Knowing What Doesn't Matter:
Exploiting Omitted Superfluous Data*

Russell Greiner, Thomas Hancock, and R. Bharat Rao†
Learning Systems Department, Siemens Corporate Research
755 College Road East, Princeton, NJ 08540-6632
E-mail: {greiner, hancock, bharat}@scr.siemens.com
Phone: (609)-734-6500, *Fax:* (609) 734-6565

Abstract

Most inductive inference algorithms (*i.e.*, "learners") 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 *superfluous* 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 showing that certain classes that seem hard to learn in the general PAC model — *viz.*, decision trees — are trivial to learn in this setting, and can even be learned in a manner that is very robust to classification noise. We also discuss how this model can be extended to deal with (1) theory revision (*i.e.*, modifying an existing decision tree); (2) "complex" attributes (which correspond to combinations of other atomic attributes); (3) blockers that occasionally include superfluous values or exclude required values; and (4) other hypothesis classes (*e.g.*, DNF formulae).

Keywords: machine learning, decision trees, diagnosis, theory refinement

Declaration: This paper has not already been accepted by and is not currently under review for a journal or another conference, nor will it be submitted for such during IJCAI's review period.

---

*This is an extended version of a paper that appeared in working notes of the *1994 AAAI Fall Symposium on "Relevance"*, New Orleans, November 1994.

†Authors listed alphabetically. We gratefully acknowledge receiving helpful comments from Dale Schuurmans and George Drastal.

1 Introduction

A diagnostician typically performs only a small fraction of all possible tests; furthermore, the choice of which tests are performed depends on the results of the tests performed earlier in the diagnosis session. As an example: Knowing that a positive t_1 blood test is sufficient to establish that a patient has `diseaseX`, a doctor can conclude that a patient has `diseaseX` after performing only t_1, if that test is positive. In recording his findings, the doctor will only record the result of this one test $\langle t_1, + \rangle$ and the diagnosis `diseaseX`. *N.b.*, the doctor does not know, and, therefore, will not record, whether the patient has symptoms corresponding to tests $t_2, t_3, \ldots, t_n$.

A learning program (read "a medical student") 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 t_2 through t_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 encode the fact that the doctor's classifier (which the learner is trying to learn) can establish `diseaseX` and terminate on observing only that t_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. Section 2 presents the formal framework and Section 3 provides our results for learning decision trees in this framework. Section 4 discusses four extensions to this model, that deal with (1) the theory revision task (*i.e.*, modifying an existing decision tree); (2) "complex" attributes (which correspond to combinations of other atomic attributes); (3) blockers that occasionally include superfluous values or exclude required values; and (4) other hypothesis classes (*e.g.*, DNF formulae). We first close this section by further motivating our framework and describing how it differs from related work on learning from incomplete data.[1]

Motivation and Related Work: Most implemented learning systems tend to work effectively when the percentage of missing features is very small, and when these missing features are randomly distributed across the samples. However, recent studies [PBH90, RCJ88] have shown that in practice many datasets are missing more than half of the feature values! Moreover, these attribute 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, just note that a diagnosis, corresponding to a single path through a n-node decision tree, can require performing only $O(\ln(n))$ of the n tests

[1] The extended version of this paper [GHR95] provides a more comprehensive literature review, as well as proofs of the claims presented here.

possible; here the remaining $O(\, n - \ln(n)\,)$ test values are irrelevant. Our model of learning can, therefore, be applicable to many diagnostic tasks, and will be especially useful where 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 differs, however, by using test-cost, rather than test-relevance, to decide which tests to omit.

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 learning context, which is appropriate for a different situation [SG93, SG94]: After one process draws completely-specified samples at random, a second process (which also could be "nature") then hides the values of certain attributes, perhaps changing the complete tuple $\langle 1, 0, 0, 1\rangle$ to the partial tuple $\langle *, 0, *, 1\rangle$, where "$*$" means this value is hidden. The learner is given only such partial tuples, each labeled with its correct class, and must produce a classifier capable of classifying such partially-specified (but unlabeled) instances. Notice the resulting classifier may not be completely accurate, as the missing data can be critical to producing the correct diagnosis. Here, the second process, called "blocking," is considered random, based on some distribution that is unknown to the learner.

While the model in this paper also assumes a random process is generating complete tuples which are then partially blocked, our model differs by dealing with blocking processes that are based on the values of the attributes and on certain particulars of the target classifier that the learner is trying to acquire. In particular, we assume that the unblocked data is sufficient to establish the correct diagnosis; hence, there is a classifier that will always produce the correct answer using only the unblocked data.

