Appears in the
Proceedings of the Thirteenth International Conference on Machine Learning (IMLC-96),
Bari Italy, July 1996.

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; Newark, NJ 07102 /
RUTCOR; 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 *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 also show that this model can be extended 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.

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 sufficient to classify this sample. After Section 2 presents our formal model, Section 3 shows how easy it is to then learn arbitrary DNF formulae (as well as decision trees) within this framework. By contrast, neither of these two well-studied classes is known to be learnable using completely-specified training samples. It 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 extended paper [GGK96] presents the proofs of all theorems, and other related results including a more comprehensive treatment of decision trees. 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 precisely 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 $O(\log n)$ tests; in this case there can be as many as $n - O(\log n)$ test values that 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 which are designed to handle incomplete information in the samples (*cf.*, [BFOS84, Qui92, LR87]), they all appear to be based on a different model [SG96, SG94]. In this model, 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*. Note that 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. Hence, we do not lose much information if an attribute is blocked, as its value is not needed for the classification. In fact, we can learn a lot by knowing that an attribute is irrelevant. Note that this notion of relevance is not absolute: an attribute may be irrelevant in the context of certain other attribute values, and relevant in other cases.

The extended paper [GGK96] connects our model with other notions of "(ir)relevance" [JKP94, Lit88] and "teachers" [GM93], and differentiates it from learnability models which either omit the class label [AS91, FGMP94], or which change some attribute values [SV88, Lit91, GS95].

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 $\varphi : X_n \mapsto \{T, F\}$, where $\tilde{x} \in X_n$ 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 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 $\tilde{x}$; *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 that allows L to draw random labeled examples $\langle \tilde{x}, \varphi(\tilde{x}) \rangle$ according to the distribution P and labeled by the target (unknown) concept $\varphi \in \mathcal{C}$. L's output is a hypothesis $h \in 2^{X_n}$ (which may or may not be in $\mathcal{C}$). 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, each labeled instance $\langle \tilde{x}, \varphi(\tilde{x}) \rangle$ is passed "as is" to the classifier. In our model, however, the learner instead only sees a "blocked version" of $\langle \tilde{x}, \varphi(\tilde{x}) \rangle$, written "$\langle \beta(\tilde{x}), \varphi(\tilde{x}) \rangle$". A blocker $\beta : X_n \rightarrow \{0,1,*\}^n$ replaces certain attribute values by the "blocked" (or in our case, "don't

[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.

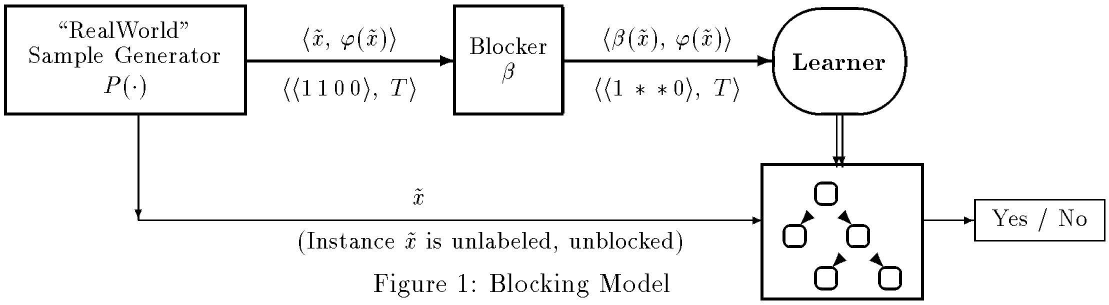

Figure 1: Blocking Model

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 either $\langle \langle 1,0,0,1 \rangle, T \rangle$, 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 $\{\, x_1 \mapsto v_1,\ \ldots,\ x_m \mapsto v_m \,\}$ (where each $v_i \in \{0,1\}$ is a specified value) 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 of 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., if $\beta(\langle v_1, \ldots, v_n \rangle) = \langle v_1, \ldots, v_m, *, \ldots, * \rangle$ 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. According to this definition, a "blocker" which never blocks any attributes at all qualifies. However, as we see later, the interesting cases arise where most or all of the superfluous attributes are in fact 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. Section 3 extends this idea to general DNF formulae.

