To appear in
Artificial Intelligence (tentatively) 91:1–2 (December 1997).

Knowing What Doesn't Matter: Exploiting the Omission of Irrelevant Data[*]

Russell Greiner
Siemens Corporate Research
755 College Road East
Princeton, NJ 08540-6632
greiner@scr.siemens.com

Adam J. Grove
NEC Research Institute
4 Independence Way
Princeton, NJ 08540
grove@research.nj.nec.com

Alexander Kogan
Rutgers University
Faculty of Management and RUTCOR
Newark, NJ 07102 and New Brunswick, NJ 08903
kogan@rutcor.rutgers.edu

Abstract

Most learning algorithms work most effectively when their training data contain *completely specified* labeled samples. In many diagnostic tasks, however, the data will include the values of only some of the attributes; we model this as a *blocking* process that hides the values of those attributes from the learner. While blockers that remove the values of critical attributes can handicap a learner, this paper instead focuses on blockers that remove only *conditionally irrelevant* attribute values, *i.e.*, values that are *not needed to classify an instance*, given the values of the other unblocked attributes. We first motivate and formalize this model of "superfluous-value blocking," and then demonstrate that these omissions can be useful, by proving that certain classes that seem hard to learn in the general PAC model — *viz.*, decision trees and DNF formulae — are trivial to learn in this setting. We then extend this model to deal with (1) theory revision (*i.e.*, modifying an existing formula); (2) blockers that occasionally include superfluous values or exclude required values; and (3) other corruptions of the training data.

Keywords: irrelevant values, blocked attributes, learnability, decision trees, DNF, diagnosis, theory revision, adversarial noise

[*]This is an extended version of the short conference paper [GGK96]. We gratefully acknowledge receiving helpful comments from R. Bharat Rao, Tom Hancock, Dan Roth, Dale Schuurmans and George Drastal, and from the anonymous reviewers. The third author gratefully acknowledges the partial support of ONR (Grants N00014-92-J1375 and N00014-92-J4083).

1 Introduction

A diagnostician typically performs only a small fraction of all possible tests; furthermore, the choice of which tests to perform depends on the results of the tests performed earlier in the diagnosis session. As an example: Knowing that a certain positive blood test x_1 is sufficient to establish that a patient has `diseaseX`, a doctor can conclude that a patient has `diseaseX` after performing only test x_1, if that test result is positive. In recording his findings, the doctor will only record the result of this one test $\langle x_1, 1\rangle$ and the diagnosis `diseaseX`. *N.b.*, the doctor does not know, and therefore will not record, whether the patient has symptoms corresponding to tests $x_2, x_3, \ldots, x_n$.

A learner (a medical student, perhaps) may later examine the doctor's files, trying to learn the doctor's diagnostic procedure. These records are quite "incomplete", in that the values of many attributes are missing; *e.g.*, they do *not* include the results of tests x_2 through x_n on this patient. However, within this model, the learner can use the fact that these attributes are missing to conclude that *the missing tests are not required to reach a diagnosis*, given the known values of the other tests. Hence, these omissions reflect the fact that the doctor's classifier (which the learner is trying to learn) can establish `diseaseX` and terminate on observing only that x_1 is positive.

This paper addresses the task of learning in this context: when each training sample specifies the values for only a subset of the attributes, together with that sample's correct class, with the understanding that the supplied values are (a minimal set that is) sufficient to classify this sample. Of course, this framework requires that a helpful "teacher" (*e.g.*, the doctor mentioned above) specify the appropriate values for all-and-only the "relevant" attributes. We will see that this relevance information can be *extremely* useful, as it greatly simplifies the learner's task of correctly identifying the teacher's classifier.

Having such a helpful teacher is a very strong requirement, but there are situations (such as this medical example) where it seems fairly plausible. Furthermore, in later sections of this paper we weaken the basic assumption of a very helpful teacher by allowing for various "noise" processes, and show that many of the positive results continue to hold. Note also that missing values are ubiquitious in the "real world" data, and irrelevance is often one of the factors causing these omissions. Given our results, which demonstrate the huge *potential* gains possible for learning systems that can exploit such "meaningful" omissions, it seems unrealistic to ignore this possible cause, and in effect assume that missing data can only be harmful. We hope that our initial results will inspire research into yet more realistic and useful models.

After Section 2 presents our formal model, Section 3 shows how easy it is to learn first decision trees, and then arbitrary DNF formulae, within this framework. By contrast, neither of these two well-studied classes is known to be learnable using completely-specified training samples. Section 4 then presents a variety of extensions to this basic model, to make it both more general and more robust, dealing with: (1) the theory revision task (*i.e.*, modifying an existing formula); (2) degradations in the training data (*i.e.*, data which occasionally includes superfluous values or excludes required values), and (3) other corruption processes, such as classification noise and attribute noise. The Appendix presents the proofs of all theorems. We close this section by further motivating our framework and describing how it differs from related work on learning from incomplete data.

Motivation and Related Work: Most implemented learning systems tend to work effectively when very few features are missing, and when these missing features are randomly distributed across the samples. However, recent studies [PBH90, RCJ88] have shown that many real-world datasets are missing more than half of the feature values! Moreover, these values are not randomly blocked, but in fact "*are missing [blocked] when they are known to be irrelevant for classification or redundant with features already present in the case description*" [PBH90], which is essentially the situation considered in this paper (see Definition 1). Towards explaining this empirical observation, note that a diagnosis often corresponds to a single path through an n-node decision tree and so may require only a small number of tests; the remaining test values are simply irrelevant. Our model of learning can, therefore, be applicable to many diagnostic tasks, and will be especially useful where the experts are unavailable or are unable to articulate the classification process they are using.

Turney [Tur95] discusses a model that also assumes that experts intentionally perform only a subset of the possible tests. His model allows the system to use test-cost to decide which tests to omit. By contrast, in our model, the environment/teacher uses test-relevance to decide which tests to present.

While there are several learning systems that can handle incomplete information in the samples (*cf.*, [BFOS84, Qui92, LR87]), they all appear to be based on a different model [SG97, SG94]: after the world produces a completely-specified sample at random, a second "blocking" process (which also could be "nature") hides the values of certain attributes *at random*. Here, no useful information is conveyed by the fact that an attribute is hidden in a particular example.

Although the model in this paper also assumes that a random process is generating complete tuples which are then partially blocked, our model differs by dealing with blocking processes that (try to) block only "irrelevant" values; *i.e.*, attributes whose values do not affect the instance's classification. More specifically, in our model the blocked values are *superfluous* or *conditionally irrelevant*; that is, *given* the unblocked values we see, the blocked values are not able to affect the classification. For example, if the doctor would conclude `diseaseX` when x_1 is positive, whether x_2 is positive or negative, then "x_2 is superfluous given that x_1 is positive". Of course, if x_1 is negative, other tests may then be relevant for the diagnosis; perhaps a negative x_2 and a positive x_3 will be sufficient to establish `diseaseX`, etc. John *et al.* [JKP94] would therefore consider x_2 to be "weakly irrelevant"; by contrast, they say an attribute is "strongly irrelevant" if its value *never* plays a role in the classification, under any circumstance (*i.e.*, independent of the values of any other attributes); *cf.*, [Lit88, Blu92, BHL95]. Our situation differs from these models, as we assume that the environment explicitly identifies weakly irrelevant (*i.e.*, superfluous) attributes.

Similarly, Russell and others [Rus89, MT94] say a set of attributes X "determine" another attribute y if any assignment to members of X is sufficient to specify the value for y; *i.e.*, all other (non-X) attributes are irrelevant. Our model, however, allows y's value to be "determined" by different sets of attributes in different situations.

Our final comments help to place our model within the framework of existing computational learning results: First, in our model, certain *attribute* values are *omitted*; this differs from the problem of unsupervised learning, in which the *class* label is omitted [SD90, Chap-

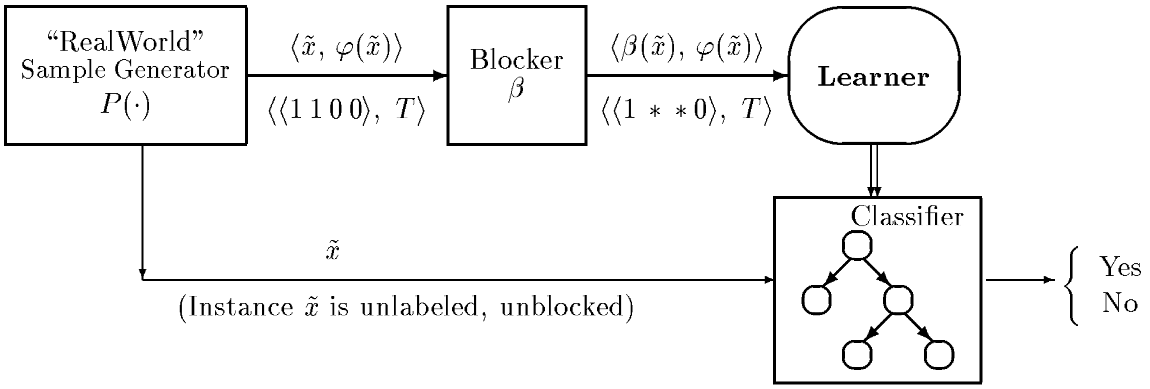
"RealWorld" Sample Generator
$P(\cdot)$
$\langle \tilde{x}, \varphi(\tilde{x}) \rangle$
$\langle \langle 1\,1\,0\,0 \rangle, T \rangle$
Blocker
β
$\langle \beta(\tilde{x}), \varphi(\tilde{x}) \rangle$
$\langle \langle 1 * * 0 \rangle, T \rangle$
Learner
Classifier
$\tilde{x}$
(Instance $\tilde{x}$ is unlabeled, unblocked)
Yes
No

Figure 1: Blocking Model

ter 3], and from problems in which some attribute values are *changed* [SV88, Lit91, GS95]. Second, as our blocker is providing additional information to the learner, its role is similar to that of a benevolent teacher. However, other teaching models, such as Goldman and Mathias [GM96], allow the teacher to present arbitrary instances to the learner, without regard to an underlying real-world distribution. By contrast, our blocker/teacher is forced to deal with the instances selected by the distribution, but can help the learner by declaring certain attribute values, within those instances, to be conditionally irrelevant.

2 Framework

Following standard practice, we identify each domain instance with a finite vector of boolean attributes $\tilde{x} = \langle x_1, \ldots, x_n \rangle$. Let $X_n = \{0,1\}^n$ be the set of all possible domain instances. The learner is trying to learn a concept φ, which we view as an indicator function $\varphi\colon X_n \mapsto \{T, F\}$, where $\tilde{x}$ is a member of φ iff $\varphi(\tilde{x}) = T$. We assume the learner knows the set of possible concepts, $\mathcal{C}$.[1] A "(labeled) example of a concept $\varphi \in \mathcal{C}$" is a pair $\langle \tilde{x}, \varphi(\tilde{x}) \rangle \in X_n \times \{T, F\}$. We assume there is a stationary distribution $P : X_n \mapsto [0,1]$ over the space of domain instances, from which random labeled instances are drawn independently, both during training and testing of the learning algorithm.

To continue the earlier example, suppose the first attribute x_1 in the instance $\tilde{x} = \langle x_1, \ldots, x_4 \rangle$ corresponds to the blood test and the subsequent attributes x_2, x_3 and x_4 correspond (respectively) to particular tests of the patient's bile, melancholy and phlegm. Then the instance $\langle 0,1,1,1 \rangle$ corresponds to a patient whose blood test was negative, but whose bile, melancholy and phlegm tests (x_2, x_3 and x_4) were all positive. Assume that the concept associated with `diseaseX` corresponds to any tuple $\langle x_1, \ldots, x_4 \rangle$ where either $x_1 = 1$ or both $x_2 = 0$ and $x_3 = 1$. Hence labeled examples of the concept `diseaseX` include $\langle \langle 1,0,1,1 \rangle, T \rangle$, $\langle \langle 1,0,0,0 \rangle, T \rangle$, $\langle \langle 0,0,1,1 \rangle, T \rangle$, and $\langle \langle 0,1,0,0 \rangle, F \rangle$. Further, $P(\tilde{x})$ specifies the probability of encountering a patient with the particular set of symptoms specified by

[1]To simplify our presentation, we will assume that each attribute has one of only two distinct values, $\{0,1\}$, and that there are only two distinct classes, written $\{T, F\}$. It is trivial to extend this analysis to consider a larger (finite) range of possible attribute values, and larger (finite) set of classes.

$\tilde{x}$; *e.g.*, $P(\langle 1,0,1,0\rangle) = 0.01$ means 1% of the time we will deal with a patient with positive blood and melancholy tests, but negative bile and phlegm tests.

In general, a *learning algorithm* L has access to a source of labeled examples $\langle \tilde{x}, \varphi(\tilde{x})\rangle$, drawn randomly and independently according to the distribution P and labeled by the target (unknown) concept $\varphi \in \mathcal{C}$. (When we consider the computational complexity of a learning algorithm L, we can assume that L takes constant time to draw each new labeled example.) L's output is a hypothesis $h : X_n \to \{T, F\}$. In many cases, one does not require that $h \in \mathcal{C}$; it usually suffices that h be evaluable in polynomial time. We discuss below how this model relates to our model of blocking, and then discuss how we evaluate L.

Model of "Blocked Learning": In standard learning models, the learning algorithm gets to see each randomly-drawn labeled instance $\langle \tilde{x}, \varphi(\tilde{x})\rangle$ "as is". In this paper we also consider learning algorithms that only get to see a "blocked version" of $\langle \tilde{x}, \varphi(\tilde{x})\rangle$, written "$\langle \beta(\tilde{x}), \varphi(\tilde{x})\rangle$". A blocker $\beta\colon X_n \to \{0,1,*\}^n$ replaces certain attribute values by the "blocked" (or in our case, "don't care") token "$*$", but otherwise leaves $\tilde{x}$ and the label $\varphi(\tilde{x})$ intact; see Figure 1.[2] Hence, a blocker could map $\langle\langle 1,1,0,1\rangle, T\rangle$ to $\langle \beta(\langle 1,1,0,1\rangle), T\rangle = \langle\langle 1,*,*,*\rangle, T\rangle$, or $\langle \beta(\langle 0,1,0,1\rangle), F\rangle = \langle\langle *,1,0,*\rangle, F\rangle$; but no such blocker can map $\langle\langle 1,1,0,1\rangle, T\rangle$ to any of $\langle\langle 1,0,0,1\rangle, T\rangle$, $\langle\langle 1,1,1,*\rangle, T\rangle$, or $\langle\langle 1,1,0,1\rangle, *\rangle$. Let $X_n^* = \{0,1,*\}^n$ denote the set of possible instance *descriptions*.