We view the values of the blocked attributes as "superfluous," or perhaps "relatively irrelevant": *irrelevant* as they are known not to play a role in classifying the sample, given the values of the other specified values (*i.e.*, *relative* to those values). *E.g.*, as the doctor will conclude `diseaseX` if t_1 is positive, whether t_2 is positive or negative, we say that "t_2 is *superfluous*, given that t_1 is positive". Of course, if t_1 is negative, other tests may then be relevant for the diagnosis; perhaps a negative t_2 and a positive t_3 will also be sufficient to establish `diseaseX`, etc. John *et al.* [JKP94] would consider t_1 to be "weakly irrelevant"; by contrast, 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, BHL91]. Of course, our situation differs from those models of learning, as our environment explicitly identifies which attributes are weakly irrelevant.

Two final comments to help place our model within the framework of existing computational learning results: First, in our model, certain *attribute* values are *omitted*; this differs from models where the *class* label is omitted [AS91, FGMP94], or where the attribute value is changed [SV88a, Lit91, GS95]. Second, as our blocker is providing additional information to the learner, its role is similar to that of a benevolent teacher. However previous teaching models such as that of Goldman and Mathias [GM93] 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 to be irrelevant.

2 Framework

Following standard practice, we identify each domain instance with a finite vector of boolean attributes $x = \langle x_1, \ldots, x_n \rangle$, and let $X_n = \{0,1\}^n$ be the set of all possible domain instances. The learner is trying to learn a concept c, which we view as an indicator function $c\colon X_n \mapsto \{T, F\}$, where x is a member of c iff $c(x) = T$. We assume the learner knows the set of possible concepts, $\mathcal{C}$.[2] A "(labeled) example of a concept $c \in \mathcal{C}$" is a pair $\langle x, c(x) \rangle \in X_n \times \{T, F\}$; and a m-sample of any $c \in \mathcal{C}$ is a sequence $\langle \langle x^1, c(x^1) \rangle, \ \ldots, \langle x^m, c(x^m) \rangle \rangle \in (X_n \times \{T, F\})^m$. 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 in the instance $x = \langle t_1, \ldots, t_4 \rangle$ corresponds to the t_1 blood test and the subsequent attributes t_2, t_3 and t_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 t_1 test was negative, but whose bile, melancholy and phlegm tests (t_2, t_3 and t_4) were all positive. Assume that the concept associated with `diseaseX` corresponds to any tuple $\langle t_1, \ldots, t_4 \rangle$ where either $t_1 = 1$ or both $t_2 = 0$ and $t_3 = 1$. 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, $P(x)$ specifies the probability of dealing with a patient with the particular set of symptoms specified by 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 $\langle x, c(x) \rangle$ according to the distribution P and labeled by the target concept $c \in \mathcal{C}$. L's output is a hypothesis h in $\mathcal{C}$. We discuss below how we this model interacts with our model of blocking, and then discuss how we evaluate L.

Model of "Blocked Learning": In standard learning models, each labeled instance $\langle x, c(x) \rangle$ is passed "as is" to the classifier. In our model, however, the learner instead only sees a "degraded version" of $\langle x, c(x) \rangle$, written "$\langle \beta(x), c(x) \rangle$", based on a blocker $\beta : X_n \to \{0, 1, *\}^n$ that replaces certain

[2] 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.