Performance Criterion: We use the standard "Probably Approximately Correct" (PAC) criterion [Val84, KLPV87] to specify the desired performance of our learners. For any hypothesis h, let $Err(h) = P(\tilde{x} : \varphi(\tilde{x}) \neq h(\tilde{x}))$ be the probability that h will misclassify an instance $\tilde{x}$ drawn from P. Then,

Definition 2 (PAC-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, 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.,*

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

In this definition, $|\varphi|$ is the "size" of the concept φ (defined below), and the probability "$P(Err(h) < \epsilon)$" is taken over the space of all samples of the appropriate size, which is simply the product distribution induced by $P(\tilde{x})$. Note also 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 though it was trained on blocked values. This is reasonable as we are assuming that, after learning, the classifier (read "doctor") will be in a position to specify the tests to be performed.

Notation: 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 $\varphi \in \mathcal{DNF}_n$, we let $|\varphi|$ be the number of terms in φ. Let k-$\mathcal{DNF}_n$ be the class of DNF formulae whose terms include at most k literals. In particular, each member of $O(\log n)$-$\mathcal{DNF}_n$ is a disjunction of terms, each of which has only $O(\log n)$ literals.

[2] This is a slight abuse of notation, as β may be stochastic. This caveat also applies to Definition 1.

3 RESULTS: LEARNING DNFs

For any target DNF formula $\varphi = t_1 \vee \cdots \vee t_s$, the β_φ blocker does the following:

- 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 $t_j(\tilde{x}) = 1$).
- 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.

For example, the β_{φ_0} blocker for the formula $\varphi_0 \equiv \bar{x}_1\bar{x}_2x_3 \;\vee\; x_1x_2\bar{x}_4 \;\vee\; x_1x_4x_6$, would leave unblocked the variables of exactly one of these terms for any positive instance; so here $\beta_{\varphi_0}(\langle 1,1,1,0,0,0,0\rangle) = \langle 1,1,*,0,*,*\rangle$ (based on the second term). For negative instances, perhaps $\beta_{\varphi_0}(\langle 1,1,0,1,1,1,0\rangle) = \langle 1,*,*,1,*,0\rangle$.

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.

On the other hand, a blocker can also be deterministic. For instance, consider a blocker that imposes a specific order on the terms in the target 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 $\tilde{x}$.

3.1 THE SIMPLEST CASE

Let LEARN-DNF be the trivial algorithm that, given the relevant parameters, simply requests

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

samples, then forms a DNF by disjoining the observed positive samples. Then:

Theorem 1 *For any $\varphi \in \mathcal{DNF}_{n,s}$, any values of $\epsilon, \delta \in (0,1)$ and any distribution over instances P, under the $B^{(DNF)}$ model,* LEARN-DNF(n,s,ϵ,δ) *returns 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 $\frac{s}{\epsilon}\ln\frac{s}{\delta}$ blocked, labeled, samples, and will return a DNF formula with at most $|\varphi|$ terms. Notice* LEARN-DNF *uses only* positive *samples.*

```
Algorithm BUILD-DT( S: set_of_samples ): DT_Type
      // Builds a tree using labeled blocked samples S
  Let NT be a new tree
  If (all s∈S labeled with same label, ℓ)
                              or (S is empty) Then
    NT.LeafLabel = ℓ    (or "... = F" if |S| = 0)
    Return( NT )
  Let x_i = variable that is unblocked for all s∈S
  Let S^0 = { ⟨x̃ − {x_i/0}, ℓ⟩ } | ⟨x̃, ℓ⟩ ∈ S, x_i/0 ∈ x̃ }
      S^1 = { ⟨x̃ − {x_i/1}, ℓ⟩ } | ⟨x̃, ℓ⟩ ∈ S, x_i/1 ∈ x̃ }
      // That is, create S^0 by assembling instances in S
      // which include x_i/0 (and then project out x_i).
      // Similarly, create S^1 based on x_i/1 instances.
  Let NT.InternalNodeLabel = x_i
      NT.If0 = BUILD-DT( S^0 )
      NT.If1 = BUILD-DT( S^1 )
  Return( NT )
End BUILD-DT
```