This paper considers *superfluous-value blockers*: *i.e.*, blockers that only block attribute values that do not affect an instance's classification, given the values of the other unblocked attribute values. To state this more precisely:

Definition 1 ("Superfluous") *Let $\varphi \in \mathcal{C}$ be a concept over attributes $\{x_1, \ldots, x_n\}$. Then:*

(1) A subset of attributes, say $\{x_{m+1}, \ldots, x_n\}$, is superfluous *given a particular assignment to the remaining values $\{\, x_1 \mapsto v_1,\ \ldots,\ x_m \mapsto v_m \,\}$ iff, for any assignment $x_{m+1} \mapsto v_{m+1} \ldots, x_n \mapsto v_n$:*

$$\varphi(v_1, \ldots, v_m, \ldots, v_n) \;=\; \varphi(v_1, \ldots, v_m, 0, \ldots, 0)$$

That is, the values given to the superfluous variables do not affect the classification.

(2) A function $\beta\colon X_n \mapsto X_n^*$ is a superfluous value blocker *if it only blocks superfluous attributes; i.e., whenever $\beta(\langle v_1, \ldots, v_n\rangle) = \langle v_1, \ldots, v_m, *, \ldots, *\rangle$ then the attributes $\{x_{m+1}, \ldots, x_n\}$ are superfluous given the partial assignment $\{\, x_1 \mapsto v_1,\ \ldots, x_m \mapsto v_m \,\}$.*

For example, $\beta(\langle 1,0,1,1\rangle) = \langle *,0,1,*\rangle$ is allowed *only if* all four instances $\langle 0,0,1,0\rangle$, $\langle 0,0,1,1\rangle$, $\langle 1,0,1,0\rangle$, and $\langle 1,0,1,1\rangle$ have the same classification.

Note that there is a difference between being superfluous and being *redundant.* If an attribute is redundant — meaning that its value is determined once certain other attributes' values are specified — then it is also superfluous given these other values. Note, however, that the converse is not necessarily true.

[2]This is a slight abuse of notation, as β may be stochastic; see the discussion in Section 3.2. This caveat also applies to Definition 1.

While Definition 1 allows "blockers" that never blocks any attribute values, we will see later that the interesting results in this paper apply only to cases in which a large subset of the possible superfluous attributes are blocked.

To motivate our model, consider the behavior of a classifier d_t using a standard decision tree, *à la* CART [BFOS84] or C4.5 [Qui92]. Here, given any instance, d_t will perform (and record) only the tests on a single path through the tree. The other variables, corresponding to tests that do not label nodes on this path, do not matter: d_t will reach the same conclusion no matter how we adjust their values. Similar claims hold for many other classification structures, including decision lists and the rule sets produced by C4.5. Section 3.2 extends this idea to general DNF formulae.

Performance Criterion: To specify how we will evaluate the learner, we first define the error of the hypothesis h returned by the learner (for a given set of samples drawn from distribution P over X^n and labeled according to concept φ) as $Err(h) = P(\tilde{x} : \varphi(\tilde{x}) \neq h(\tilde{x}))$; *i.e.*, the probability that h will misclassify an instance $\tilde{x}$ drawn from P.

We use the standard "Probably Approximately Correct" (PAC) criterion [Val84, KLPV87] to specify the desired performance of our learners.

Definition 2 (PAC-learning) *A learning algorithm* L PAC-learns *a set of concepts* $\mathcal{C}$ *if, for some polynomial function* $p(\cdots)$*, for all target concepts* $\varphi \in \mathcal{C}$*, distributions* P *over* X_n*, and error parameters* $\epsilon, \delta > 0$*,* L *runs in time at most* $p(\frac{1}{\epsilon}, \frac{1}{\delta}, |\varphi|)$*, and outputs a hypothesis* h *whose error is, with probability at least* $1-\delta$*, less than* ϵ*; i.e.,*

$$\forall P \in \text{"distributions on } X_n\text{"},\ \varphi \in \mathcal{C},\ \epsilon, \delta > 0, \quad P(\ \mathrm{Err}(h) < \epsilon\ ) \quad \geq \quad 1-\delta\ .$$

In this definition, $|\varphi|$ is the "size" of the concept φ (defined below). To understand the condition involving "$P(Err(h) < \epsilon)$", recall that a learning algorithm, by definition, has access to labeled training examples drawn randomly according to P. Thus the output, h, of L is typically probabilistic, with a distribution induced from P because it depends on the particular training examples that were seen. So by allowing L to return a "bad" hypothesis with probability δ we allow for cases in which L happens to see an unrepresentative training sample. Next, note that the number of instances seen by L can be at most L's running time, and thus is also polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$ and $|\varphi|$. Finally, notice that the learner is expected to acquire a classifier that has high accuracy on *completely specified* instances, even if it was trained on blocked values. This is reasonable, as we are assuming that, after learning, the classifier (read "doctor") is in a position to specify the tests to be performed. (Of course, such a complete-value classifier can trivially classify any instance whose superfluous values were removed: As these attributes are irrelevant, the classifier can simply replace each omitted attribute with, say, 0.)

We later consider a stronger learning model, called *mistake-bound* learning [Lit88], which is used to evaluate learning algorithms that are used in an *on-line* fashion. Here, the learner successively draws random *unlabeled* samples and must guess the correct label; it is then told the right label. If we can bound the total number of incorrect guesses made, over any sequence of drawn examples, then the algorithm is said to exhibit an (absolute) mistake bound. We will provide mistake bounds for some of our algorithms. Note however that this on-line model seems rather unnatural in our setting, as it seems unlikely that we would get the relevance information for each instance, but not its class label. (*E.g.*, in our motivating example of a medical student examining doctor's records, why would the doctor, who is blocking each instance appropriately, not supply the label?)

Notation: The concept classes of most interest to us are decision trees and disjunctive normal form (DNF) formulae.[3] We use $\mathcal{DT}_{n,s}$ to denote the set of decision trees defined over the n boolean variables $x_1, \ldots, x_n$, with at most s leaf nodes;[4] further, $\mathcal{DT}_n = \bigcup_s \mathcal{DT}_{n,s}$ is the set of all decision trees over $x_1, \ldots, x_n$. For any decision tree $d \in \mathcal{DT}_n$, we let $|d|$ be the number of leaf nodes in d. Let $\mathcal{DNF}_{n,s}$ be the set of DNF formulae, over the n boolean variables $x_1, \ldots, x_n$, with at most s terms, and let $\mathcal{DNF}_n = \bigcup_s \mathcal{DNF}_{n,s}$ be the set of all DNF formulae over $x_1, \ldots, x_n$. For any DNF formula $\varphi \in \mathcal{DNF}_n$, $|\varphi|$ is the number of terms in φ. Finally, for constant k, k-$\mathcal{DNF}_n$ is the class of DNF formulae whose terms include at most k literals. Similarly, a formula in $(c \log n)$-$\mathcal{DNF}_n$ is a disjunction of terms, each of which has at most $c \log n$ literals.

3 Learning Decision Trees and DNFs: Simple Cases

This section discusses the challenge of learning two standard classes of concepts within our "superfluous blocking model", decision trees (Subsection 3.1) and DNF formulae (Subsection 3.2). Section 4 later presents several extensions and variations of these situations.

3.1 Learning Decision Trees

In this section, we suppose that a learning algorithm is given training samples that are both labeled and blocked using a target decision tree d_t; the learner's task, then, is to recover d_t. To be specific, we assume that the blocking process β_{d_t} does the following: Given a complete instance $\vec{x} = \langle x_1, \ldots, x_n \rangle$, β_{d_t} traverses the decision tree d_t from the root down, recording all (and only!) the tests that were performed in this traversal, as well as the final classification from $\{T, F\}$. Thus, the reported tests correspond exactly to some path through d_t. All attributes not on the path are blocked, and so are reported as "$*$". We consider below learning decision trees under this particular blocking model, which we call the "$B^{(DT)}$ learning model".

For example, imagine that a doctor was using the d_0 decision tree shown in Figure 2, in which he descends to a node's right child if the node's test is positive, and goes to the left child otherwise. Given the complete instance $\langle x_1, x_2, x_3, x_4, x_5, x_6 \rangle = \langle 0, 1, 1, 0, 0, 0 \rangle$, the doctor (using d_0) will first perform test x_1 and as it fails, descend to the left, to the node labeled x_2. As this x_2 test succeeds, d_0 reaches a leaf node, labeled F. Here, the learner will see

$$\langle \beta_{d_0}(\langle 0,1,1,0,0,0,0 \rangle),\ d_0(\langle 0,1,1,0,0,0,0 \rangle) \rangle \ = \ \langle \langle 0,1,*,*,*,* \rangle,\ F \rangle.$$

[3]A DNF formula is a disjunction of terms, where each term is a conjunction of literals, where each literal is either positive (x) or negative ($\bar{x}$); e.g., $\varphi \ \equiv \ (x_1 \bar{x_2}) \vee (x_2 x_4 \bar{x_5}) \vee (x_9)$ is a DNF formula.

[4]Throughout this paper, we restrict attention to decision trees in which each internal node tests a single variable, branching according to whether the variable has value 0 or 1.

$d_0 \quad =$

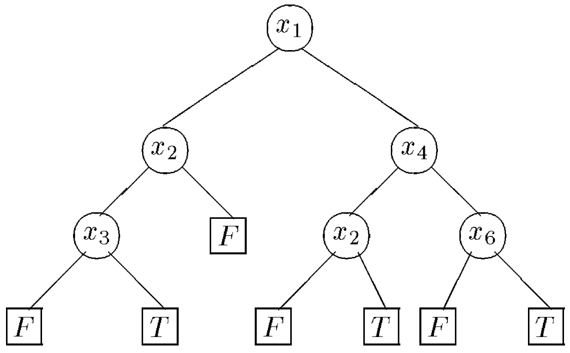

Figure 2: Decision Tree, d_0

Alternatively, given the instance $\langle 1,1,0,0,1,0\rangle$, the learner will see $\langle \beta_{d_0}(\langle 1,1,0,0,1,0\rangle), d_0(\langle 1,1,0,0,1,0\rangle)\rangle = \langle\langle 1,1,*,0,*,*\rangle, T\rangle$. Note that two trees can be logically equivalent (*i.e.*, encode the same classification function), but produce different blocking behavior.

There is currently no known algorithm capable of learning decision trees in the standard PAC model.[5] There is, however, a simple algorithm that can learn decision trees from these blocked instances in the $B^{(DT)}$ model. If the target decision tree is believed to consist of no more than s leaves, the LEARN-DT(n, s, ϵ, δ) algorithm first draws a sample of

$$m_{DT:n,s} \quad = \quad \frac{1}{\epsilon}\left(s\ln(8n) + \ln\frac{1}{\delta}\right) \tag{1}$$

random (blocked and labeled) training examples, and then calls the BUILD-DT algorithm (shown in Figure 3), which builds a decision tree that correctly classifies this sample. BUILD-DT first selects as the root any attribute value that is never blocked, then splits on this attribute, calling itself recursively. To understand why BUILD-DT works, observe that (*i*) the root of the target tree can never be blocked and (*ii*) any variable that is never blocked appears on the path to every leaf reached by the sample and hence can be placed at the root without penalty.

Theorem 1[6] LEARN-DT(n, s, ϵ, δ) *PAC-learns decision trees in* $\mathcal{DT}_{n,s}$ *under the* $B^{(DT)}$ *blocking model. That is, for any* $d \in \mathcal{DT}_{n,s}$*, any values of* $\epsilon, \delta \in (0,1)$ *and any distribution over instances* P*,* LEARN-DT(n, s, ϵ, δ) *will return a tree* $d' \in \mathcal{DT}_{n,s}$ *whose error (on unblocked, unlabeled instances) is, with probability at least* $1-\delta$*, at most* ϵ*. Moreover,* LEARN-DT *requires* $O(\frac{s}{\epsilon}\ln\frac{n}{\delta})$ *blocked labeled samples and returns a tree of size* $|d'| \leq |d|$*. When used in an on-line fashion,* LEARN-DT *exhibits a mistake bound of* s*.*

While this algorithm is parameterized by the size of the tree s, it is possible to use the standard technique of repeated attempts, with successively doubled estimates of s [HKLW91], to produce an algorithm that does not have to know s in advance. Here, the learning algorithm LEARN-DT$'$ first draws a set of samples based on the assumption that $s = 1$,

[5]The most general known algorithms run in pseudo-polynomial time, *i.e.*, they learn an s-leaf decision tree in time polynomial in $s^{O(\log s)}$ [EH89, Riv87].

[6]All proofs are in the Appendix.

```
Algorithm Build-DT( S: set_of_labeled_blocked_samples ): DT_Type
          /* Builds a tree using labeled blocked samples S */
  Let NT be a new tree
  If (S is empty) Then
    NT.LeafLabel = $F$
    Return( NT )
  If (all samples in S are labeled with same label, $\ell$) Then
    NT.LeafLabel = $\ell$
    Return( NT )
  Let $x_i$ be any variable that is unblocked for every sample in $S$.
  Let $S^0 = \{ \langle \tilde{x}_{-i}, \ell \rangle \} \mid \langle \tilde{x}, \ell \rangle \in S, \tilde{x}_i = 0 \}$
      $S^1 = \{ \langle \tilde{x}_{-i}, \ell \rangle \} \mid \langle \tilde{x}, \ell \rangle \in S, \tilde{x}_i = 1 \}$
        /* I.e. to form $S^0$: Assemble the instances in S such that $x_i = 0$; and project out $x_i$.
           (Here, $\tilde{x}_{-i}$ denotes the tuple $\tilde{x}$ with the i'th component removed.)
           $S^1$ is constructed analogously, from those instances with $x_i = 1$. */
  Let NT.InternalNodeLabel = $x_i$
      NT.If0 = Build-DT( $S^0$ )
      NT.If1 = Build-DT( $S^1$ )
  Return( NT )
End Build-DT
```

Figure 3: Build-DT Algorithm for learning decision trees, in $B^{(DT)}$ Model

then calls Build-DT on this set. The resulting tree is accepted if it has 1 leaf; otherwise, Learn-DT′ draws a set of samples based on $s = 2$, calls Build-DT, and accepts the tree if it has at most 2 leaves. If not, it successively tries $s = 4$, $s = 8$, $s = 16$, ..., until it succeeds. An argument similar to the one given in the proof of Theorem 1 shows that (with high probability) Learn-DT′ will succeed once s is large enough.[7]

3.2 Learning DNF Formulae

This subsection considers learning arbitrary DNF formulae (a class that significantly generalizes decision trees[8]) when superfluous values are omitted. Here, each blocked positive instance is simply an implicant of the target formula φ, and each blocked negative instance is an implicant of the target's negation $\neg\varphi$. However, while decision trees have an obvious "evaluation procedure" which describes the particular implicant to use, there are many different ways of specifying which implicant should be returned when considering DNF formulae. For now, we focus on the model that most closely resembles the $B^{(DT)}$ model for decision trees.

Blocking Model: Given any decision tree t, a logically-equivalent DNF formula φ_t contains, as terms, the "conditions" of the paths (from the root) to each T-labeled leaf node n_ℓ, where

[7]Actually, Learn-DT′ will have to request slightly more than Equation 1's $m_{DT:n,s}$ samples when considering each value of s, as it has to consider the possibility of making a mistake on any of the $\log(2^n) = n$ values of s; see [HKLW91].