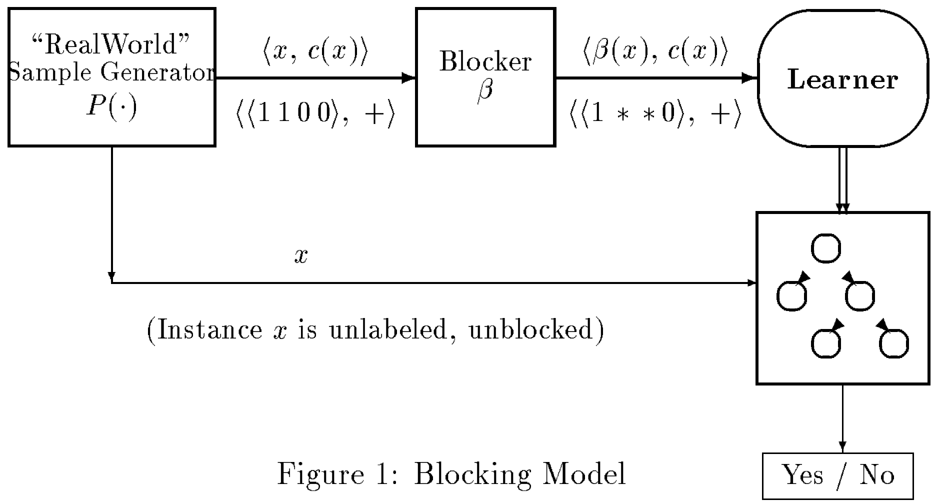


Figure 1: Blocking Model

attribute values by the "blocked" (or in our case, "don't care") token "$*$", but otherwise leaves x, and the label $c(x)$ intact; see Figure 1. 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 only *superfluous-value blockers*: *i.e.*, the blocker will block only 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") *Given a concept $c \in \mathcal{C}$ over the set of attributes $\{x_1, \ldots, x_n\}$:*
(1) *The attributes $\{x_{m+1}, \ldots, x_n\}$ are* superfluous *given the partial assignment $\{x_1 \mapsto v_1,\ \ldots,\ x_m \mapsto v_m\}$ (where each $v_i \in \{0,1\}$ is a specified value) iff*
$$\forall i = (m{+}1)..n,\ y_i \in \{0,1\} :\ c(\langle v_1, \ldots, v_m, y_{m+1}, \ldots, y_m\rangle) = c(\langle v_1, \ldots, v_m, 0, \ldots, 0\rangle)$$
(2) *a function $\beta : X_n \mapsto X_n^*$ is a legal blocker if it only blocks superfluous attributes; i.e., if $\beta(\langle v_1, \ldots, v_n\rangle) = \langle v_1, \ldots, v_m, *, \ldots, *\rangle$ implies 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, a blocker β can map the specified instance $\langle 1,0,1,1\rangle$ to the partial $\beta(\langle 1,0,1,1\rangle) = \langle *,0,1,*\rangle$ *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 class (*e.g.*, perhaps all are in class T).

To motivate this model, consider the behavior of a classifier d_t using a standard decision tree t, *à la* CART [BFOS84] or C4.5 [Qui92]. Here, given any instance, d_t will perform only the tests on a single path through t. Hence, d_t will see only the values of the attributes corresponding to those tests. Notice that 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.

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 and labeled according to concept c, as $Err(h) = P(x : c(x) \neq h(x))$, which is the probability that h will misclassify an instance x drawn from P.

We now use the standard "PAC-criterion" [Val84, KLPV87] to specify the desired performance of our learners:

Definition 2 (PAC-learning) *An algorithm, L,* PAC-learns *a set of concepts $\mathcal{C}$ if, for some polynomial function $p(\cdots)$, for all target concepts $c \in \mathcal{C}$, distributions P, and error parameters $\epsilon, \delta > 0$, L runs in time at most $p(\frac{1}{\epsilon}, \frac{1}{\delta}, |c|)$,*[3] *and outputs a hypothesis $h \in \mathcal{C}$ whose error is, with probability at least $1 - \delta$, under ϵ; i.e.,*

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

Notice the learner is expected to acquire a classifier that has high accuracy on *completely specified* instances, even though it was trained on blocked values. To explain why this is reasonable, note that we can assume that the classifier (read "doctor") is in a position to specify the tests should be performed. (Also, as irrelevant values are known not to matter for the classification, the learner can simply substitute any value, say "0", for each blocked attribute.)

3 Results for Decision Trees

This section discusses the challenge of learning decision trees, within this "superfluous blocking model" in the simplest situation; the next section deals with more complicated cases. For notation, let $\mathcal{DT}_n$ be the set of decision trees defined over n boolean variables, X_n.

The $DT[E]$ model assumes that the learner begins with an Empty initial decision tree, which is "grown" into a full decision tree based on a set of samples, and that the blocker is using the target decision tree, d_T, to classify the instances. On receiving a complete instance, $x = \langle x_1, \ldots, x_n \rangle$, the blocker traverses the d_T decision tree, from the root down, recording not just the appropriate classification for this instance $d_T(x) \in \{T, F\}$, but also the set of tests that were performed, which correspond to the path through d_T. The blocker will then pass on only the attributes encountered on the path, and will block all of the other attributes.

For example, imagine that a doctor was using the 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

[3] Here, $|c|$ is the "size" of the concept c, which is defined in subsequent sections. Note that by bounding the running time we also bound the size of sample the algorithm may use.

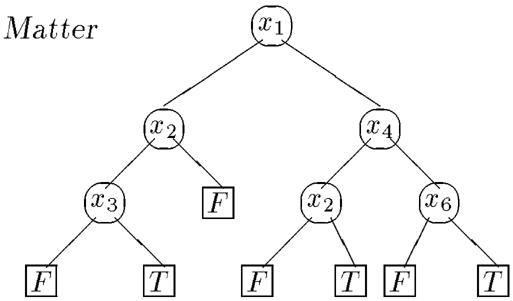
$d_0 \quad =$ 

Figure 2: Decision Tree, d_0

see $\langle \beta(\langle 0,1,1,0,0,0,0\rangle), F\rangle = \langle\langle 0,1,*,*,*,*\rangle, F\rangle$. Alternatively, given the instance $\langle 1,1,0,0,1,0\rangle$, the learner will see $\langle \beta(\langle 1,1,0,0,1,0\rangle), d_0(\langle 1,1,0,0,1,0\rangle)\rangle = \langle\langle 1,1,*,0,*,*\rangle, T\rangle$. Note that different trees that encode the same classification function can have different blocking functions.