Figure 2: BUILD-DT Algorithm

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 (aka "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 1) require a bound s on the size of the target formula, we can avoid this by using the standard technique of successively doubling estimates of s [HKLW91].[3]

Relation to Decision Trees: The correctness of LEARN-DNF immediately implies that decision trees are also trivial to learn in our model. In particular, any decision tree d can be converted into a DNF formula φ_d by disjoining the "path-conditions" leading to the all T-labeled leaf nodes, where each "path-condition" is the conjunction of the test-results performed on a

[3] *I.e.*, the more general learning algorithm LEARN-DNF′ first draws a set of samples based on the assumption that $s = 1$, and calls LEARN-DNF on this set. It accepts the learned formula has only 1 term; otherwise, LEARN-DNF′ draws a set of samples based on $s = 2$, calls LEARN-DNF, and accepts if the formula has at most 2 terms. If not, it successively tries $s = 4$, $s = 8$, $s = 16$, ..., until it succeeds.

```
Algorithm MODIFY-DNF( φ_init : DNF_Type; n,r : N; α,ε,δ ∈ (0,1) ): DNF_Type
  Draw  m_r = (4r/αε) ln (2r/δ) training (blocked, labeled) samples
    // Alternatively, if α is not known a priori, we can instead collect (4r/ε) ln (2r/δ) samples
    // that φ_init does not label correctly (i.e., either mislabels, or is unable to label)
  Let S+  = positive samples that φ_init either can't classify or classifies as F;
      S-  = negative samples that φ_init either can't classify or classifies as T.
  Let T = S+ ∪ φ_init.
    // I.e., T includes both un- and mis-classified positive examples, and the terms from φ_init.
  Remove from T any term that is consistent with some term in S-.
    // E.g., the negative example x1 x̄2 ∈ S- will remove from T the terms x1 and x̄3 x5, but not x̄1 x8.
  Return disjunction of terms in T.
End MODIFY-DNF
```

Figure 3: The MODIFY-DNF algorithm for modifying an initial DNF formula

path. The natural decision-tree blocker, β_d, for decision tree d simply returns the tests that are performed when traversing d in the process of classifying a given instance. Notice that this means β_d returns a term of φ_d when given a positive instance, and a term of $\neg\varphi_d$ when given a negative instance, which shows that β_d qualifies as a DNF-blocker for φ_d. Thus LEARN-DNF can be used, without change, to learn a DNF representation of d. However, the BUILD-DT algorithm (Figure 2) learns the decision tree directly (*i.e.*, as a tree) and does not use the DNF representation even as an intermediate step. This algorithm is correct, in the same PAC sense used in Theorem 1, whenever it is given at least $|S| \geq \frac{1}{\epsilon}\left(s\ln(8n) + \ln\frac{1}{\delta}\right)$ samples. (Here, s bounds the number of leaves in the decision tree.) Thus, this algorithm can be significantly more efficient than LEARN-DNF if $s \gg n$; see [GGK96] for more discussion.

3.2 THEORY REVISION

In many situations, we may already have an initial DNF φ_{init} 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 where the TR designates "Theory Revision", corresponding to the many existing systems that perform essentially the same task, albeit in the framework of Horn-clause based reasoning systems; *cf.*, [WP93, MB88, OM90, LDRG94].