[8]*I.e.*, every poly-sized decision tree is logically equivalent to a poly-sized DNF formula, but not *vice versa*.

the "condition" of the path $\langle n_1, \ldots, n_\ell \rangle$ is the conjunction of the variables of the nodes n_i, whose sign (either x_i or $\bar{x}_i$) is determined by whether n_{i+1} is n_i's $+$-child or $-$-child. For example, a DNF formula corresponding to Figure 2's d_0 is

$$\varphi_{d_0} \quad \equiv \quad \bar{x}_1 \bar{x}_2 x_3 \ \vee \ x_1 \bar{x}_4 x_2 \ \vee \ x_1 x_4 x_6 \ .$$

Notice that, given any positive instance, the β_{d_0} blocker would leave unblocked the variables of exactly one of these terms.

In general, given an arbitrary target DNF formula $\varphi = t_1 \vee \cdots \vee t_s$, we define a β_φ blocker as any blocker that acts in the following way:

- Given a positive instance $\langle \tilde{x}, T \rangle$, β_φ leaves unblocked exactly the variables of one of the terms in φ that $\tilde{x}$ satisfies (*i.e.*, $\beta_\varphi(\tilde{x})$ returns some t_j such that t_j evaluates to *true* under $\tilde{x}$).
- Given a negative instance $\langle \tilde{x}, F \rangle$, $\beta_\varphi(\tilde{x})$ is an implicant of $\neg\varphi$ constructed from $\tilde{x}$. That is, for each t_i, there is at least one unblocked variable from $\tilde{x}$ in $\beta_\varphi(\tilde{x})$ that appears with the opposite sign in t_i.

We let the term "$B^{(DNF)}$ learning model" refer to learning DNF concepts φ, under β_φ blocking. Although the notation suggests that β_φ is functional, there might be several distinct terms t_i that the blocker could return for a given positive instance $\tilde{x}$, whenever $\tilde{x}$ is implied by several terms in φ. None of our later results change if the blocker chooses among the possibilities stochastically, so long as the choice process is stationary for positive instances. For negative instances we do not even require stationarity.

Of course, many blockers are deterministic. For example, the natural blocker for a decision tree (which is a special case of DNF formula) deterministically returns the conditions of the unique path traversed evaluating an instance. It is easy to extend this idea to define a deterministic blocker for any DNF formula. For instance, consider a blocker that imposes a specific order on the terms in the given DNF formula, and also an order on the literals within each term. Then given a positive instance $\tilde{x}$, this blocker examines the DNF terms in the given order until finding one that is satisfied, and then returns this term. For each negative example $\tilde{x}$, this blocker collects literals by walking through the terms, and for each term, including the first variable that occurs with opposite sign to its appearance in $\tilde{x}$. This particular blocker has the property that, given any s-term DNF formula, it will leave unblocked at most s literals for each negative example. We will see that our results hold even if the blocker returns "too many" unblocked literals on *negative* examples.

Learning DNF Formulae, under Blocking: There is a trivial algorithm, called "LEARN-DNF", that can PAC-learn s-term DNF formulae in this $B^{(DNF)}$ model. LEARN-DNF simply requests

$$m_{DNF:n,s} \quad = \quad \frac{4s}{\epsilon} \ln \frac{s}{\delta}$$

samples, then forms a DNF formula by disjoining the observed positive samples.[9] Hence:

[9]Notice that this approach resembles simple "table learning", and so can be considered related to case-based and nearest-neighbor algorithms. It differs, of course, in that each of the training examples (a.k.a. "cases", "neighbors") is given to us in an appropriately generalized form, which means that we can use simple subsumption to decide whether each such training example "covers" a test instance.

Theorem 2 LEARN-DNF(n, s, ϵ, δ) *PAC-learns $\mathcal{DNF}_{n,s}$ under the $B^{(DNF)}$ blocking model. That is, for any $\varphi \in \mathcal{DNF}_{n,s}$, any values of $\epsilon, \delta \in (0,1)$ and any distribution over instances P,* LEARN-DNF(n, s, ϵ, δ) *will return a DNF formula $\varphi' \in \mathcal{DNF}_{n,s}$ whose error (on unblocked, unlabeled instances) is, with probability at least $1-\delta$, at most ϵ. Moreover,* LEARN-DNF *requires $O(\frac{s}{\epsilon}\ln\frac{s}{\delta})$ blocked labeled samples, and will return a DNF formula with at most $|\varphi|$ terms. Notice also* LEARN-DNF *uses only* positive *samples. When used in an on-line fashion,* LEARN-DNF *exhibits a mistake bound of s.*

By contrast, learning arbitrary DNF formulae in the standard model is one of the major open challenges of PAC-learning in general [Ang92].

Many learnability results are expressed in terms of the size of the *smallest* DNF formula for a concept. However, our result deals explicitly with the specific formula that the environment (a.k.a. "teacher") is using, and hence our results are of the form "the computational cost is polynomial in the size of the formula considered". Unfortunately, this formula could be exponentially larger than the smallest equivalent DNF formula. Also, while LEARN-DNF (and Theorem 2) require a bound s on the size of the target formula, we can avoid this by using the already-mentioned technique of successively doubling estimates of s.

Other blocking models: We close this section by noting that $B^{(DNF)}$ is not the only natural blocking model that might be considered for DNF formulae. We earlier mentioned a particular type of $B^{(DNF)}$ blocker that examines the terms of a DNF formula one-by-one in some fixed order, stopping when it finds a term that satisfies the given instance. For example, such a blocker for the concept $\varphi = \bar{x}_1 x_2 \vee x_5 \bar{x}_2 \vee x_3 x_4 \vee x_3 \bar{x}_5 x_6$, given the instance $x_1 x_2 x_3 x_4 x_5 x_6$, would return $x_3 x_4$. Imagine, however, a doctor was actually evaluating this formula by considering its terms in this order. To evaluate $\bar{x}_1 x_2$, he would test x_1, and finding that $x_1 = 1$, reject the first term. On the second term, he would first test x_5, and as $x_5 = 1$, proceed to examine x_2. As $x_2 = 1$, he would then reject the second term and reach the third term. On confirming that $x_3 = 1$ and $x_4 = 1$, he would return T. As the doctor has now performed the tests $\{x_1, x_2, x_3, x_4, x_5\}$, it may make sense for him to record all of this information, even though only a subset was actually necessary for the final classification. In contrast, our $B^{(DNF)}$ blocks more: here, it would return just $x_3 = 1$ and $x_4 = 1$, as these reasons (attributes) are sufficient to support the positive classification. Hence, our model is appropriate if the doctor is required only to provide the evidence that *justifies* his decision.

Some apparently different blocking models actually fit within our framework. For example, imagine there are several different "teachers", each with his own (syntactically) distinct, but logically equivalent, DNF formula, and each example is blocked according to one of these formulae (perhaps chosen at random). While this may appear to be a different notion of blocking, we can treat it as an ordinary instance of $B^{(DNF)}$ blocking, albeit with a formula formed by disjoining the experts' individual formulae. While the learner may produce a formula that is unnecessarily long (because it is learning from redundant sources), no other special treatment is required. We can similarly use this technique to handle multiple decision trees, although we will learn a DNF formula rather than a decision tree.

Another case where the standard $B^{(DNF)}$ blocker turns out to be sufficient occurs with certain blockers that block even *more* than the $B^{(DNF)}$ model does. Consider a blocker that presents a *minimal* amount of information needed to determine the label; *i.e.*, by presenting the learner with just *prime* implicants. (An implicant is prime if no proper subset of its literals is an implicant.) Such a blocker might first select a suitable term from the target DNF formula φ, and then remove from the term some literals that are not essential for the classification, and finally returning the resulting prime implicant. For example, given the target formula $\varphi = x_1\bar{x}_2 \vee x_1x_2$, and instance $\tilde{x} = \langle 1, 0\rangle$, such a blocker could just return $\langle 1, *\rangle$ because the value of x_2 is, in fact, irrelevant here. Notice that a term in φ may be subsumed by many different prime implicants — in fact, exponentially many.[10] Even if the blocker gets to choose among these prime implicants, possibly returning a different prime implicant every time an instance satisfies a particular term, the learning task remains easy. In fact, exactly the same LEARN-DNF algorithm and the same Theorem 2 bound remain valid, as the formula produced in this case is subsumed by φ, and it will clearly subsume the formula produced using the standard β_φ blocker, for the same set of training samples.

4 Extensions: Theory Revision, Degradation, Noise

This section presents a variety of extensions to our basic "superfluous value" model, to make it both more general and more robust. Subsection 4.1 first considers the situation where the learner begins with an initial classifier, which it modifies in light of new examples. The other subsections discuss ways to make our model more robust to "teacher error". Subsection 4.2 models "degraded blockers" that occasionally exclude certain required attribute values, or include some superfluous values, and then provides algorithms that can cope with certain ranges of such corruptions; Subsection 4.3 similarly analyzes classification and attribute noise within our framework.

4.1 Theory Revision: Improving a given Initial Classifier

In many situations, we may already have an initial theory φ_{init} (which may be either a decision tree or a DNF formula) that is considered quite accurate, but not perfect. This subsection describes ways of using a set of labeled samples to improve φ_{init}; *i.e.*, to form a new theory φ_{better} that is similar to φ_{init}, but is (with high probability) more accurate. We let $B_{TR}^{(DNF)}$ refer to this model (for DNF formulae) where the TR designates "<u>T</u>heory <u>R</u>evision", corresponding to the many existing systems that perform essentially the same task, albeit in the framework of Horn-clause based reasoning systems; *cf.*, [Tow91, WP93, MB88, OM90, LDRG94]. After this, we consider theory revision for *decision trees*, $B_{TR}^{(DT)}$.

There are several obvious advantages to theory revision over the "grow from scratch" approach discussed in the previous section. First, notice from Theorem 2 that the number of samples required to build a DNF formula is proportional to the size of the final formula, which can be exponential in the number of attributes. So, given only a small number of labeled samples, we may be unable to reliably produce an adequate theory, much less the

[10]For example, consider $\varphi = \bigvee_{i=1}^{n}(x_i\bar{y}_i \vee \bar{x}_iy_i) \vee (\bigwedge_{i=1}^{n} x_iy_i)$, and notice that the final term is subsumed by any of the 2^n prime implicants of the form $\bigwedge_{i=1}^{n}\{x_i \text{ or } y_i\}$, where each "$\{x_i \text{ or } y_i\}$" denotes a single literal which is either x_i xor y_i.

```
Algorithm MODIFY-DNF( $\varphi_{init}$ : DNF_Type; $n, r$ : $\mathcal{N}$; $\alpha, \epsilon, \delta \in (0,1)$ ): DNF_Type
  Draw $m_r = \frac{4r}{\alpha\epsilon}\ln\frac{2r}{\delta}$ training (blocked, labeled) samples
    /* Alternatively, if $\alpha$ is not known a priori, we can instead collect $\frac{4r}{\epsilon}\ln\frac{2r}{\delta}$ samples
       that $\varphi_{init}$ does not label correctly (i.e., either mislabels, or is unable to label[11]) */
  Let $S^+$ = positive samples that $\varphi_{init}$ either can't classify or classifies as F;
      $S^-$ = negative samples that $\varphi_{init}$ either can't classify or classifies as T.
  Let $T = S^+ \cup \varphi_{init}$.
    /* I.e., $T$ includes both un- and mis-classified positive examples, and the terms from $\varphi_{init}$. */
  Remove from $T$ any term that is consistent with some term in $S^-$.
    /* E.g., the negative example $x_1\bar{x}_2 \in S^-$ will remove from $T$ the terms $x_1$ and $\bar{x}_3x_5$,
       but not $\bar{x}_1x_8$. */
  Return disjunction of terms in $T$.
End MODIFY-DNF
```

Figure 4: The MODIFY-DNF algorithm for modifying an initial DNF formula

optimal one. We show below that this same set of samples may, however, be sufficient to specify how to improve a given initial theory.

Stated more precisely, we assume our initial theory $\varphi_{init} \in \mathcal{DNF}_n$ has classification error $\alpha = Pr(\,\varphi_{init}\,\Delta\,\varphi_{cor}\,)$, where φ_{cor} is the correct target theory and

$$\varphi_{init}\,\Delta\,\varphi_{cor} \quad = \quad \{\ \tilde{x} \mid (\varphi_{init}(\tilde{x}) \wedge \neg\varphi_{cor}(\tilde{x})) \ \vee\ (\neg\varphi_{init}(\tilde{x}) \wedge \varphi_{cor}(\tilde{x}))\ \}$$

is the symmetric set difference between instances satisfied by φ_{init} versus φ_{cor}. Our goal is to produce a new theory $\varphi' \in \mathcal{DNF}_n$ whose error is, with probability at least $1-\delta$, at most $\epsilon \times \alpha$. We want to do this using a number of samples that is polynomial in $1/\alpha$, $1/\epsilon$, $\ln(1/\delta)$ and in some measure of the "difference" between our current theory and φ_{cor}. For this purpose, we use the measure:

$$r(\varphi_{cor},\ \varphi_{init}) \quad = \quad |\varphi_{cor} - \varphi_{init}| \ + \ |\varphi_{init} - \varphi_{cor}|$$

which is the syntactic difference between φ_{init} and φ_{cor} — *i.e.*, the total number of terms that must be either added to, or removed from, φ_{init} to form the desired φ_{cor}. (Of course, this "$\varphi_{cor} - \varphi_{init}$" is the set-difference between the set of terms in φ_{cor} and the set in φ_{init}.) Here, we assume that each complete sample is drawn at random from a stationary distribution, then blocked according to the $\beta_{\varphi_{cor}}$ blocker (*i.e.*, the blocker based on the target φ_{cor} formula).

We can achieve this task using the MODIFY-DNF algorithm shown in Figure 4:

Theorem 3 *Given any target s-term DNF formula $\varphi_{cor} \in \mathcal{DNF}_{n,s}$, let $\varphi_{init} \in \mathcal{DNF}_n$ be an initial formula whose syntactic difference is $r = r(\varphi_{cor}, \varphi_{init})$ and whose classification error is $\alpha = Pr(\,\varphi_{cor}\,\Delta\,\varphi_{init}\,) > 0$. Then, for any values of $\epsilon, \delta \in (0,1)$ and any distribution P, under the $B^{(DNF)}$ model using the $\beta_{\varphi_{cor}}$ blocker,* MODIFY-DNF$(\varphi_{init}, n, r, \alpha, \epsilon, \delta)$ *will*

[11]For example, notice the initial concept $\varphi_{init} = x_1x_2$ is unable to label the blocked instance $\langle 1, *, * \rangle$, as it would label the $\langle 1, 1, ? \rangle$ extension "T", and the $\langle 1, 0, ? \rangle$ extension "F". This $\langle 1, *, * \rangle$ instance could be the result of blocking the $\langle 1, 1, 0 \rangle$ instance using the blocker for the target concept $\varphi_{cor} = x_1 \vee x_3$.

return a formula $\varphi' \in \mathcal{DNF}_{n,\,s+r}$*, whose error (on unblocked, unlabeled instances) is, with probability at least* $1-\delta$*, at most* $\alpha \times \epsilon$*. Moreover,* Modify-DNF *requires* $O(\frac{r}{\alpha\epsilon}\ln\frac{r}{\delta})$ *blocked labeled samples.*

This resulting φ' formula agrees with each training sample seen; *i.e.*, if Modify-DNF used the sample $\langle \tilde{x}, T\rangle$, then $\varphi'(\tilde{x}) = T$, and if Modify-DNF used $\langle \tilde{x}, F\rangle$, then $\varphi'(\tilde{x}) = F$.