There is currently no known algorithm capable of learning decision trees in the standard PAC model.[4] There is, however, a simple algorithm that can learn decision trees from these blocked instances in the $DT[E]$ model. If the target decision tree is believed to consist of no more than s nodes, the algorithm first draws a sample of

$$m_s \quad = \quad \frac{1}{\epsilon}\left[s\ln(8n) + \ln(1/\delta)\right]$$

random (blocked and labeled) training examples, and then calls the routine BUILDDT (figure 3) to build a small decision tree that correctly classifies this sample. The critical observations are 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. BUILDDT first selects as the root any attribute value that is never blocked, then splits on this attribute, and calls itself recursively.

While this algorithm is parameterized by the size of the tree s, it is easy to produce a modified version that does not require this parameter by using the standard technique [HKLW91] of repeated attempts with successively doubled estimates of $\mathtt{s}$. We call this modified procedure GROWDT(n, ϵ, δ).

Theorem 1 *For any $c_0 \in \mathcal{DT}_n$ any values of $\epsilon, \delta \in (0,1)$ and any distribution, under the $DT[E]$ model,* GROWDT*(n, ϵ, δ) will, with probability at least $1-\delta$, return a tree $c' \in \mathcal{DT}_n$, whose error (on unblocked, unlabeled instances) is at most ϵ. Moreover,* GROWDT *requires $O(|c_0|\frac{1}{\epsilon}\ln(\frac{n}{\delta}))$ blocked labeled samples and returns a tree of size $|c'| \leq |c_0|$ (where $|c|$ is the number of nodes in c).*

4 Extensions

4.1 Theory Revision: Non-Empty Initial Tree

In many situations, we may already have an initial decision tree, $d_{init} \in \mathcal{DT}_n$, which is considered quite accurate, but not perfect. This subsection describes

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

```
Algorithm BuildDT( S: set_of_partial_samples ): TreeType
           /* Builds a tree using samples S */
  Let NT be new tree
  If (all samples in S are labeled with same ℓ) or (S is empty) then
    NT.LeafLabel = ℓ    (or "... = T" if |S| = 0)
    Return( NT )
  Let x_i be any variable that is unblocked forall s ∈ S.
  Let S^0 = {⟨x − {x_i/0}, ℓ⟩ | ⟨x, ℓ⟩ ∈ S,  x_i/0 ∈ x}
      S^1 = {⟨x − {x_i/1}, ℓ⟩ | ⟨x, ℓ⟩ ∈ S,  x_i/1 ∈ x}
        /* To form S^0: Assemble the instances in S that include the x_i/0 assignment,
           then project out that attribute; similarly for S^1. */
  Let NT.InternalNodeLabel = x_i
      NT.If0 = BuildDT( S^0 );    NT.If1 = BuildDT( S^1 )
  Return( NT )
End BuildDT
```

Figure 3: BuildDT Algorithm for learning decision trees

ways of using a set of labeled samples to modify d_{init}; *i.e.*, to form a new tree d_{better} that is similar to d_{init}, but (with high probability) more accurate. We let $DT[TR{+}]$ 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.*, [Tow91, WP93, MB88, OM90, LDRG94]. (We explain the final "+" below.)

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 decision tree is proportional to the size of the final tree, which can be exponential in the number of attributes. This means that, given only a small number of labeled samples, we may be unable to produce even an adequate tree, much less the optimal one. This same set of samples, however, may be sufficient to specify how to improve a given initial theory, meaning the decision tree obtained by a theory revision process can be much better than one obtained from scratch. Another advantage of the theory revision approach is that it facilitates "learning while doing": That is, we could use the initial $d_0 = d_{init}$ to classify (unlabeled) instances as they are seen. If d_0's label is found to be incorrect,[5] we then consult an expert who provides the correct label, possibly after asking for (and receiving) the values of the critical attributes that d_0 had inappropriately judged to be superfluous. The learner can use this information to form a new decision tree, call it d_1, which is then used as its classifier. The theory revision process then iterates: consulting the expert if d_1 produces an incorrect label, and using that new information to produce a newer, better d_2, and so forth.[6] Here, the

[5]For example, if d_0 is being used to propose a repair to a faulty device, the user can trivially see whether the proposed repair worked or not.

[6]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 the implementation and experiments with a system in this model.

total amount of expert-time may be much less than if we began with an empty tree.

To explain the $DT[TR+]$ model in more detail: As before, the real-world continues to generate completely-specified samples x, each labeled $d_T(x) \in \{T, F\}$ by the target (but unknown) decision tree d_T. Here, the blocker first runs this sample through the *initial* decision tree d_{init}, to obtain both a proposed (but not necessarily correct) label $d_{init}(x)$ and a particular set of unblocked attribute values; call this $\beta_{d_{init}}(x)$. (To motivate this: 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, d_{init}.)