There are several obvious advantages to theory revision over the "grow from scratch" approach discussed in the previous section. First, notice from Theorem 1 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 produce an adequate theory, much less the optimal one. We show below that this same set of samples may, however, be sufficient to specify how to improve a given initial theory. Another advantage of the theory revision approach is that it facilitates "learning while doing". That is, we could use the initial $\varphi_0 = \varphi_{init}$ to classify (unlabeled) instances as they are seen. If φ_0's label is found to be incorrect, we then consult an expert who provides the correct label, possibly after asking for (and receiving) the values of the critical attributes that φ_0 had inappropriately judged to be superfluous. The learner can use this information to form a new formula, call it φ_1, which it then uses as its classifier. The theory revision process then iterates: consulting the expert if φ_1 produces an incorrect label, and using that new information to produce a newer, better φ_2, and so forth. Here, the total number of expert consultations could be much less than if we began with an empty tree.

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 $\varphi_{init}\,\Delta\,\varphi_{cor}$ is the set of instances on which φ_{init} and φ_{cor} disagree; that is, it is the symmetric set difference between instances satisfied by the two concepts. 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:

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

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}.[4] Here, we assume that each complete sample is drawn at random from a stationary distribution, then

[4] 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}. We will often view a DNF formula as a set of its terms.

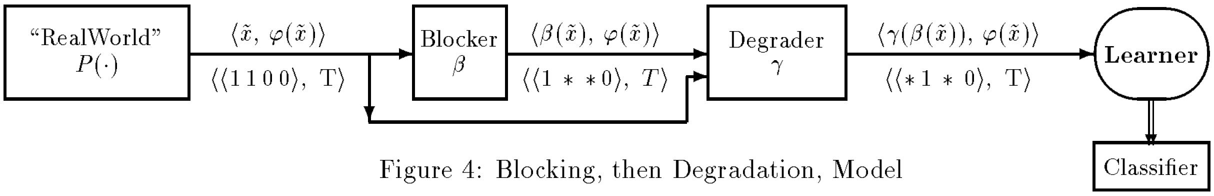

Figure 4: Blocking, then Degradation, Model

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 3:

Theorem 2 *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 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.*

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 do not know r in advance, we can use the standard iterative doubling technique to find an appropriate value. But in this case, 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 is an advantage if, for example, r is in fact exponentially larger than s. This allows us to avoid wasting many samples "unlearning" φ_{init}.

The formula φ' returned by MODIFY-DNF has the property that it agrees with each training sample seen; *i.e.*, if it used the sample $\langle\tilde{x}, T\rangle$, then $\varphi'(\tilde{x}) = T$, and if it used $\langle\tilde{x}, F\rangle$, then $\varphi'(\tilde{x}) = F$.

Finally, notice that Theorem 2 assumes that each instance is blocked by the blocker $\beta_{\varphi_{cor}}$. [GGK96] motivates and discusses other meaningful blocking models, and also connects this theory revision idea with the notions of "on-line algorithms" and "anytime algorithms" [DB88]. Finally, it also presents algorithms for modifying decision trees.

3.3 TRAINING-DATA DEGRADATION

So far our $B^{(DNF)}$ model has assumed that the blocker removes all-and-only the superfluous attribute values. There may, however, be cases 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. 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.

3.3.1 Attribute Degradation

The $\gamma^{AD}_{\vec{\mu},\vec{\nu}}$ degrader randomly degrades each blocked training instance, on an attribute-by-attribute basis: If the blocker passes unblocked the attribute x_i (*i.e.*, views x_i as relevant), then the $\gamma^{AD}_{\vec{\mu},\vec{\nu}}$ degrader will, with probability μ_i, reset 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.*, views x_i as irrelevant), the $\gamma^{AD}_{\vec{\mu},\vec{\nu}}$ degrader will, with probability ν_i, reset x_i to its 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. It is important to realize that this degradation process never changes 1 to 0 or 0 to 1; the only "noise" it introduces concerns whether an attribute is relevant or not.

