}^k$ under empty blocking, using

$$M_k(\, \epsilon,\, \delta\,) \quad = \quad \frac{err_M^2\, 2^{4k+6}}{9\epsilon^2} \log \frac{4\, g(k)}{\delta} \tag{13}$$

completely-specified, labeled instances, and so to consider all n attributes (i.e. $k = n$), it needs to use $h(\, \epsilon,\, \delta\,) = M_n(\, \epsilon,\, \delta\,)$ instances.

As shown in Figure 5, LAC^* first draws $h_1 = h(\, 1/2,\, 1/2)$ 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

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

This phase, therefore, costs at most $h_1 \times W$ where $W = err_{min} + \sum_i c_i$. It then uses $L^{(n)}$ and these now completely-specified and labeled instances to construct a classifier α_1 whose cost will be, with probability at least 1/2, 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

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

The LAC^* algorithm then repeats this "explore then exploit" cycle, but with tighter bounds: Here, the exploration phase draws $h_2 = h(\,1/4,\,1/4)$ 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. Even if we no longer assume that the first set of instances is within 1/2 of optimal, we see that, after $m_2 = h_1 + Y_1 + h_2 + Y_2$ instances, the difference is

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

In general, our LAC continues with this explore-exploit loop — on the r^{th} cycle, it draws $h_r = h(\,1/2^r,\,1/2^r)$ 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_r whose cost is, with probability at least $1 - 1/2^r$, within $1/2^r$ of optimal. It then exploits this α_r for the next $Y_r = [2^r W - 1] \times [\sum_{i=1}^{r} h_i \ + \ \sum_{i=1}^{r-1} Y_i]$ instances. It is easy to confirm that, after $m_r = \sum_{i=1}^{r} h_i \ + \ \sum_{i=1}^{r} Y_i$ instances, the average difference is, with probability at least $1 - 2^r$,

$$\textit{diff}(LAC^*, m_r) \quad \leq \quad \frac{1}{2^{r-1}}$$

Note that, as r grows, we become increasingly confident that the resulting α_r 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 $r+1$, we could instead use α_r to produce a label.

Also, we don't have to use $M_n(1/2, 1/2)$ 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 $h'_1 = M_1(1/2, 1/2)$, then $h'_2 = M_2(1/4, 1/4)$, $\ldots h'_r = M_r(1/2^r, 1/2^r)$, $\ldots$ until reaching $h'_n = M_n(1/2^n, 1/2^n)$, and then after leaving the tree depth at n and only updating ϵ and δ.

Finally, notice we use a completely different sample for each α_i; *i.e.*, we do *not* re-use S_i when learning α_j for any $j > i$. This proof did not explore subtle ways of re-using these S_is again later.

Notice that none of these tricks would change the correctness of the theorem's asymptotic claim. ∎

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.

[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.

[BRJ98] R. Basri, D. 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.

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

[Che52] H. 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.

[Coo90] G.F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2–3):393–405, 1990.

[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 124–131, 2000.

[FSST97] Y. Freund, S. Seung, E. Shamir, and N. Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28:133, 1997.

[GGK97] R. Greiner, A. Grove, and A. Kogan. Knowing what doesn't matter: Exploiting the omission of irrelevant data. *Artificial Intelligence*, pages 345–380, 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.

[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] W. 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.

[How66] R. Howard. Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, 2(1):22–26, 1966.

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

[Kha99] R. Khardon. Learning to take actions. *Machine Learning*, 35(1):57–90, 1999.

[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.

[KS94] M. Kearns and R. Schapire. Efficient distribution-free learning of probabilistic concepts. *Journal of Computer and System Sciences*, 48:464–497, 1994.

[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.

[Rip96] B. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, UK, 1996.

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

[SB98] R. S. Sutton and A. 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.

[SGD97] Tobias Scheffer, Russell Greiner, and Christian Darken. Why experimentation can be better than "perfect guidance". In *Proceedings of the Fourteenth International Conference on Machine Learning (IMLC97)*, Nashville, July 1997.

[SHKM92] C. Sammut, S. Hurst, D. Kedzier, and D. Michie. Learning to fly. In *ICML92*, Aberdeen, 1992.

[SMB95] B. Schulmeister, W. Müller, and M. Bleich. Modelling the expert's control behavior by machine learning algorithms. In *Proc. International Workshop on Artificial Intelligence Techniques*, 1995.

[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.