Note that Modify-DNF's sample complexity does not depend on s, but rather on r, which could be much smaller than s. On the other hand, Modify-DNF's *computational* complexity does depend on s, as it has to consider all of φ_{init}'s terms, and there could be as many as $s+r$ of these.

If we did not know r in advance, we can use the standard iterative doubling technique to find an appropriate value. If so, it may make sense to run this Modify-DNF algorithm in parallel with Learn-DNF, and stop as soon as either algorithm finds an acceptable candidate theory. The sample complexity of the resulting pair-of-algorithms is linear in $\min\{r, s\}$, which could be a big advantage if, for example, r was exponentially larger than s (which would happen if φ_{init} includes an exponential number of terms, and φ_{cor} includes only a polynomial number). This would allow us to avoid wasting (many) samples "unlearning" φ_{init}.

Our sample complexity has the same $O(\frac{r}{\epsilon}\ln r)$ form that Mooney provides in [Moo94], for revising a theory using completely specified training examples (*i.e.*, his theory revision system does not have access to our very-helpful teacher, which provides the "relevance" information). His result, however, deals with the task of finding a distance-r theory (*i.e.*, a theory within a syntactic distance of r from the current theory) whose error is under ϵ, assuming there is a 0-error distance-r theory. By contrast, we address the harder task of finding a distance-r theory whose error is only $\alpha\epsilon \le \epsilon$ (*i.e.*, which is multiplicatively reduced from the error of the current theory). The analysis in [Gre95] also considers revising a theory using complete examples; it however considers the "agnostic" setting [KSS92] — *i.e.*, it does not assume there is a 0-error distance-r theory. It proves that $O(\frac{r}{\epsilon^2}\ln r)$ complete training examples are sufficient to identify a theory whose error is within ϵ of the *best* distance-r theory. (This best theory may not have 0 error.) Notice that neither [Moo94] nor [Gre95] explicitly deals with the error of the initial theory.

Greiner [Gre95] also considers the *computational* challenge of finding this near-optimal theory given these samples, proving that it is NP-hard to find a theory whose error is even close to (*i.e.*, within a small polynomial of) the optimal distance-r theory. By contrast, note the Modify-DNF algorithm is able to solve a harder task (*i.e.*, finding a theory whose error is $\alpha\epsilon \le \epsilon$) in polynomial-time; this suggests a further advantage of using the given relevance information.

Theorem 3 assumes that each instance is blocked by $\beta_{\varphi_{cor}}$. To motivate a slightly different blocker, imagine the only way to determine the value of each attribute is to perform an expensive test. We would then perform only the tests deemed essential by the best available authority, namely φ_{init}, *unless* φ_{init} led us astray. Here, we would call upon a human expert, who would use his φ_{cor} to extract a new set of appropriate attribute values. That is, we

$d_2 \quad =$

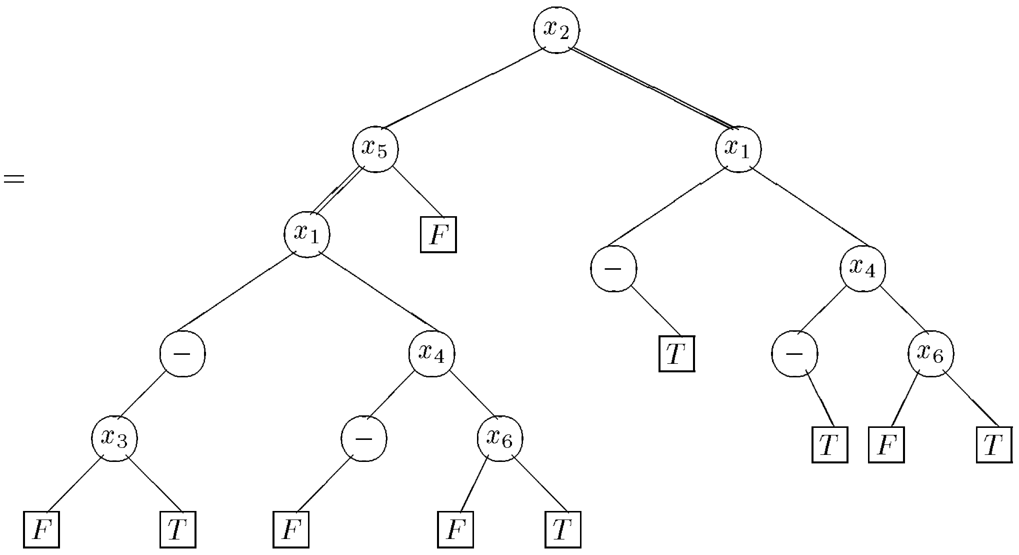

Figure 5: Modify-DT applied to d_0 (from Fig. 2), using $\langle\langle 0,*,1,*,*,*\rangle, F\rangle$

would use the blocker

$$\beta_*(\tilde{x}) \quad = \quad \begin{cases} \beta_{\varphi_{init}}(\tilde{x}) & \text{if} \quad \varphi_{init}(\tilde{x}) = \varphi_{cor}(\tilde{x}) \\ \beta_{\varphi_{cor}}(\tilde{x}) & \text{otherwise} \end{cases}$$

Notice a system using this blocker will tell the learner both (1) of the tests $\beta_{\varphi_{init}}$ considered superfluous, which were really relevant; and (2) of the tests $\beta_{\varphi_{init}}$ considered relevant, which were really superfluous. It is easy to see that that Modify-DNF algorithm also works if we use this β_* blocker.[12]

Modifying Decision Trees: There is a trivial way to use the Modify-DNF algorithm to modify a theory encoded as a decision tree: First express the decision tree t as the DNF formula φ_t, and then use a set of samples to learn a new DNF formula. Unfortunately, the formula produced need not correspond to a small decision tree. For this situation, we use the Modify-DT procedure which, given an initial decision tree d_{init}, together with parameters specified below, first collects an (appropriately sized) set of blocked-and-labeled samples, and then passes these samples, along with d_{init}, to BuildDT$'$. This BuildDT$'$ algorithm differs from Build-DT only in its termination condition. Recall from Figure 3 that Build-DT terminates when all remaining samples have the same label, or when this set is empty. BuildDT$'$, however, continues to grow the tree even when all samples have the same label, until there are no more variables left to test. Using the empty set $\{\}$ to denote an empty tuple (where all variables are either $*$, or have been projected away) we can write the new

[12]In fact, this research project was originally motivated by exactly this task; *viz.*, revising an existing diagnostic theory where each training sample contains only the information used to reach a specific conclusion, from a particular faulty knowledge base. Langley *et al.* [LDRG94] describes our implementation, together with a corpus of experiments, within this model.

termination condition as:

```
If    (every entry in S is ⟨{}, T⟩ )
Then  NT.LeafLabel  =  T
      Return( NT )
If    (every entry in $S$ is ⟨{}, F⟩ )
Then  NT.LeafLabel  =  F
      Return( NT )
If    (S is empty)
Then  Let NT = "diminished version of initial $d_{init}$ tree"
      Return( NT )
```

Here, each "diminished subtree of d_{init}" corresponds to the parts of the initial d_{init} that were not eliminated in the path to that leaf. An equivalent way of thinking about the last step is that we first append an exact copy of d_{init} to such leaves. However, d_{init} may test some variables that were already tested in the path above where d_{init} was appended. Thus, we can simplify ("diminish") the appended copy of d_{init} by removing these repeated tests.

As an example, suppose Modify-DT, working on the initial tree d_0 (from Figure 2), received only the labeled instance $a = \langle\langle *, 0, *, *, 1, * \rangle, F\rangle$. It would then produce the d_2 tree shown in Figure 5. (In this figure, the arcs shown as double lines point to (diminished) versions of d_{init}.) The labeled instance a corresponds to the path from x_2 to x_5 (*i.e.*, down x_2's left-child), and then from that x_5 to its right child (labeled with $\boxed{\text{F}}$). Now observe the two other subtrees, which correspond to the $x_2 = 1$ and $x_2 = 0$-and-$x_5 = 0$ situations; here Modify-DT places diminished versions of d_0, which correspond to the $\{x_2 \mapsto 1\}$ and $\{x_2 \mapsto 0,\ x_5 \mapsto 0\}$ assignments. For pedagogical reasons, Figure 5 includes "phantom nodes", shown as "$\ominus$" which correspond to where x_2 and x_5 were in the original d_0 tree. Of course the "diminishing" process (simplification) removes these nodes, because they correspond to tests that were made higher in the tree.

To understand why Modify-DT works, note that we can view its output as consisting of an initial "prefix" subtree, which we call d_S, some of whose "leaves" point to versions of d_0. If we present d_S with one of the examples in S, *i.e.*, one of the examples in the training set, we reach a leaf of d_S labeled correctly for s (*i.e.*, T or F); for each such s, we never reach a leaf of d_S which points to a version of d_0. Thus, Modify-DT's result classifies the training data correctly. Now consider an instance that does not match one of the training examples. When d_S is presented with this instance, it will always reach one of the leaves to which a version of d_0 was appended. Thus, on such examples, Modify-DT's result will produce the same label that d_0 would have produced. To summarize: Modify-DT *produces a tree that will return the correct label for each training instance; but, in every other situation, it will produce the same label that d_0 had returned.*

To state the result more precisely, we first need to define the syntactic difference between the decision trees $d, e \in \mathcal{DT}_n$: Let $d^{(+)} = \{d_1^{(+)}, \ldots, d_k^{(+)}\}$ (resp., $d^{(-)} = \{d_1^{(-)}, \ldots, d_\ell^{(-)}\}$) be the set of conditions of the paths in d that lead to leaf nodes labeled T (resp., labeled F); we similarly define $e^{(+)} = \{e_1^{(+)}, \ldots, e_{k'}^{(+)}\}$ and $e^{(-)} = \{e_1^{(-)}, \ldots, e_{\ell'}^{(-)}\}$ for the e tree. (Hence, for Figure 2's d_0, $d_0^{(+)} = \{\ \bar{x}_1\bar{x}_2x_3,\ x_1\bar{x}_4x_2,\ x_1x_4x_6\ \}$ and $d_0^{(-)} = \{\ \bar{x}_1\bar{x}_2\bar{x}_3,\ \bar{x}_1x_2,\ x_1\bar{x}_4\bar{x}_2,\ x_1x_4\bar{x}_6\ \}$.)

```
Algorithm Modify-DT( $d_{init}$ : DT_Type; $r, n$ : $\mathcal{N}$; $\alpha, \epsilon, \delta \in (0,1)$ ): DT_Type
  Draw $m_r = \frac{2r}{\alpha\epsilon} \ln \frac{16n}{\delta}$ training (blocked, labeled) samples, $S$
    /* Alternatively, if $\alpha$ is not known a priori, we can instead collect $\frac{2r}{\epsilon} \ln \frac{16r}{\delta}$ samples
       that $d_{init}$ does not label correctly (i.e., either mislabels, or is unable to label) */
  Let $d_{out}$ = Build-DT'( $S$, $d_{init}$ )
  Return $d_{out}$
End Modify-DT
```

Figure 6: The Modify-DT algorithm for modifying an initial Decision Tree

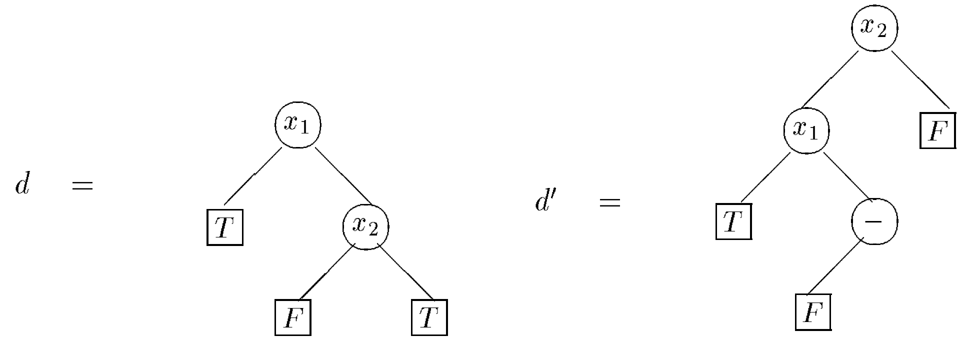

Figure 7: Simple Decision Trees

We then define

$$r_{DT}(d, e) \quad = \quad |d^{(+)} - e^{(+)}| \; + \; |e^{(+)} - d^{(+)}| \; + \; |d^{(-)} - e^{(-)}| \; + \; |e^{(-)} - d^{(-)}|$$

as the set difference between the corresponding sets. We can also define $\alpha = Pr(\, d \,\Delta\, e \,)$ to be the probability that d will label an instance differently than e. With these definitions, we can prove:

Theorem 4 *For any given s-leaf target decision tree $d_{cor} \in \mathcal{DT}_{n,s}$, let $d_{init} \in \mathcal{DT}_n$ be an initial decision tree whose syntactic difference is $r = r_{DT}(d_{cor}, d_{init})$ and whose classification error is $\alpha = Pr(\, d_{cor} \,\Delta\, d_{init} \,) \; > \; 0$. Then, for any values of $\epsilon, \delta \in (0,1)$ and any distribution P, under the $B^{(DT)}$ model using the $\beta_{d_{cor}}$ blocker,* Modify-DT$(d_{init}, n, r, \alpha, \epsilon, \delta)$ *will return a tree $d' \in \mathcal{DT}_{n,\,(r+s)r}$, whose error (on unblocked, unlabeled instances) is, with probability at least $1 - \delta$, at most $\alpha \times \epsilon$. Moreover,* Modify-DT *requires $O(\frac{r}{\alpha\epsilon} \ln \frac{n}{\delta})$ blocked labeled samples.*

Notice the boolean function produced by Modify-DT is *not* the same as the function produced by first converting d_0 to a DNF formula, and then calling Modify-DNF: While Modify-DNF ruthlessly removes any term that is consistent with any negative instance, the gentler, more conservative Modify-DT instead tries hard to preserve as much of the original d_0 as possible. As a simple example, consider the d decision tree shown on the left side of Figure 7, and assume we receive the negative sample $\langle x_2, F \rangle$. Here, Modify-DT would return the d' decision tree shown in Figure 7's right side, which corresponds to the boolean function "$\bar{x}_1\bar{x}_2$". Notice this function is reasonable, as (1) d' will label negatively any instance that is subsumed by x_2, and (2) of the remaining space (*i.e.*, where $x_2 = 0$), d' returns the same label that d would: which is T iff $x_1 = 0$.