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

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

Proposition 3

1. *Learning $\mathcal{DNF}_{n,s}$ with $(0,1)$ attribute degradation is as hard as learning $\mathcal{DNF}_{n,s}$ in the stan-*

```
Algorithm LEARN-DNF^{μν}( n, s, k : N; ε, δ ∈ (0,1) ): DNF_Type
  Draw m_{μν} = (48/ε)(log 1/δ)((8ks/ε) ln (4s/δ))^{2k+1} training (blocked, degraded and labeled) samples
  Let S^+ be the first at most (8ks/ε) ln (4s/δ) positive samples seen.
  Let S^- be all the negative samples seen.
  Let C be set of all size-2k subsets of S^+.
  Initialize T = {}.
  For each c = {c^(1), ..., c^(2k)} ∈ C:
          // Propose a candidate t = ⟨t_1, ..., t_n⟩ by component-wise voting
    For each i = 1...n
          { num_i(c,+) = |{c^(j) : c_i^(j) = "+"}| }
      Let { num_i(c,-) = |{c^(j) : c_i^(j) = "-"}| }
          { num_i(c,*) = |{c^(j) : c_i^(j) = "*"}| }
      Let t_i = argmax{ num_i(c,χ) : χ ∈ {1,0,*} }
          // Decide whether to accept term t = ⟨t_1, ..., t_n⟩
    If t contradicts at least ε|S^-| / (4|C|) of the terms in S^-,
      Add t to T.
  Return the disjunction of terms in T.
End LEARN-DNF^{μν}
```

Figure 5: The LEARN-DNF$^{\mu\nu}$ algorithm for $(1/8n^{1/k},\ 1/8n^{1/k},\ \epsilon/(8m_{\mu\nu}),\ 1)$ degradation.

dard model (with complete attributes).

2. *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.*
3. *It is impossible to learn $\mathcal{DNF}_{n,s}$ with $(1, 0)$ attribute degradation.*

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, alone. In each case, given this degradation we can expect to see the important terms in the DNF formula at least once, and identify them as such (rather than as degraded terms). Below, we consider μ and ν less than $\min\{k\frac{\ln n}{n},\ 0.5\}$ for some *constant* k. (The actual algorithms appear in [GGK96].)

Theorem 4 *It is possible to PAC-learn $\mathcal{DNF}_{n,s}$ with $(0, k\frac{\ln n}{n})$ attribute degradation (resp., $(k\frac{\ln n}{n},\ 0)$ attribute degradation), using $O(\frac{sn^{2k}}{\epsilon}\ln\frac{s}{\delta})$ examples (resp., $O(\frac{s^2n^{2k}}{\epsilon}\ln\frac{s}{\delta})$ examples), and using only positive examples.*
Moreover, it is possible to 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.

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 $\gamma^{AD'}_{\vec{\mu}^{(+)},\vec{\nu}^{(+)},\vec{\mu}^{(-)},\vec{\nu}^{(-)}}$ 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 $x_3 = *$ 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.

Here, this result shows that, if we receive both positive and negative examples, 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. However, we can recover terms by collecting all $\binom{m}{2k}$ subsets of size $2k$ positive samples, and "voting" (see [KMR+94]). That is, we construct a term from each size-$2k$ 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. Even if a term is unlikely to ever appear without any degradation, this voting procedure can recover it. Of course, the procedure will also generate terms that should not be included in the learner's hypothesis; we use the negative examples to filter out these inappropriate terms. 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 5.

Theorem 5
The LEARN-DNF$^{\mu\nu}$ algorithm (Figure 5) can 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.*

3.3.2 Adversarial Degradation

The previous section considered probabilistic degradation, of the sort that might arise from a noisy communication channel. A different model regards 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.

The $\gamma_k^{(A,\,\pm inst)}$ adversary can change up to k attributes in each instance, for some constant k. That is, for each such attribute, the adversary can reveal any of 0, 1, or $*$, no matter what the actual value is, or whether it was originally blocked. Unfortunately:

Proposition 6 *It is impossible to learn* $\mathcal{DNF}_{n,s}$ *with adversarial degradation* $\gamma_1^{(A,\,\pm inst)}$.