If d_{init}'s label is correct (*i.e.*, if $d_T(x) = d_{init}(x)$), then the learner receives $\langle \beta_{d_{init}}(x), d_T(x) \rangle$. Otherwise, $d_T(x) \neq d_{init}(x)$, the learner receives $\langle \beta_{d_{init} \vee d_T}(x), d_T(x) \rangle$, where the $\beta_{d_{init} \vee d_T}$ blocker specifies the value of any attribute that is unblocked by either $\beta_{d_{init}}$ or β_{d_T}. Hence, if $\beta_{d_{init}}(\langle 0,0,1,1,0,0 \rangle) = \langle 0,0,*,*,*,* \rangle$ and $\beta_{d_T}(\langle 0,0,1,1,0,0 \rangle) = \langle *,0,1,*,*,* \rangle$, then $\beta_{d_{init} \vee d_T}(\langle 0,0,1,1,0, \rangle) = \langle 0,0,1,*,*,* \rangle$.

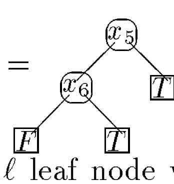
The ModifyDT **Algorithm:** The ModifyDT process is designed to handle the $DT[TR+]$ situation. We outline a "batch" version, which produces a new d' after seeing a large enough quantity of labeled partially-specified values.[7]

Like GrowDT, ModifyDT first accumulates a sufficient set of samples, then uses them to produce a decision tree of a given size; it then produces a new tree by replacing many of d_{init}'s leaf nodes with new subtrees. That is, we can identify each presented $\langle \beta_{d_{init} \vee d_T}(x), d_T(x) \rangle$ instance with the particular leaf node in d_{init} that d_{init} would reach in processing that sample. For example, using the d_0 tree from Figure 2, we would identify $\langle \langle 0,0,0,*,1,* \rangle, T \rangle$ with the far left $[F]$-labeled node, call it ℓ_{000---}. For each leaf node ℓ, let $Inst(\ell)$ be the subset of the samples identified with ℓ, including only the attributes that are not in the path to ℓ. So, perhaps

$$Inst(\ell_{000---}) \quad = \quad \left\{ \begin{array}{c} \langle \langle -,-,-,*,1,* \rangle, T \rangle \\ \langle \langle -,-,-,*,0,0 \rangle, F \rangle \\ \langle \langle -,-,-,*,0,1 \rangle, T \rangle \end{array} \right\}$$

ModifyDT then calls BuildDT to build a tree, based on this $Inst(\ell)$ set. Here, this produces

BuildDT($Inst(\ell_{000---})$) $=$



Finally, ModifyDT replaces each ℓ leaf node with the new BuildDT($Inst(\ell)$) subtree. Here, this produces the the d_1 tree shown in Figure 4.

[7]The extended paper [GHR95] presents the actual pseudo-code for this ModifyDT algorithm, and also bounds the number of samples needed, as a function of the number of additional nodes that must be added. It also presents a modified version that works incrementally: potentially updating the current decision tree after seeing each sample that is either incorrectly labeled or incorrectly blocked.

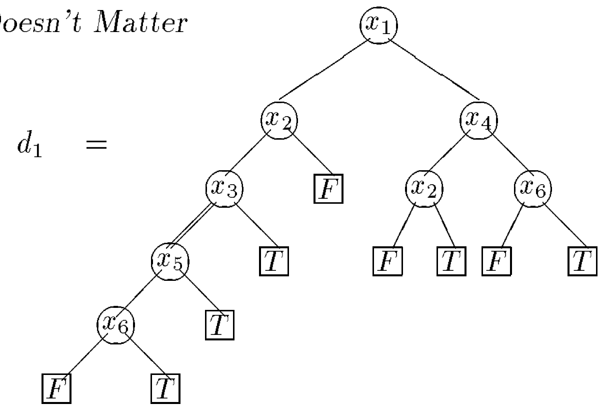


Figure 4: Tree MODIFYDT formed from d_0, based on $\langle\langle 0,0,0,*,1,*\rangle, F\rangle$, $\langle\langle 0,0,0,*,0,0\rangle, F\rangle$, $\langle\langle 0,0,0,*,0,1\rangle, T\rangle$

Notice that d_1 will correctly label each of the instances seen. (MODIFYDT can then perform some post-processing, to simplify the tree — *e.g.*, if both branches from an internal node descend to functionally-equivalent subtrees, than that node and its subtrees can be replaced with one copy of the subtree; etc.)