On the other hand, notice the original d encodes the boolean function $\bar{x}_1 \vee x_1x_2$. Here, on seeing $\langle x_2, F\rangle$, Modify-DNF would remove *both* terms (as each is consistent with x_2), leaving only the constant "F" function.

One might consider defining a Modify-DNF$'$ algorithm that resembles Modify-DT by "preserving as much of the initial φ_{init} as possible". That is, instead of simply removing any term in φ_{init} that is consistent with some negative example, we might replace it with a weakened version that is inconsistent with all negative examples. So given $\langle x_2\bar{x}_3, F\rangle$ as an example, we might replace $x_1 \in \varphi_{init}$ by $x_1\bar{x}_2$ and x_1x_3. Although this is analogous to Modify-DT, it is problematic. The difference is that a target DNF formula φ_{cor}, unlike a target decision tree, might generate exponentially many negative examples. Unless we have seen almost all of these, the error could still be large. Suppose that, in the above example, x_1 is in fact completely irrelevant to the target formula. Then all terms in φ_{init} involving x_1 risk false positives. In fact, the algorithm's hypothesis will not be sufficiently correct until it has seen enough negative examples that the terms involving x_1 have either vanished, or have very small coverage. Unfortunately, this could require very many samples, and furthermore, lead to a very lengthy hypothesis.

4.2 Degradation of the Training Data

So far our $B^{(DT)}$ and $B^{(DNF)}$ models have assumed that the blocker removes all-and-only the superfluous attribute values. There may, however, be situations where the environment/teacher reports an irrelevant value, or fails to report a relevant one. In general, we can model this by assuming that a "degradation module" can interfere with the blocked data, degrading it before presenting it to the learner; see Figure 8. This subsection presents a range of results that deal with several types of degradation processes. In all these results, we assume blocking is done with the standard $B^{(DNF)}$ model. Furthermore, we continue to use the PAC criterion for evaluating learning algorithms. In particular, we judge success by performance on complete (unblocked, undegraded) instances even though training is done with blocked and possibly degraded samples.

4.2.1 Attribute Degradation

The $\gamma^{PA}_{\vec{\mu},\vec{\nu}}$ degrader randomly degrades each blocked training instance, on an attribute-by-attribute basis: If the blocker passes unblocked the attribute x_i (*i.e.*, it is relevant), then the $\gamma^{PA}_{\vec{\mu},\vec{\nu}}$ degrader will, with probability μ_i, set x_i's value to $*$; and with probability $1-\mu_i$, simply pass the correct value. Similarly, if the blocker has set attribute x_i to $*$ (*i.e.*, found x_i to be irrelevant), the $\gamma^{PA}_{\vec{\mu},\vec{\nu}}$ degrader will, with probability ν_i, reset x_i to its original unblocked value; otherwise, with probability $1 - \nu_i$, it will simply pass $x_i = *$. Notice the values of μ_i can differ for different i's, as can the values of ν_i. (Hence, this is "non-uniform" attribute degradation.)

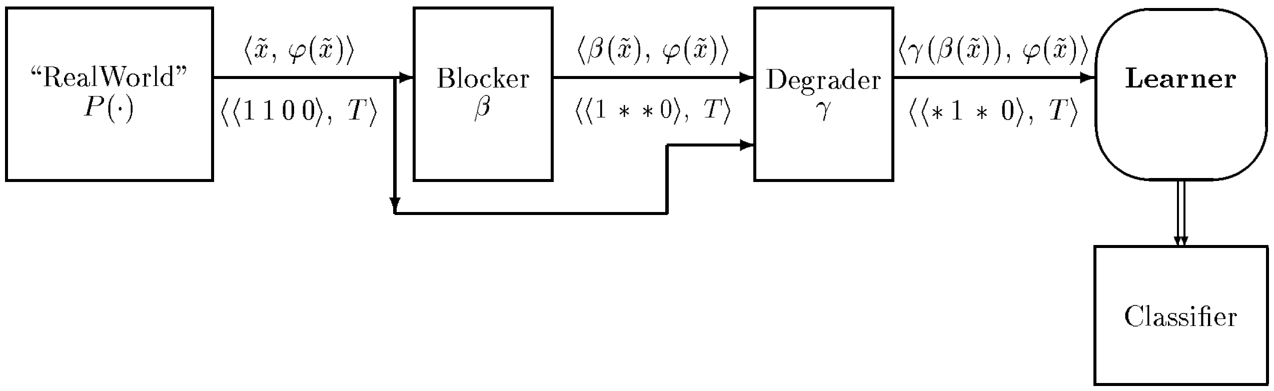
"RealWorld" $P(\cdot)$
$\langle \tilde{x},\, \varphi(\tilde{x})\rangle$
$\langle\langle 1\,1\,0\,0\rangle,\, T\rangle$
Blocker β
$\langle \beta(\tilde{x}),\, \varphi(\tilde{x})\rangle$
$\langle\langle 1 * * 0\rangle,\, T\rangle$
Degrader γ
$\langle \gamma(\beta(\tilde{x})),\, \varphi(\tilde{x})\rangle$
$\langle\langle * 1 * 0\rangle,\, T\rangle$
Learner
Classifier

Figure 8: Blocking, then Degradation, Model

For any $\mu, \nu \in [0,1]$, we say a learning algorithm can *PAC-learn with* (μ,ν) *attribute degradation* if it can PAC-learn given any $\gamma^{PA}_{\vec{\mu},\vec{\nu}}$ degrader, where each $\mu_i \le \mu$ and $\nu_i \le \nu$.

The previous sections all implicitly used the $\gamma^{PA}_{\vec{0},\vec{0}}$ degrader, to show that we can learn with $\mu = \nu = 0$. It is easy to show that there are some upper bounds on the amount of degradation we can tolerate.

Proposition 5

1. *PAC-learning* $\mathcal{DNF}_{n,s}$ *with* $(0,\, 1)$ *attribute degradation is as hard as PAC-learning* $\mathcal{DNF}_{n,s}$ *in the standard model (with complete attributes).*
2. *It is impossible to PAC-learn* $\mathcal{DNF}_{n,s}$ *with* $(1,\, 0)$ *attribute degradation.*
3. *PAC-learning* $\mathcal{DNF}_{n,s}$ *with* $(\frac{1}{2},\, \frac{1}{2})$ *attribute degradation is as hard as learning* $\mathcal{DNF}_{n,s}$ *in the standard model.*

Parts 1 and 2 are immediate: (0, 1)-degradation simply presents *all* attribute values, which *is* the standard model; and under (1, 0) degradation, each training instance is simply a list of $*$'s. If $(\mu,\nu) = (\frac{1}{2},\, \frac{1}{2})$ then we have lost all information about relevance, which means a $*$ is as likely to indicate relevance as not, and so we are clearly in a no better situation (and perhaps much worse off) than learning DNFs in the standard model. (To be more precise: Imagine we had an algorithm $L_{1/2,1/2}$ that could PAC-learn given (1/2, 1/2)-degradation. It would then be trivial to PAC-learn given all attribute values: Given any completely specified instance, randomly change 1/2 of the values to $*$, and then pass the resulting partially-specified instance to $L_{1/2,1/2}$.)

There are, however, algorithms that can learn with small amounts of degradation. First, it is relatively easy to learn from positive examples alone, with up to $O(\ln n/n)$ degradation of either single type. The idea in each case is that we can expect to see most important terms in the DNF formula at least once, and identify them as such (rather than as degraded terms). In the following two results, we consider μ and ν less than $\min\{k \ln n/n,\, 0.5\}$ for some *constant* k.

```
Algorithm LEARN-DNF^ν( $n,\ s,\ k : \mathcal{N}$; $\epsilon, \delta \in (0,1)$ ): DNF_Type
  Draw $m_\nu = \frac{4sn^{2k}}{\epsilon} \ln \frac{s}{\delta}$ training (blocked, degraded and labeled) samples
  Let $S^+$ be the positive samples.
  Let $T$ be the set of "smallest" terms in $S^+$.
    /* For example, if $S^+$ includes both $x_1x_2$ and $x_1x_2x_3$, then $T$ includes only $x_1x_2$. */
  Return disjunction of terms in $T$.
End LEARN-DNF^ν
```

Figure 9: The LEARN-DNF$^\nu$ algorithm for $(0,\ k\ln(n)/n)$ attribute degradation

```
Algorithm LEARN-DNF^μ( $n, s, k : \mathcal{N}$; $\epsilon, \delta \in (0,1)$ ): DNF_Type
  Draw $m_\mu = \frac{24s^2n^{2k}}{\epsilon} \ln \frac{3s}{\delta}$ training (blocked, degraded and labeled) samples
  Let $S^+$ be the positive samples.
  Let $T$ be the set of "largest" terms in $S^+$.
    /* For example, if $S^+$ includes both $x_1x_2$ and $x_1x_2x_3$, then $T$ includes only $x_1x_2x_3$ */
  Let $T_*$ be set of terms in $T$ that occur at least $6sln\frac{36}{\delta}$ times in $S^+$.
  Return disjunction of terms in $T_*$.
End LEARN-DNF^μ
```

Figure 10: The LEARN-DNF$^\mu$ algorithm for $(k\ln n/n,\ 0)$ attribute degradation

Theorem 6 *The* LEARN-DNF$^\nu$ *algorithm, shown in Figure 9, can PAC-learn* $\mathcal{DNF}_{n,s}$ *with* $(0, k\frac{\ln n}{n})$ *attribute degradation, using* $O(\frac{sn^{2k}}{\epsilon} \ln \frac{s}{\delta})$ *examples, and using only positive examples.*

Theorem 7 *The* LEARN-DNF$^\mu$ *algorithm, shown in Figure 10, can PAC-learn* $\mathcal{DNF}_{n,s}$ *with* $(k\frac{\ln n}{n}, 0)$ *attribute degradation, using* $O(\frac{s^2n^{2k}}{\epsilon} \ln \frac{s}{\delta})$ *examples, and using only positive examples.*

The situation is even better for $(c \log n)$-$\mathcal{DNF}_{n,s}$ formula under $(\mu, 0)$ degradation. Here, as each term has only $c \log n$ variables, we are much more likely to see an undegraded example of each term in any sample set. As a corollary of Theorem 7, we can prove:

Theorem 8 *The* LEARN-DNF$^\mu$ *algorithm can PAC-learn* $(c \log n)$-$\mathcal{DNF}_{n,s}$ *with* $(\mu, 0)$ *attribute degradation, using positive examples alone, for any (known) value of* μ *bounded away from* 1. *To achieve this, we call* LEARN-DNF$^\mu$($n,\ s,\ c\ln(1-\mu)/2,\ \epsilon,\ \delta$).

Finally, we can also learn with both types of degradation. To see this result in its best light, it is useful to distinguish between the degradation rate for positive instances and negative instances. In particular, the $\gamma^{PA'}_{\vec{\mu}^{(+)},\vec{\nu}^{(+)},\vec{\mu}^{(-)},\vec{\nu}^{(-)}}$ degradation model will apply $(\vec{\mu}^{(+)}, \vec{\nu}^{(+)})$ attribute degradation to each positive instance, and apply $(\vec{\mu}^{(-)}, \vec{\nu}^{(-)})$ degradation to negative instances. Hence, given an instance $\langle\langle 1, 0, *, 1\rangle, T\rangle$, it will change $x_1 = 1$ to $*$ with probability $\mu_1^{(+)}$, and change $x_3 = *$ to (say) 1 with probability $\nu_3^{(+)}$. But given the instance $\langle\langle 1, 0, *, 0\rangle, F\rangle$, it will change $x_1 = 1$ to $*$ with probability $\mu_1^{(-)}$, and change $x_3 = *$

```
Algorithm LEARN-DNF$^{\mu\nu}$( $n$, $s$, $k$ : $\mathcal{N}$; $\epsilon,\delta \in (0,1)$ ): DNF_Type
  Draw $m_{\mu\nu} = \frac{48}{\epsilon}(\log\frac{1}{\delta})(\frac{8ks}{\epsilon}\ln\frac{4s}{\delta})^{2k+1}$ training (blocked, degraded and labeled) samples
  Let $S^+$ be the first, at most $\frac{8ks}{\epsilon}\ln\frac{4s}{\delta}$, positive samples seen.
  Let $S^-$ be all the negative samples seen.
  Let $C$ be set of all $2k$-element subsets of $S^+$.
  Initialize $T$ = {}.
  For each $c = \{c^{(1)},\ldots,c^{(2k)}\} \in C$:
            /* Propose a candidate $t = \langle t_1,\ldots,t_n\rangle$ by component-wise voting */
    For each $i = 1 \ldots n$
       For $\chi \in$ { 1, 0, * }
          Let num$_i$(c, $\chi$) = $|\{c^{(j)} : c_i^{(j)} = \text{``}\chi\text{''}\}|$
       Let $t_i$ = argmax{ num$_i$(c,$\chi$) : $\chi \in \{1, 0, *\}$ }
            /* Decide whether to accept term $t = \langle t_1,\ldots,t_n\rangle$ */
    If $t$ contradicts at least $\frac{\epsilon\,|S^-|}{4\,|C|}$ of the terms in $S^-$,
       Add $t$ to $T$.
  Return the disjunction of terms in $T$.
End LEARN-DNF$^{\mu\nu}$
```

Figure 11: The LEARN-DNF$^{\mu\nu}$ algorithm for $(n^{-1/k}/8,\ n^{-1/k}/8,\ \epsilon/(8m_{\mu\nu}),\ 1)$ degradation.

to (say) 1 with probability $\nu_3^{(-)}$. As we see shortly, $\vec{\mu}^{(-)}$ degradation can be very disruptive. In contrast, our next result is unaffected by arbitrary $\vec{\nu}^{(-)}$ degradation. In fact, allowing for $\vec{\nu}^{(-)}$ degradation is somewhat redundant, as our definition of the $B^{(DNF)}$ model already allows the *blocker* to reveal extra attributes in negative examples; thus $\vec{\nu}^{(-)}$ is included only for completeness.

Here, we can deal with positive degradation of order $O(n^{-\frac{1}{k}})$, for arbitrary k. This degradation is sufficiently large that we may not ever see an entirely correct (*i.e.*, completely undegraded) term, within polynomially many samples. Thus, the basic strategy used in the earlier results does not apply. However, we can recover terms by collecting (all) subsets of $2k$ positive samples, and "voting" (for use of a similar technique, see [KMR$^+$94]). That is, we construct a term from the subsample by considering each variable x_i in turn, and setting it to 0, 1, or $*$ according to which value is given most often to x_i in the subsample (ties can be broken in an arbitrary fashion). Even if a term is unlikely to ever appear without any degradation, this voting procedure can recover it. The reason is that, by looking at all subsets of $2k$ positive samples, we expect to include sets of samples that are all degradations of the same term. Although a relevant attribute might be missing from any given instance in such a set, it is likely to be present in several of the others. Similarly, although an irrelevant attribute might be revealed in one sample, the attribute is unlikely to appear in very many others. Thus, "voting" in such a subsample can be expected to identify the truly relevant attributes with high probability. Of course, the procedure will also look at many subsets of $2k$ positive samples that are not all degradations of a common term. Voting in such subsets will tend to produce spurious terms, which should not be included in the learner's hypothesis.