The less powerful $\gamma_k^{(A,\,+inst)}$ (resp., $\gamma_k^{(A,\,-inst)}$) degrader can only degrade *positive* (resp., negative) samples.

Theorem 7 *It is possible to learn* $\mathcal{DNF}_{n,s}$ *under* $\gamma_k^{(A,\,+inst)}$ *degradation, for any constant* k*. This holds even if there is also* $(n^{-\frac{1}{k}}/8,\ n^{-\frac{1}{k}}/8,\ \epsilon/(8m_{\mu\nu}),\ 1)$ *attribute degradation.*

It is possible to learn $\mathcal{DNF}_{n,s}$ *under* $\gamma_k^{(A,\,-inst)}$ *degradation, 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 $\langle x_1x_2,\ F\rangle$.) Under $\gamma_\eta^{(A,\,+samp)}$ (resp., $\gamma_\eta^{(A,\,-samp)}$) degradation, the adversary has the power to change positive (resp., negative) examples in this fashion.

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

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

The extended paper [GGK96] also proves it is possible to learn $\mathcal{DNF}_{n,s}$ with:

- classification noise of $\alpha < 0.5$ (*i.e.*, stochastically changing the label of an instance with probability bounded by α),
- attribute noise of $\rho < n^{-\frac{1}{k}}/16$ (*i.e.*, stochastically changing the value of an attribute with some probability bounded by ρ).

It also discusses when we can learn in the presence of combinations of various degradation processes.

4 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 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 DNF formula, and finally extend our model to handle various types of "corruption" in the blocking process (so that a missing value is an unreliable indicator of irrelevance), as well as noise processes.

Acknowledgments

Note that this is an extended version of the paper, "Dealing with (Intentionally) Omitted Data: Exploiting Relative Irrelevancies", which appears in working notes of the *1994 AAAI Fall Symposium on "Relevance"*, New Orleans, November 1994.

We gratefully acknowledge receiving helpful comments from R. Bharat Rao, Tom Hancock, Dan Roth, Dale Schuurmans and George Drastal.

References

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

[AS91] D. Angluin and D. Slonim. Learning monotone DNF with an incomplete membership oracle. In *COLT-91*, p. 139–146, 1991.

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

[DB88] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *AAAI-88*, p. 49–54, August 1988.

[FGMP94] M. Frazier, S. Goldman, N. Mishra, and L. Pitt. Learning from a consistently ignorant teacher. In *COLT-94*, p. 328–339. ACM, New York, 1994.

[GGK96] Russell Greiner, Adam Grove, and Alex Kogan. Knowing what doesn't matter. Tech. report, Siemens Corp. Res., 1996. `ftp://scr.siemens.com/pub/learning/Papers/greiner/superfluous-journal.ps`

[GM93] S. Goldman and D. Mathias. Teaching a smarter learner. In *COLT-93*, p. 67–76. ACM, New York, 1993.

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

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

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

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

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

[KMR+94] M. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. Schapire, and L. Sellie. On the learnability of discrete distributions. In *STOC*, p. 273–282, 1994.

[KR93] E. Kushilevitz and D. Roth. On learning visual concepts and DNF formulae. In *COLT-93*, p. 317–326. New York, 1993.

[LDRG94] P. Langley, G. Drastal, B. Rao, and R. Greiner. Theory revision in fault hierarchies. In *Workshop on Principles of Diagnosis*, New Paltz, 1994.

[Lit88] N. 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 *COLT-91*, p. 147–156, 1991.

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

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

[OM90] D. Ourston and R. Mooney. Changing the rules: A comprehensive approach to theory refinement. In *AAAI-90*, p. 815–820, 1990.

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

[Qui92] J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1992.

[RCJ88] K. Ruberg, S. Cornick, and K.A. James. House calls: Building and maintaining a diagnostic rule-base. In *3rd Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1988.

[SG96] D. Schuurmans and R. Greiner. Learning to classify incomplete examples. In *Computational Learning Theory and 'Natural' Learning Systems, IV*, MIT, 1996.

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

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

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

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

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