It is easy to see that that the resulting tree d' will return the correct answer for each instance x corresponding to a sample seen (*i.e.*, for each $s \in S$), and otherwise, when no $s \in S$ matches x, d' will return the same answer as the original d_{init}.

Learner is also told which attributes are NOT needed: In the $DT[TR+]$ model, the "environment" (read "expert") tells the learner which additional attributes are necessary, via the $\beta_{d_T \vee d_{init}}$ blocker. An alternative blocker might also specify that some of the attributes that d_{init} included are, in fact, superfluous — *i.e.*, that some of the tests d_{init} requested should not have been run. The $DT[TR\pm]$ model provides this information by using the β_{d_T} blocker. (As the learner knows d_{init}, it can determine the values of $\beta_{d_{init}}$, if necessary.)

While we could simply use the GROWDT algorithm to produce a tree from scratch in this situations, for the reasons given above, we instead prefer to produce a new tree that continues to return d_{init}'s answers for each instance not seen in the sample. To do this, our MODIFYDT2 algorithm will grow the tree "inside-out": It will use the labeled samples as GROWDT did, to grow a tree *from the root*. However, where GROWDT would put a leaf node (labeled with a class), MODIFYDT2 will instead put a diminished subtree of d_{init}, corresponding to the parts of the initial d_{init} that were not eliminated in the path to that leaf. For example, suppose MODIFYDT2, working on the initial tree d_0, received (only) the labeled instance $a = \langle\langle *,0,*,*,1,*\rangle, F\rangle$. It would then produce the d_2 tree show in Figure 5. 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 ℓ_{-0--1-} (labeled with $[F]$). Now observe the two other subtrees, which correspond to the $x_2 = 1$ and $x_2 = 0$-and-$x_5 = 0$ situations; here MODIFYDT2 places diminished versions of d_0, which correspond to the $\{x_2/1\}$ and $\{x_2/0,\ x_5/0\}$ assignments. (For pedagogical reasons, Figure 5

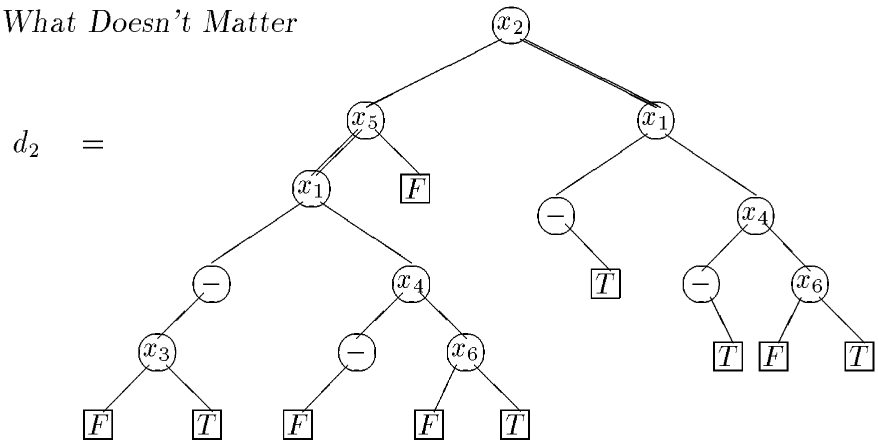


Figure 5: Tree ModifyDT2 formed from d_0, based on $\langle\langle 0,*,1,*,*,*\rangle, F\rangle$

includes "phantom nodes", shown as $\ominus$which correspond to where x_2 and x_5 had been in the original d_0 tree, to illustrate the connection. Of course, these nodes would not be present in the subtree that ModifyDT2 would produce.)

In general, we can view ModifyDT2's result as an initial "prefix" subtree, which we call d_{gdt}, whose "leafs" correspond to sub-trees. The prefix d_{gdt} is exactly what GrowDT would produce, using the given set of samples. (For example, given only $\langle\langle *,0,*,*,1,*\rangle, F\rangle$, GrowDT would produce the $x_2 - x_5$ tree embedded within Figure 5's d_2 tree. Given more samples, ModifyDT2 would produce a more elaborate tree.) Each of the "leaf sub-trees" is a diminished version of the initial d_0, which omits the nodes corresponding to known assignments, in the sense that d_2's $x_2 = 1$ subtree has eliminated the node that had been labeled x_2, replacing that node with its $x_2/1$ child. This means the resulting d_2 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.

4.2 "Complex Attributes"

The algorithms discussed above all assume that each "node" in the decision tree corresponds to a single test, *e.g.*, the t_1 mentioned above. In general, the doctor may have several tests that are equally able to determine the presence of some symptom — *e.g.*, he may be able to determine a patient's blood type by either asking the patient, or by performing a simple blood test. Here, the doctor may decide on t_{1a} or t_{1b} based on the availability of some resource, the time of day, or some other external factors. We can model this by assuming the doctor flips a coin to decide whether to use t_{1a} or t_{1b}. The resulting "decision structure" no longer corresponds to a decision tree, as the prior test results no longer deterministically specify which test will be run. Worse, notice the algorithms presented above will not work here, as there will be no single attribute whose value is always specified when reaching this "$t_{1a} \vee t_{1b}$" situation.