We use the negative examples to filter out these inappropriate terms.[13] Unfortunately, we can tolerate much less degradation for negative examples. The following result requires $\mu^{(-)} < \epsilon/(8m_{\mu\nu})$, where $m_{\mu\nu}$ is the (polynomial size) quantity given in Figure 11.

Theorem 9 *The* LEARN-DNF$^{\mu\nu}$ *algorithm, shown in Figure 11, can PAC-learn* $\mathcal{DNF}_{n,s}$ *with* $(n^{-\frac{1}{k}}/8,\ n^{-\frac{1}{k}}/8,\ \epsilon/(8m_{\mu\nu}),\ 1)$ *attribute degradation, using* $O(\frac{1}{\epsilon}(\ln\frac{1}{\delta})(\frac{s}{\epsilon}\ln\frac{s}{\delta})^{2k+1})$ *examples.*

4.2.2 Adversarial Degradation

The previous section considered probabilistic degradation, of the sort that might arise from a noisy communication channel. Even at best, such models will only approximate the "real" degradation process in a particular task, and in other cases these simple probabilistic models will be completely inappropriate. Thus it is useful to consider a broad variety of other degradation models. In particular, another commonly studied class of models are those which regard the degradation process as a malicious adversary, who knows the DNF formula, the sample distribution, the blocker and even our specific learning algorithm, and has some (limited) power to alter examples in an arbitrary fashion.

We consider first the $\gamma_k^{(A,\pm inst)}$ adversary that can change up to k variables on each instance, for some constant k. For any $k \in \{0,\ldots,n\}$, we say a learning algorithm can *PAC-learn with k/instance degradation* if it can PAC-learn in the presence of such a $\gamma_k^{(A,\pm inst)}$ degrader. (Here, we regard $*$ as simply another value that the degradation process can change a value from, or to.) In general, such an adversary can prevent learning, even if $k=1$:

Proposition 10 *It is impossible to PAC-learn* $\mathcal{DNF}_{n,s}$ *with* 1*/instance degradation,* $\gamma_1^{(A,\pm inst)}$.

To see this, consider the two simple DNF formulae $\varphi_1 = x_1$ and $\varphi_2 = \bar{x}_1$, and notice it is critical to see the value of x_1 to distinguish between these two different functions. Here, however, a $\gamma_1^{(A,\pm inst)}$ degrader can simply conceal this attribute in all samples.

However, we can consider less powerful adversaries, such as the $\gamma_k^{(A,+inst)}$ (resp., $\gamma_k^{(A,-inst)}$) degraders, which can only degrade *positive* (resp., negative) samples. In each case, learning is sometimes possible. The result for $\gamma_k^{(A,-inst)}$ follows immediately from the fact that we can learn DNF formula from undegraded *positive* samples alone; see Theorem 2. For $\gamma_k^{(A,+inst)}$, we need only enumerate all the terms from which the degraded positive examples might have come (note there are polynomially many of these), and then filter using negative examples. This is an application of the idea of *polynomial explainability* [KR93].

[13] Another way of filtering out some inappropriate terms is to consider only terms that are voted for "overwhelmingly" by some size-$2k$ subset; *i.e.*, to retain the term generated by a subset only if $\max\{\text{num}_i(c,\chi)\}$ is at least some minimal value for each attribute. However, negative examples are still needed as a final filter. Such a refinement requires essentially the same number of samples — *i.e.*, it would only improve on the original LEARN-DNF$^{\mu\nu}$ algorithm by a constant factor.

Theorem 11 *It is possible to PAC-learn $\mathcal{DNF}_{n,s}$ with k/positive-instance degradation $\gamma_k^{(A,+inst)}$, for any constant k. This holds even in the presense of $(n^{-\frac{1}{k}}/8,\ n^{-\frac{1}{k}}/8,\ \epsilon/(8m_{\mu\nu}),\ 1)$ attribute degradation.*

It is possible to PAC-learn $\mathcal{DNF}_{n,s}$ with k/negative-instance degradation $\gamma_k^{(A,-inst)}$, for any constant k. This holds even in the presence of $(0,\ O(\ln n/n),\ 1,\ 1)$ attribute degradation.

A quite different type of degradation occurs if an adversary can *arbitrarily* change instances [KL93]. However, we assume that the adversary has to pass a certain fraction of instances unchanged: *i.e.*, on each instance the adversary will, with probability $1-\eta$, show the learner exactly the appropriate blocked instance. However, with probability η, the adversary can replace the instance with *anything* else, as long as the same class label is unchanged. (*E.g.*, if the target concept is $\varphi = x_1x_2$ and the blocked labeled instance is $\langle x_1x_2,\ T\rangle$, the adversary may replace this instance with say $\langle \bar{x}_1x_9,\ T\rangle$, even though the correct label for $\bar{x}_1x_9$ is not T. It cannot, however, replace this $\langle x_1x_2,\ T\rangle$ with $\langle \bar{x}_1x_9,\ F\rangle$, nor even with $\langle x_1x_2,\ F\rangle$.)

We call this $\gamma_\eta^{(A,+samp)}$ (resp., $\gamma_\eta^{(A,-samp)}$, $\gamma_\eta^{(A,\pm samp)}$) degradation if the adversary has the power to change positive (resp., negative, any) examples in this fashion. It is easy to show that an analogue to Theorem 11 holds. That is, we can tolerate $\gamma_\eta^{(A,+samp)}$ for any fixed $\eta < 1$. The idea is that, by drawing $O(1/\eta)$ times as many positive examples as we would if there were no $\gamma_\eta^{(A,+samp)}$ degradation, we should see "enough" undegraded examples (and, if we use negative examples to filter, the presence of the other corrupted examples cannot harm us). In contrast, we can tolerate nontrivial $\gamma_\eta^{(A,-samp)}$ degradation only in settings where negative examples are unnecessary anyway.

Proposition 12 *It is possible to PAC-learn $\mathcal{DNF}_{n,s}$ with $\gamma_\eta^{(A,+samp)}$ degradation for constant $\eta < 1$. This holds even in the presence of $(n^{-\frac{1}{k}}/8,\ n^{-\frac{1}{k}}/8,\ \epsilon/(8m_{\mu\nu}),\ 1)$ attribute degradation.*

It is possible to PAC-learn $\mathcal{DNF}_{n,s}$ with $\gamma_\eta^{(A,-samp)}$ for any η. This holds even in the presence of $(0, O(\ln n/n), 1, 1)$ attribute degradation.

Note that $\gamma_\eta^{(A,\pm samp)}$ degradation is even harder, as is the (very similar) model in which the adversary can corrupt only positive examples but gets to change the class label. In either case, we cannot trust negative examples enough to use them as a filter for unwanted positive terms.

4.3 Classification and Attribute Noise

For the reasons hinted in the previous section, *classification noise* — where a positive example may be reported as negative, or vice versa — tends to be problematic when it occurs together with other forms of degradation. However, if classification noise is all we have to worry about, things are much better. In the following, let α be the fixed probability that an example is mislabelled.

The basic idea is simply to collect a somewhat larger sample, so that most important terms in the DNF formula (or paths in the decision tree) are likely to have been seen many times. Any term might have been seen labeled both T and F, but we can assume that the correct label is the one occurring more often. If the sample size is large enough, this will only have a small probability of leading to error.

Theorem 13 *It is possible to PAC-learn* $\mathcal{DNF}_{n,s}$ *with any classification noise* $\alpha < 0.5$. *This holds even in the presense of* $(0, O(\ln n/n), 0, 1)$ *attribute degradation.*

Essentially the same ideas also show:

Theorem 14 *It is possible to PAC-learn* $\mathcal{DT}_{n,s}$ *with classification noise* $\alpha < 0.5$.

The final corruption process we consider is attribute noise. By this, we refer to a process that can probabilistically change the value of an attribute to any value in $\{0, 1, *\}$. This differs from attribute degradation by allowing more: for instance, $x_i = 1$ can be changed to $x_1 = 0$. The probability of each change can depend on the particular attribute being changed (*e.g.*, x_1 or x_2) and the type of change (*e.g.*, 1 to 0, or 0 to 1); in the following we use ρ simply as an upper bound on these probabilities.

In some contexts, the difference between attribute noise and degradation is important. However, an examination of the proof of Theorem 9 shows that this difference is not important here. Hence:

Proposition 15 *It is possible to PAC-learn* $\mathcal{DNF}_{n,s}$ *with attribute noise* $\rho < n^{-\frac{1}{k}}/16$. *This holds even in the presense of* $(n^{-\frac{1}{k}}/16,\ n^{-\frac{1}{k}}/16,\ \epsilon/(16m_{\mu\nu}),\ 1)$ *attribute degradation.*

We close this section by noting that it would be interesting to develop algorithms that can tolerate many different types of corruption simultaneously. However, we expect that the connection between missing data and "irrelevance" will typically be imperfect, and will furthermore lack any obvious, clean, formalization. If we nevertheless wish to make use of the information implied by the missing information — the goal of this work — it will be important to tolerate as much, and as many different types of, corruption as possible. Proposition 15 is a good step in this direction, although there is no reason to suppose that more cannot be achieved.

5 Conclusion

Most learning systems are designed to work best when the training data consists of completely-specified attribute-value tuples. To the extent that the issue has been considered, missing attribute values have generally been regarded as extremely undesirable. The main point of this paper is that sometimes the opposite is true. Sometimes the fact that an attribute is missing is very informative: it tells us about *relevance*. This information can be so useful that very hard problems can become trivial.

Moreover, this exact situation, where missing information can be useful, can occur in practice. Most classification systems perform and record only a small fraction of the set of possible tests to reach a classification. So if training data has been produced by such a system — as in our motivating example of a student examining medical records — our model of superfluous value blocking seems very appropriate.

This paper provides several specific learning algorithms that can deal with the partially-specified instances that such classification systems tend to produce. We show, in particular, that it can be very easy to "PAC learn" decision trees and DNF formulae in this model — classes that, despite intense study, are not known to be learnable if the learner is given completely-specified tuples. We then show how these algorithms can be extended to incrementally modify a given initial decision tree or DNF formula, and finally extend our model to handle various types of "corruption" in the blocking process (so that a missing value is not a reliable indicator of irrelevance), as well as noise.

A Proofs

Many of the following proofs use the following well-known form of the *Chernoff bound* [Che52]:

Proposition. Given m Bernoulli trials with probability of success p, the number of successes S satisfies:

$$\begin{aligned} Pr(\ S \geq (1+\gamma)mp\ ) &< e^{-(mp\gamma^2/3)} \\ Pr(\ S \leq (1-\gamma)mp\ ) &< e^{-(mp\gamma^2/2)} \end{aligned}$$

for any $0 \leq \gamma \leq 1$. ∎

We begin by proving the result that decision trees can be learned under the $B^{(DT)}$ model.

Proof of Theorem 1: Given that samples were in fact generated from some tree d_t, and then blocked by β_{d_t} (*i.e.*, blocked according to the $B^{(DT)}$ model), we can make the following claim about BUILD-DT: BUILD-DT terminates having constructed a tree d of size at most $|d_t|$, that classifies all the examples in S correctly. Correctness (given termination) is obvious by the nature of the algorithm. For termination, note that the only possible obstacle to termination occurs when BUILD-DT must find a variable x_i that is unblocked in a (sub-)sample S. But BUILD-DT recursively constructs subsamples by grouping samples that *agree* on values of some common variables. The samples in S correspond to paths in the d_t, and there must have been some node in the d_t at which they began to diverge from each other. (Since we know some samples in S are distinct, they must diverge somewhere.) The variable labeling the first such node must appear in all samples in S, and so would be a suitable candidate for x_i.

Next we prove the size bound. BUILD-DT has a (potentially) nondeterministic step when it chooses a variable that is unblocked in every sample. It is easy to see that BUILD-DT can in fact always construct a subtree of d_t (although to do this it would need to make the right choices). However, we now show that the size of the tree BUILD-DT constructs is independent of the choices it makes, and from these two claims it follows that the constructed tree is never larger than $|d_t|$. The proof of the second claim is by induction on the number of variables. If $n = 1$ the result is immediate. Now consider a set of samples S over $n+1$ variables, and suppose x_i and x_j are two of the variables that BUILD-DT can choose for the root (which means that they each appear unblocked in all the samples in S). Let d_i be the tree BUILD-DT would construct if it were to choose x_i as the root. When BUILD-DT is constructing the left and right subtrees of d_i, it may again have to choose among possible root variables. But x_j will be among the candidate roots for both subtrees (because the subtrees are constructed using subsets of S, and so x_j will be unblocked for all samples in each subset). Since these subtrees are over n variables we can appeal to the inductive hypothesis, which says that the size of the subtrees is not affected by the choice we make. We can therefore assume that x_j is chosen as the root of each subtree. That is, we can suppose that d_i consists of x_i at the root, followed by x_j at both level 2 nodes in the tree. However, we can argue analogously for the tree d_j which has x_j as the root: we can assume that x_i is always tested immediately after x_j. But now if we compare the four 3rd-level subtrees of d_i and d_j we see that they are both constructed by partitioning S according to the values of x_i and x_j and then recursively calling BUILD-DT. Thus they should in fact be the same in both trees (although in a different order). Thus the claim is true for $n+1$ and so the inductive argument is complete.

The second part of the proof concerns the sample size needed. Note that if a tree t has error more than ϵ, the probability that t correctly classifies all of m samples is less than $(1-\epsilon)^m$. Let $\kappa_s(\epsilon)$ be the number of trees (of size s) that have at least ϵ error, and note that $\kappa_s(\epsilon)$ is bounded by the total number of distinct trees of size s, which is at most $(8n)^s$ [EH89]. Then the probability that *any* tree with error more than ϵ will be entirely correct on m examples is at most

$$\kappa_s(\epsilon) \times (1-\epsilon)^m \quad \leq \quad (8n)^s \times (1-\epsilon)^m \quad \leq \quad (8n)^s e^{-\epsilon m}.$$

This is smaller than δ if

$$m \quad \geq \quad (s \ln(8n) - \ln \delta)/\epsilon \tag{2}$$

Any tree of size s or less that is completely correct on this many samples is thus likely to have error at most ϵ. Since the tree constructed by BUILD-DT meets this description, we are done. The mistake bound is immediate, because in fact each example we see corresponds to a leaf in the target tree; thus we see at most s distinct examples. It follows from the construction of BUILD-DT that it cannot repeat a mistake.

We close by noting that a different analysis of the sample size can be given, which uses the representation of a s-leaf decision tree as an s-term DNF formula and Theorem 2. The latter result, proved below, shows that we can expect to see "enough" positive samples when $m = O((s/\epsilon)\ln(s/\delta))$. This bound may be a better or worse bound than Equation 2, depending on the size of s relative to n. This approach also shows that we can learn decision trees using only positive samples. ∎

Before giving the proofs concerning DNF formulae, it is useful to establish some notation and a useful lemma. We will use the convention that a conjunctive term over n variables can be identified with a vector in $\{0,1,*\}^n$, in the natural way: if $x_i \in t$ the vector's component is 1, $\bar{x}_i$ becomes 0, and otherwise the component is $*$. For example, if $n=5$ we identify $x_1 \wedge \bar{x}_3 \wedge x_4$ with $\langle 1,*,0,1,*\rangle$. There is a natural partial ordering over $\{0,1,*\}$ according to specificity: $t \prec t'$ if t' is identical to t except that some components that are 0 or 1 in t may be $*$ in t'. If $t \prec t'$ we say that t *subsumes* t'.