There is, however, another straightforward routine that can learn such structures, CA-BuildDT$_k$. If ever CA-BuildDT$_2$ reaches a situation where there is no "categorical attribute", it then simply looks for a pair of attributes whose specified values are disjoint and exhaustive; here, it would find that $\{\ t_{1a},\ t_{1b}\ \}$ have this property. It would then create a node labeled "$t_{1a} \vee t_{1b}$", and recur appropriately: the +-branch under this node will include all instances whose t_{1a} value was + when t_{1a} was specified, or whose t_{1b} value was + when t_{1b} was specified; and the others all will go under the − arc.

There is an obvious extension of this idea to handle arbitrary boolean combinations of any k-tuple of attributes; the CA-BUILDDT$_k$ will requires $O(n^k)$ time to find the appropriate k-tuple.

4.3 Imperfect blocking

So far our model has assumed that the blocker removes all-and-only the superfluous attribute values, and also never "flips" the value of any attribute (*i.e.*, never presents a `1` as a `0`). It is easy to generalize this model by removing these restrictions, producing a $\beta^{\vec{p}}$ blocker, parameterized by the 6-tuple, $\vec{p} = \langle p_{rc}, p_{r*}, p_{ri}, p_{sc}, p_{s*}, p_{si} \rangle$: Given any target decision tree d_T, for each instance, we can label each attribute value as either required or superfluous. In the "uniform" case, for each required attribute, $\beta^{\vec{p}}$ stochastically makes a 3-way decision: presenting the attribute's correct value with probability p_{rc}, $*$ with probability p_{rc}, and the incorrect value with probability p_{ri} (*e.g.*, using 0 when the correct attribute value is 1). Similarly, if the attribute is superfluous, $\beta^{\vec{p}}$ will return the attribute's correct value, the value "$*$" or the wrong value, with respective probabilities p_{sc}, p_{s*}, or p_{si}. Hence,

	Attribute value is	
Blocker's value is:	Required	Superfluous
Correct	p_{rc}	p_{sc}
$*$	p_{r*}	p_{s*}
Incorrect	p_{ri}	p_{si}

The $DT[E]$ model, discussed above, uses $p_{rc} = p_{s*} = 1$ with the other p_i values all 0; and the traditional complete-tuple model uses $p_{rc} = p_{sc} = 1$. Previous attribute noise models [KL88, SV88b, GS95] do not consider missing attribute values (*i.e.*, they assume $p_{r*} = p_{s*} = 0$).

We also consider the "product" (as opposed to "uniform") situation: Here, the probability that the j^{th} attribute is blocked (resp., presented incorrectly) when it is relevant is *bounded by* p_{r*} (respectively, bounded by p_{ri}); notice that different attributes can have different probability values here. Similarly, when the j^{th} attribute is superfluous, its probability of being blocked or incorrect is bounded by p_{s*} or p_{si}. Clearly the product model is strictly more general than the uniform model.

The following theorem specifies certain other settings of $\vec{p}$ under which it is possible to PAC-learn decision trees:

Theorem 2 *1. It is easy to* PAC*-learn* $\boxed{\begin{matrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{matrix}}$ *and* $\boxed{\begin{matrix} 1 & p \\ 0 & 1-p \\ 0 & 0 \end{matrix}}$, *in either the uniform or product case.*

2. Learning decision trees in the $\boxed{\begin{matrix} 1-p & p \\ p & 1-p \\ 0 & 0 \end{matrix}}$ *product model is as hard as learning decision trees in the standard model, when* $p \geq 1/2$.

3. For information theoretic reasons, no algorithm can (ϵ, δ)*-PAC-learn boolean formulae with* n *variables in the* $\boxed{\begin{array}{cc} 1-\nu & 1-\nu \\ 0 & 0 \\ \nu & \nu \end{array}}$ *product model if* $\nu \geq 6\epsilon/n$*, even if the value of* ν *is known, provided* $\nu < 1/2$ *and* $\epsilon < 1/9$*. (Of course, this shows the hardness of* pac*-learning* $\mathcal{DT}_n$ *with attribute noise.)*

4.4 Dealing with Arbitrary DNF Formulae

We can also consider learning other classes of formulae, beyond decision trees. In particular, consider learning arbitrary DNF formulae when superfluous values are omitted. Here, each blocked instance is simply an implicant of either the target formula φ or its negation $\neg\varphi$. However, while decision trees have an obvious "evaluation procedure" that describes the particular implicant to use, there are many different ways of specifying which implicant should be returned when considering DNF formulae. We consider two obvious possibilities, and for each, discuss whether it allows the class to be learned.