Recall that we write the s-term DNF formula φ to be learned as $\varphi = t_1 \vee \cdots \vee t_s$, and we use P to denote the (unknown) underlying distribution over domain instances (*i.e.*, over X_n). This distribution, together with the concept φ and the blocker β, induce the following two useful measures. First, for $i \leq s$, we define the *coverage* of the term t_i as $c(t_i) = P(\,\{\tilde{x} \mid \varphi(\tilde{x}) \textit{ and } \beta(\tilde{x}) = t_i\}\,)$; *i.e.*, the probability that the blocker will produce t_i from a positive instance (prior to any degradation). Second, for any term t, we define $P^{\varphi-}(t)$ to be the probability that t is true but φ is false, *i.e.*, $P(\,\{\tilde{x} \mid t(\tilde{x}) \textit{ and } \neg\varphi(\tilde{x})\}\,)$. Note that $P^{\varphi-}(t)$ is 0 for any term $t_i \in \varphi$, and also for any term subsumed by some t_i.

All our algorithms for learning DNF formulae produce a hypothesis which is itself in $\mathcal{DNF}_n$, although possibly with more than s terms. The following easy lemma gives sufficient conditions for such a hypothesis φ' to have error less than ϵ.

Lemma. Let $\varphi = t_1 \vee \cdots \vee t_s$ and $\varphi' = t'_1 \vee \cdots \vee t'_r$. Let $s' \leq s$ be the number of terms in φ but not in φ', and let $r' \leq r$ be the number of terms in φ' but not φ. Then $P(\,\varphi \Delta \varphi'\,) < \epsilon$ if

1. Each t_i with $c(t_i) > \epsilon/(2s')$ is included in φ', and
2. For all $i \leq r$, $P^{\varphi-}(t'_i) < \epsilon/(2r')$.

Proof: Let $U \subset \{t_1, t_2, \ldots, t_s\}$ be the s' terms in φ that are not in φ'. By Condition 1, these all have small coverage. Now if $\tilde{x}$ satisfies φ but not φ', then $\beta(\tilde{x})$ must be in U (or else $\varphi'(\tilde{x})$ would be true). Thus

$$\begin{aligned} Pr(\varphi \wedge \neg\varphi') &= \sum_{t \in U} P(\{\tilde{x} \mid \varphi(\tilde{x}) \wedge \neg\varphi'(\tilde{x}) \wedge \beta(\tilde{x}) = t\}) \\ &\leq \sum_{t \in U} c(t) \quad \leq \quad |U|\frac{\epsilon}{2s'} \quad = \quad s'\frac{\epsilon}{2s'} \quad = \quad \frac{\epsilon}{2}\,. \end{aligned}$$

Error is also possible if $\varphi'(\tilde{x})$ holds but not $\varphi(\tilde{x})$. However, using Condition 2, the probability of this occurring is bounded by

$$Pr(\neg\varphi \wedge \varphi') \quad \leq \quad \sum_{i=1}^{r} P(\neg\varphi \wedge t'_i) \quad = \quad \sum_{i=1}^{r} P^{\varphi-}(t'_i) \quad \leq \quad r'\frac{\epsilon}{2r'} \quad = \quad \frac{\epsilon}{2}\,,$$

where the penultimate step (introducing r') follows because at most r' of the terms in φ' do not occur in φ, and only these terms can have nonzero $P^{\varphi-}(t'_i)$.

Combining the two parts shows that the total probability of error is at most ϵ, as required. ▮

We use this lemma in each of the following results. With two exceptions, we always use it in a slightly weaker form with r' replaced by r and s' by s. We call any $t_i \in \varphi$ such that $c(t_i) > \epsilon/(2s)$ a *heavy* term. Note also that, even though coverage (and hence, heaviness), is defined with respect to a specific blocker, a term may well lead to a correct classification of more terms than its coverage suggests.

Recall that our first result for DNF formulae concerns the case where there is no degradation.

Proof of Theorem 2: Because there is no degradation, every positive instance we see is in fact a term of φ, and so the disjunction of these instances trivially satisfies condition 2 of the lemma.

It remains to verify that we will see, and so include, all heavy terms. If t_i is such a term, the expected number of occurrences in m samples is at least $m\epsilon/(2s)$. Using the Chernoff bounds (here with $\gamma = 1$) the probability of never seeing t_i is at most $e^{-m\epsilon/(4s)}$. There are at most s such terms t_i, and so the probability that there is any such term which fails to be seen is at most $se^{-m\epsilon/(4s)}$. This is less than δ if $m > \ln(s/\delta)4s/\epsilon$. If our sample exceeds this size, we will recover all the heavy terms with probability at least $1 - \delta$, as required.

The mistake bound is immediate. Each positive example we see corresponds to a term in the target formula, and once we see such a term we never make a mistake on it again. Thus we make at most s mistakes on positive examples. It is easy to see that LEARN-DNF never makes mistakes on negative examples. ▮

Our next results concern Theory Revision:

Proof of Theorem 3: Let s_0 be the number of terms in φ_{cor} but not φ_{init}, and r_0 count the terms in φ_{init} but not φ_{cor}. (Of course, $r_0 + s_0 = r(\varphi_{cor}, \varphi_{init})$.)

Arguing exactly as in the proof of Theorem 2, if we consider $m > \ln(2s_0/\delta)4r_0/(\alpha\epsilon)$ samples then, with probability at least $1 - \delta/2$, we can expect to see all terms in φ_{cor} that have coverage at least $\alpha\epsilon/(2s_0)$.

Now consider the (at most r_0) terms t' in φ_{init} such that $P^{\varphi_{cor}-}(t') > \alpha\epsilon/(2r_0)$. A similar Chernoff bound analysis shows that we can expect to see at least one negative example contradicting each such t' if we examine a sample of size at least $\ln(2r_0/\delta)4r_0/(\alpha\epsilon)$.

So suppose we examine a sample of size $\ln(2r(\varphi_{cor}, \varphi_{init})/\delta)4r(\varphi_{cor}, \varphi_{init})/(\alpha\epsilon)$, and apply MODIFY-DNF to produce a hypothesis φ'_{init}. From the above arguments, φ'_{init} should contain all terms from φ_{cor} with coverage more than $\alpha\epsilon/(2s_0)$, and no terms t' with $P^{\varphi_{cor}-}(t') > \alpha\epsilon/(2r_0)$. Now we can apply the lemma in its original (strong) form to φ_{cor} and φ'_{init}. Noting that r' and s' as used in the lemma necessarily satisfy $r' \le r_0$ and $s' \le s_0$, we see that both conditions of the lemma are satisfied. Thus φ'_{init} has error at most $\alpha\epsilon$, as required. ▮

Proof of Theorem 4: Recall that we can regard BUILD-DT$'$ as producing a "prefix" tree d_S built from examples S that d_{init} did not, or could not, classify correctly. There are two types of leaf nodes in d_S: those corresponding to examples in S, and "default" nodes which are replaced by a diminished version of d_{init}.

For now, ignore the default nodes (e.g., imagine that each such leaf is labeled F, rather than with a diminished tree). Let E be the event, on the original sample space X_n, that an instance cannot be classified correctly by d_{init} *after* d_{cor} *blocking*. Note that d_{init} might, in fact, be able to classify some (unblocked) instances in E correctly — but membership in E is determined by what happens after d_{cor} blocking, which might block enough attributes that d_{init} is unable to decide upon a definite classification.

The tree d_S is learned using examples that come from instances in E. In fact, d_S is a tree learned for the concept d_{cor}, except under a new domain distribution P', where P' is equal to the original distribution P conditioned on the event E. Let γ be the (unknown) probability of E. By Theorem 1, for any ϵ', d_S will have an error of at most ϵ' (with probability at least $1-\delta/2$) under P' if S has size at least $(r/\epsilon')\ln(16n/\delta)$. (We can use r, rather than s, here because there are at most r leaves of d_{cor} that are relevant to examples in E.) It is easy to see that the labeling of default nodes by diminished trees rather than by F does not affect this result. For S to contain this many samples, we should expect to have to draw about $1/\gamma$ times as many samples from P; a simple Chernoff bound analysis shows that $(2r/(\epsilon'\gamma))\ln(16n/\delta)$ suffices (for $r>3$) to guarantee this with probability at least $1-\delta/2$. Thus, after this many samples, we expect d_S to have error at most ϵ' under P'. So the probability that an instance comes from E *and* that d_S misclassifies this instance, under the original distribution P, is at most $\epsilon'\gamma$.

Now notice that if an instance does not come from E then d_{init} classifies it correctly. (Proof: As non-default leaves correspond to samples in E, each such instance will be classified using a default leaf in d_S. Furthermore, anything classified by a default leaf is given the same label that d_{init} itself would give. Thus, instances that are not in E are always classified correctly by d_S.) It follows that $\epsilon'\gamma$ in fact bounds the total error, under P, over all of X_n. The result follows by substituting $\epsilon'=\alpha\epsilon/\gamma$ in the bounds of the previous paragraph.

We close by bounding the size of d_{out}. Each path in d_S (d_{out}'s prefix) corresponds to a path in d_{cor}, and so there are at most $s=|d_{cor}|$ of them. Some of these d_S leaves are then replaced by diminished versions of d_{init}, which have at most $|d_{init}|\leq s+r$ leaves. Thus, the resulting d_{out} tree has at most $s(s+r)$ leaves. ▮

Our remaining results all deal with various types of degradation. When there is ν-degradation with $\nu=O(\ln n/n)$, we may need a sample size larger than was needed for $\nu=0$, to be sure of seeing all the heavy terms. However, this is essentially the only change required to the proof of Theorem 2.

Proof of Theorem 6: Consider any heavy term t_i. By definition, the blocker will produce t_i with probability at least $\epsilon/2s$. However, t_i may not survive degradation intact. Suppose $\nu<\min\{k\ln n/n, 0.5\}$. If t_i contains $l\leq n$ $*$'s, the probability that none of these $*$'s degrade is at least $(1-\nu)^l$, which is more than $(1-k\ln n/n)^n$. The extremely conservative lower bound, $(1-\lambda/n)^n>e^{-2\lambda}$ for any $\lambda<1/2$, yields a lower bound of $1/n^{2k}$ for any $n>1$. (As $(1-k\ln n/n)^n\approx e^{-k\ln n}=1/n^k$ for large enough n, the exponent can in practice be reduced to simply k.)

Thus, the probability of seeing t_i is at least $\epsilon/(2sn^{2k})$. The Chernoff bound argument used in Theorem 2 now applies, but using this probability rather than $\epsilon/(2s)$. It follows that a sample size of $m>(\ln s/\delta)4sn^{2k}/\epsilon$ is enough so that, with probability at least $1-\delta$, we see every heavy term undegraded at least once.

Because of the ν-degradation, we also see terms other than the t_i. But since any such term t is subsumed by some $t_i\in\varphi$, we know that $P^{\varphi-}(t)=0$ and thus such terms are harmless. If we wish to obtain a shorter hypothesis we can discard any subsumed terms. However, even after removing subsumed terms, the result is not necessary a subset of the terms in φ. It is true that any degradation of a heavy term t_i will be removed, because t_i itself will be present. However, we may see degradations of non-heavy terms without seeing the (undegraded) terms themselves, and these degradations cannot be filtered by subsumption. ∎

Our next result permits μ-degradation of at most $O(\ln n/n)$, but no ν-degradation. This case is not simply a symmetric variant of the previous result, as μ-degradation can produce such terms t for which $P^{\varphi-}(t)$ can be large. (By contrast, the terms produced by ν-degradation are subsumed by some $t_i \in \varphi$ and so are harmless, as $P^{\varphi-}(t) = 0$.) We must therefore identify and remove such degraded terms.

Proof of Theorem 7: Recall that Algorithm LEARN-DNF$^{\mu}$ keeps all terms that occur more than $m\epsilon/(4sn^{2k})$ times in a sample of size m, and which furthermore do not subsume any other instance in the sample. We can, without loss of generality, assume that the DNF formula φ is irredundant in the sense that for no t_i, t_j do we have $t_i \prec t_j$ (for otherwise, we could remove t_j). It follows that if we see any t_i from φ, it can never subsume any other instance in the sample.

The algorithm will be correct if we choose the sample size m large enough to ensure that (with probability at least $1 - \delta/3$ each) the following three conditions hold:

First, m must be large enough so that all heavy terms are seen more than $m\epsilon/(4sn^{2k})$ times, where $\mu < k \ln n/n$. By the argument in the first paragraph, if we do see a heavy term it will never subsume any other instance, and so this will be enough to show that all the heavy terms are included in our hypothesis. But, arguing as in the proof of Theorem 6, the expected number of undegraded occurrences of a heavy term is at least $m\epsilon/(2sn^{2k})$. Using Chernoff bounds, the probability of seeing the term fewer than half this many times, is at most $e^{-m\epsilon/(16sn^{2k})}$. So it suffices to have $m > \ln(3s/\delta)16sn^{2k}/\epsilon$.

Our second requirement is that each term t_i such that $c(t_i) > \epsilon/(8s^2n^{2k})$ should be seen at least once. Using Chernoff bounds as in Theorem 2, it suffices that $m > \ln(3s/\delta)16s^2n^{2k}$. This means that any term which is a degraded version of such a t_i will not be included in our hypothesis, no matter how many times we see it, because it subsumes t_i and we expect to see t_i at least once.

Thus the only terms which might be incorrectly included in our hypothesis are degradations of terms t_i for which $c(t_i) < \epsilon/(8s^2n^{2k})$. But the blocker produces such terms very infrequently. Even if every single occurrence of such a term was degraded into the same "bad" term t_b, the expected number of occurrences of t_b will be less than $ms\epsilon/(8s^2n^{2k})$ $= m\epsilon/8sn^{2k}$. Using Chernoff bounds, we bound the probability that the actual number of occurrences due to these "very light" t_i exceeds $m\epsilon/4sn^{2k}$, by $e^{-m\epsilon/(24sn^{2k})}$. Here, then, $m > 24sn^{2k}\ln(3/\delta)/\epsilon$ suffices.

Combining the three parts of the proof, we see that if $m > \ln(3s/\delta)24s^2n^{2k}/\epsilon$, our hypothesis will almost certainly include all heavy terms, and is very unlikely to include anything else. ∎

Proof of Theorem 8: If t_i contains only $c \ln n$ terms, the probability of an instance of t_i being seen undegraded is at least $(1-\mu)^{c \ln n} = n^{c \ln(1-\mu)}$ which, for fixed μ and c, is polynomial in n. Examining the previous proof, we see that it carries through if we substitute $n^{c \ln(1-\mu)}$ (the new lower bound on a term being undegraded) for n^{2k} (the old bound) throughout. The result follows. ∎

Our final result about probabilistic degradation permits both μ and ν degradation, but requires negative examples.