Select Appropriate Term: Consider a particular encoding of the DNF formula, as a particular disjunction of conjunctive terms. Here, when given a positive instance, a benevolent blocker (think "teacher") can simply specify the variables in the first term the instance satisfies; the blocker can return an arbitrary implicant of $\neg\varphi$ for each negative instance. It is easy to see that this renders the learner's task trivial.

Ordering of the Variables: Another model assumes the blocker has ordered the variables, say $\langle x_1, \ldots, x_n \rangle$, and given any instance, returns the values of the smallest initial "prefix" of this ordering that qualifies as an implicant. If an adversary can specify this ordering, then learning in this model can be as difficult as learning DNF in the usual "completely-specified tuple" model. (Here, the adversary can essentially force every variable of a "hard-to-learn formula" to be specified.)

5 Conclusion

Even though most classification systems perform and record only a small fraction of the set of possible tests to reach a classification, most learning systems are designed to work best when their training data consists of completely-specified attribute-value tuples. This report presents learning algorithms designed to deal with exactly the partially-specified instances that classification systems tend to produce. We show, in particular, that it is easy to "PAC learn" decision trees in this model — a class of structures that is not known to be learnable if the learner is given completely-specified tuples. We also show how this algorithm can be extended to incrementally modify a given initial decision tree, handle "complex attributes", and also extend this model to handle various types of "noise" — where the blocker can stochastically omit required values, or include superfluous values. Finally, we consider the complications in going beyond decision trees, describing variants of this "superfluous value blockers" that pertain to more general classes, such as arbitrary DNF formulae, and present situations when it is easy, versus difficult, to learn such classes. These results, when coupled with [SG93, SG94], help to form a comprehensive description of how learning algorithms should deal with missing data.

References

[AS91] D. Angluin and D. K. Slonim. Learning monotone DNF with an incomplete membership oracle. In *Proc. 4th Annu. Workshop on Comput. Learning Theory*, pages 139–146. Morgan Kaufmann, San Mateo, CA, 1991.

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

[BHL91] A. Blum, L. Hellerstein, and N. Littlestone. Learning in the presence of finitely or infinitely many irrelevant attributes. In *Proc. 4th Annu. Workshop on Comput. Learning Theory*, pages 157–166. Morgan Kaufmann, San Mateo, CA, 1991.

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

[EH89] A. Ehrenfeucht and D. Haussler. Learning decision trees from random examples. *Information and Computation*, 82(3):231–246, 1989.

[FGMP94] Mike Frazier, Sally Goldman, Nina Mishra, and Leonard Pitt. Learning from a consistently ignorant teacher. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pages 328–339, 1994.

[GHR95] Russell Greiner, Thomas Hancock, and R. Bharat Rao. Knowing what doesn't matter: Exploiting (intentionally) omitted superfluous data. Technical report, Siemens Corporate Research, 1995.

[GM93] S. Golman and D. Mathias. Teaching a smarter learner. In *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory*, pages 67–76. ACM Press, New York, NY, 1993.

[GS95] Sally A. Goldman and Robert A. Sloan. Can pac learning algorithms tolerate random attribute noise? *Algorithmica*, page to appear, 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 Workshop*, pages 121–29, N.J., 1994.

[KL88] M. Kearns and M. Li. Learning in the presence of malicious errors. In *Proc. 20th Annu. ACM Sympos. Theory Comput.*, pages 267–280. ACM Press, New York, NY, 1988.

[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.

[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. Morgan Kaufmann, San Mateo, CA, 1991.

[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.

[OM90] Dirk Ourston and Raymond J. Mooney. Changing the rules: A comprehensive approach to theory refinement. In *Proceedings of AAAI-90*, pages 815–20, 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 Acquistion for Knowledge-Based Systems Workshop*, 1988.

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

[SG93] Dale Schuurmans and Russell Greiner. Learning to classify incomplete examples. In *Fourth Annual Workshop on Computational Learning Theory and 'Natural' Learning Systems (CLNL93)*, Provincetown MA, 1993.

[SG94] Dale Schuurmans and Russell Greiner. Learning default concepts. In *CSSCI-94*, 1994.

[SV88a] G. Shackelford and D. Volper. Learning k-DNF with noise in the attributes. In *Proceedings COLT-88*, pages 97–103, 1988.

[SV88b] G. Shackelford and D. Volper. Learning k-DNF with noise in the attributes. In *Proc. 1st Annu. Workshop on Comput. Learning Theory*, pages 97–103, San Mateo, CA, 1988. published by Morgan Kaufmann.

[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 inductin algorithm. *Journal of AI Research*, (accepted subject to revision), 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.