Proof of Theorem 9: The proof has two steps. First, we show that we can use the positive examples to construct a "small" (*i.e.*, polynomial size) set of *candidate terms* $C = \{t'_1, t'_2, \ldots, t'_r\}$ that is very likely to include all the heavy t_i. We then show how to use negative examples, to filter out all terms with $P^{\varphi-}(t'_i) > \epsilon/2r$.

We begin with the generation of candidate terms. Recall that LEARN-DNF$^{\mu\nu}$considers all subsets of $2k$ positive examples, and for each subset constructs a candidate term using a component-wise "vote". Suppose we could be assured that some subset of $2k$ instances were all degradations of the same heavy term t_i. What is the probability that this subset "votes" for exactly t_i? Consider any variable x. Each instance gives the variable a different value from t_i with probability at most γ, where $\gamma = \max\{\mu^{(+)}, \nu^{(+)}\}$. As the correct value will win the vote unless at least half of the sample votes for something other than the correct value, we need to bound the probability that k or more of the $2k$ instances give the wrong value, which is at most $\binom{2k}{k}\gamma^k < (4\gamma)^k$ (using the observation that the number of subsets of size k, $\binom{2k}{k}$, is under the total number of subsets, 2^{2k}). Hence, the probability that *any* attribute will be given the wrong value is at most $n(4\gamma)^k$, which is less than $1/2$ if $\gamma < (1/2n)^{1/k}/4$. For this, $\gamma < n^{-1/k}/8$ suffices, which is an assumption of the theorem.

So if we were to have d disjoint size-$2k$ subsamples that come from t_i, the probability that none of these vote for t_i is less than $1/2^d$. If we have d such subsets for each heavy t_i, the probability that any term is not voted for at least once is at most $s/2^d$ (as there are at most s such heavy terms). This is less than $\delta/4$ if

$$d \quad \geq \quad \left(\ln \frac{4s}{\delta}\right) / \ln(2). \tag{3}$$

(The entire proof will require 4 distinct conditions to hold, which is why we ensure that each fails with probability at most $\delta/4$, rather than just δ.)

We can thus be confident that C contains each heavy t_i so long as we get to see at least $2kd$ (degraded) examples of each such term. An easy Chernoff bound argument shows that $m^+ \geq \max\{(8 \ln(4s/\delta),\ 8kd)\}/\epsilon$ suffices, which (after substituting the Equation 3 bound on d) holds if $m^+ > \ln(4s/\delta) 8ks/\epsilon$. Of course, we would generally not wish to use many more examples than this, because the number of candidates grows as $(m^+)^{2k}$.

The second part of the proof uses a general technique which shows that if $\mu^{(-)} < \epsilon/(2r)$, then using $O(\ln(4r/\delta) r/\epsilon)$ negative examples one can (with probability at least $1 - \delta/4$) filter any set of r candidate terms so as to ensure property 2 of the lemma, and yet not discard any terms $t_i \in \varphi$ among the candidates.

First note that if $t_i \in \varphi$, then every example t^- produced by the negative blocker (before degradation) will contradict t_i. That is, t^- and t_i disagree on the value (0 or 1) of at least one variable, x say. However t^- might be degraded, to t' say. But t' and t_i will still be contradictory unless x was degraded to $*$ in t'. This happens with probability at most $\mu^{(-)}$. It follows that the number of negative examples we see that do not contradict t_i is expected to be at most $\mu^{(-)} m^-$ where m^- is the number of negative samples.

Now consider a term $t_b \notin \varphi$ such that $P^{\varphi-}(t_b) > \epsilon/(2r)$. By the definition of $P^{\varphi-}(\cdot)$, this means that, with probability at least $\epsilon/(2r)$, a negative instance is consistent with t_b. Clearly neither blocking nor degradation can make such an example inconsistent with t_b. Thus, we expect to see at least $\epsilon m^-/(2r)$ negative examples consistent with t_b.

Recall that our algorithm considers m^- negative examples and keeps only those candidate terms that are consistent with fewer than $\epsilon m^-/(4r)$ of them. Using Chernoff bounds and $\mu^{(-)} < \epsilon/(8r)$, the probability of a term t_i being incorrectly rejected is at most $e^{\epsilon m^-/(24r)}$, and if $c(t_b) > \epsilon/(2r)$ the probability of t_b being kept is at most $e^{\epsilon m^-/(16r)}$. Thus, it suffices to have $m^- > \frac{24r}{\epsilon} \ln \frac{4r}{\delta}$.

Note that, using the positive sample size m^+ suggested above, $r < (\ln(4s/\delta)8ks/\epsilon)^{2k}$ and so (very conservatively) m^- can be $24(\log 1/\delta)(\ln(4s/\delta)8ks/\epsilon)^{2k+1}$.

The final question concerns whether we can expect to see enough of both positive and negative examples. But if negative examples occur less than ϵ often, then "true" (*i.e.*, a tautological concept) will be an acceptable hypothesis. Similarly, if positive examples are rare, "false" can be used. Otherwise, if both positive and negative examples have rates more than ϵ, we expect to see at least $m\epsilon$ positive instances, and $m\epsilon$ negative instances, in a sample of size m. It it is easy to show that we will (with probability at least $1 - \delta/4$) see enough instances of both positive and negative samples if $m > 2m^-/\epsilon$. The result follows. ▮

Proof of Theorem 11: First, consider $\gamma_k^{(A,+inst)}$ k/positive-instance degradation for any constant k. Note that any observed positive example can be an adversarially degraded form of one of only $2^k\binom{n}{k} = \text{poly}(n)$ "uncorrupted" terms. (These *uncorrupted* terms may have (μ, ν) attribute degradation, but no adversarial degradation.) Thus, given any polynomial sample of positive instances, we can construct another larger sample (but still of polynomial size) that is sure to contain all the uncorrupted terms from which the examples were derived. If we use this larger sample in place of the original positive sample, it is easy to verify that the proof of Theorem 9 continues to hold, with few changes. In particular, we will be able to reconstruct all the original terms by voting, and then can use negative examples to filter out all the unwanted terms.

The case of $\gamma_k^{(A,-inst)}$ k/negative-instance degradation in the presence of $(0, O(\ln n/n), 1, 1)$ attribute degradation is an immediate corollary of Theorem 6, because the latter result never used negative instances anyway. ▮

Proof of Theorem 13: We begin with the case where there is no ν-degradation. The algorithm we use is to collect m_k samples and report the disjunction of all terms that appear, with either label, k or more times and which are labeled T more often than they are labeled F. We specify below the appropriate values of m_k and k.

To begin, we should choose m_k large enough so that we expect to see each heavy term at least k times. A fairly routine Chernoff bound analysis shows that

$$m_k = \frac{2s}{\epsilon}\left(2k + 8\ln\frac{2s}{\delta}\right) \tag{4}$$

suffices. (The proof is to show that given this sample size we: (1) expect to see each heavy term $2k$ or more times, and (2) will almost certainly see any heavy term at least half its expectation.)

Now suppose our sample includes $k' \geq k$ occurrences of a term t. We expect the term to be classified correctly at least $(1-\alpha)k'$ times. Using Chernoff bounds, with $\gamma = (1-2\alpha)/(2-2\alpha)$, the probability that the sample actually includes $k'/2$ or more misclassifications is at most $e^{-k'(1-2\alpha)^2/(8(1-\alpha))}$.

In a sample of size m_k there may be many samples occurring k or more times, but a (very!) conservative bound is 3^n, the number of terms. (Using the apparently much tighter bound of m_k/k leads to a substantially more complex analysis, for little ultimate gain.) The probability that any of these is misclassified by more than half its instances is thus at most $3^n \exp\left(-\frac{k(1-2\alpha)^2}{8(1-\alpha)}\right)$, which is less than $\delta/2$ if

$$k \quad \geq \quad \frac{8}{(1-2\alpha)^2}\left(1.1n \; + \; \ln\frac{2}{\delta}\right) \tag{5}$$

Hence, if we examine $O(\frac{sn}{\epsilon(1-2\alpha)^2} \ln\frac{2s}{\delta})$ samples, we can expect to see all heavy terms k or more times, each of these terms will be labeled T more often than not, and no negative terms will mistakenly be identified as being positive. The result follows.

The case for $\nu = O(\ln n/n)$ degradation uses an extremely similar argument; see the proof of Theorem 6 for the required modifications. ▮

Proof of Theorem 14: PAC-learning decision trees in the presence of classification noise follows from the techniques of the previous theorem. Namely, first let $k \geq 8/(1-2\alpha)^2 \ln(2s/\delta)$ (which improves on Equation 5, using the observation that there can be at most s blocked instances in an s-leaf tree) and $m_k \geq (2s)/\epsilon \; (2k + 8\ln(2s/\delta))$ from Equation 4. Then after m_k samples, we expect to see all "heavy" leaves of d_{cor} at least k times, and to be able to correctly identify each of these leaves' classification. (Here we say that a leaf is heavy if it has coverage more than $\epsilon/(2s)$, whether or not it is labeled T or F). We can then pass *just* the leaves that occur k or more times in the original sample, together with the appropriate classification, to BUILD-DT. This will construct a tree whose only error is attributable to the non-heavy leaves that our sub-sample omitted. But the total error this can cause is at most ϵ. ▮

References

[Ang92] D. Angluin. Computational learning theory: survey and selected bibliography. In *Proc. 24th Annu. ACM Sympos. Theory Comput.*, pages 351–369. ACM Press, New York, NY, 1992.

[BFOS84] L. Breiman, J. Friedman, J. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth and Brooks, Monterey, CA, 1984.

[BHL95] Avrim Blum, Lisa Hellerstein, and Nick Littlestone. Learning in the presence of finitely or infinitely many irrelevant attributes. *J. of Comput. Syst. Sci.*, 50(1):32–40, 1995. Earlier version in 4th COLT, 1991.

[Blu92] A. Blum. Learning boolean functions in an infinite attribute space. *Machine Learning*, 9(4):373–386, October 1992.

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

[EH89] A. Ehrenfeucht and D. Haussler. Learning decision trees from random examples. *Information and Computation*, 82:231–246, 1989. First appeared in COLT 88.

[GGK96] Russell Greiner, Adam Grove, and Alex Kogan. Exploiting the omission of irrelevant data. In *Proceedings of the Thirteenth International Machine Learning Conference*, Bari, Italy, July 1996. Morgan Kaufmann.

[GM96] Sally A. Goldman and H. David Mathias. Teaching a smarter learner. *J. of Comput. Syst. Sci.*, 52(2):255–267, 1996. Earlier version in 6th COLT, 1993.

[Gre95] Russell Greiner. The complexity of theory revision. In *Proceedings of IJCAI-95*, 1995.

[GS95] Sally A. Goldman and Robert A. Sloan. Can PAC learning algorithms tolerate random attribute noise? *Algorithmica*, 14(1), July 1995.

[HKLW91] D. Haussler, M. Kearns, N. Littlestone, and M. K. Warmuth. Equivalence of models for polynomial learnability. *Inform. Comput.*, 95(2):129–161, December 1991.

[JKP94] George H. John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In *Proceedings of the Eleventh International Machine Learning Conference*, pages 121–129, N.J., 1994. Morgan Kaufmann.

[KL93] Michael Kearns and Ming Li. Learning in the presence of malicious errors. *SIAM J. Comput.*, 22:807–837, 1993.

[KLPV87] Michael Kearns, Ming Li, Leonard Pitt, and Leslie Valiant. On the learnability of boolean formulae. In *Proceedings of the 19th Symposium on the Theory of Computations*, pages 285–295, New York, May 1987.

[KMR$^+$94] M. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. Schapire, and L. Sellie. On the learnabilty of discrete districutions. In *Proceedings of Twenty-sixth ACM Symposium on Theory of Computing*, pages 273–282, 1994.

[KR93] E. Kushilevitz and D. Roth. On learning visual concepts and DNF formulae. In *Proc. 6th Annu. Workshop on Comput. Learning Theory*, pages 317–326. ACM Press, New York, NY, 1993.

[KSS92] M. J. Kearns, R. E. Schapire, and L. M. Sellie. Toward efficient agnostic leaning. In *Proceedings COLT-92*, pages 341–352. ACM Press, 1992.

[LDRG94] Pat Langley, George Drastal, R. Bharat Rao, and Russell Greiner. Theory revision in fault hierarchies. In *Proceedings of The Fifth International Workshop on Principles of Diagnosis (DX-94)*, New Paltz, NY, 1994.

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

[Lit91] N. Littlestone. Redundant noisy attributes, attribute errors, and linear threshold learning using Winnow. In *Proc. 4th Annu. Workshop on Comput. Learning Theory*, pages 147–156, San Mateo, CA, 1991. Morgan Kaufmann.

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

[MB88] S. Muggleton and W. Buntine. Machine invention of first order predicates by inverting resolution. In *Proceedings of IML-88*, pages 339–51. Morgan Kaufmann, 1988.

[Moo94] Raymond Mooney. A preliminary PAC analysis of theory revision. In T. Petsche and S. Hanson, editors, *Third Annual Workshop on Computational Learning Theory and Natural Learning Systems (CLNL-92).* MIT Press, 1994.

[MT94] Sridhar Mahadevan and Prasad Tadepalli. Quantifying prior determination knowledge using the pac learning model. *Machine Learning*, 17:69–105, 1994.

[OM90] Dirk Ourston and Raymond J. Mooney. Changing the rules: A comprehensive approach to theory refinement. In *Proceedings of AAAI-90*, pages 815–820, 1990.

[PBH90] B. W. Porter, R. Bareiss, and R. C. Holte. Concept learning and heuristic classification in weak-theory domains. *Artificial Intelligence*, 45(1–2):229–63, 1990.

[Qui92] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, San Mateo, 1992.

[RCJ88] K. Ruberg, S.M. Cornick, and K.A. James. House calls: Building and maintaining a diagnostic rule-base. In *Proceedings Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1988.

[Riv87] R. L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[Rus89] Stuart Russell. *The Use of Knowledge in Analogy and Induction.* Morgan Kaufmann, 1989. (Also, PhD thesis from Stanford University).

[SD90] Jude W. Shavlik and Thomas G. Dietterich. *Readings in Machine Learning.* Morgan Kaufmann, 1990.

[SG94] Dale Schuurmans and Russell Greiner. Learning default concepts. In *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, pages 519–523, 1994.

[SG97] Dale Schuurmans and Russell Greiner. *Learning to Classify Incomplete Examples*, chapter 6. MIT Press, 1997.

[SV88] G. Shackelford and D. Volper. Learning k-DNF with noise in the attributes. In *Proceedings COLT-88*, pages 97–103, 1988.

[Tow91] Geoff Towell. *Symbolic Knowledge and Neural Networks: Insertion, Refinement and Extraction.* PhD thesis, University of Wisconsin, Madison, 1991.

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

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

[WP93] James Wogulis and Michael J. Pazzani. A methodology for evaluating theory revision systems: Results with Audrey II. In *Proceedings of IJCAI-93*, pages 1128–1134, 1993.