Learning Boolean Concepts in the Presence of Many Irrelevant Features

Hussein Almuallim
Department of Information and Computer Science
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia
E-mail: facp07b@saupm00.bitnet

Thomas G. Dietterich
Department of Computer Science
Oregon State University
Corvallis, OR 97331-3202, U.S.A.
E-mail: tgd@cs.orst.edu

Correspondence to: Hussein Almuallim, Department of Information and Computer Science, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia.

Abbreviated title: Learning with many irrelevant features.

Abstract

In many domains, an appropriate inductive bias is the MIN-FEATURES bias, which prefers consistent hypotheses definable over as few features as possible. This paper defines and studies this bias in Boolean domains. First, it is shown that any learning algorithm implementing the MIN-FEATURES bias requires $\Theta(\frac{1}{\epsilon}\ln\frac{1}{\delta}+\frac{1}{\epsilon}[2^p+p\ln n])$ training examples to guarantee PAC-learning a concept having p relevant features out of n available features. This bound is only logarithmic in the number of irrelevant features.

For implementing the MIN-FEATURES bias, the paper presents five algorithms that identify a subset of features sufficient to construct a hypothesis consistent with the training examples. FOCUS-1 is a straightforward algorithm that returns a minimal and sufficient subset of features in quasi-polynomial time. FOCUS-2 does the same task as FOCUS-1 but is empirically shown to be substantially faster than FOCUS-1. Finally, the Simple-Greedy, Mutual-Information-Greedy and Weighted-Greedy algorithms are three greedy heuristics that trade optimality for computational efficiency.

Experimental studies are presented that compare these exact and approximate algorithms to two well-known algorithms, ID3 and FRINGE, in learning situations where many irrelevant features are present. These experiments show that—contrary to expectations—the ID3 and FRINGE algorithms do not implement good approximations of MIN-FEATURES. The sample complexity and generalization performance of the FOCUS algorithms is substantially better than either ID3 or FRINGE on learning problems where the MIN-FEATURES bias is appropriate. These experiments also show that, among our three heuristics, the Weighted-Greedy algorithm provides an excellent approximation to the FOCUS algorithms.

1 Introduction

Historically, the development of inductive learning algorithms has been a two-step process: (i) select a representation scheme (e.g., decision trees), (ii) develop an algorithm to find instances of the scheme that are consistent with given collections of training examples. A shortcoming of this approach is that there is no separation between a *specification* of the desired learning behavior of the algorithm and its *implementation*. Specifically, the *bias* of the algorithm is adopted implicitly, particularly as a side-effect of the second step. Often, it is difficult even to state the bias in any simple way. Consequently, it is difficult to tell in advance whether the bias is appropriate for a new learning problem.

Recently, a few authors [5, 24] have advocated a different procedure: (i) adopt a bias over some space of hypotheses (or, equivalently, select a prior probability distribution over the space), (ii) select a scheme for representing hypotheses in the space, and (iii) design an algorithm that implements this bias, at least approximately.

The goal of this paper is to pursue this second procedure. We consider the space of all binary functions defined over n Boolean input features. We adopt the following bias, which we call the MIN-FEATURES bias: if two functions are consistent with the training examples, prefer the function that involves fewer input features (break ties arbitrarily). This is a bias in favor of simplicity—but not mere syntactic simplicity. Functions over fewer variables are *semantically* simpler than functions over more variables, because they consider fewer aspects of each example.

We begin by adopting a straightforward representation for binary functions defined over n input features. We then analyze the sample complexity of any probably-approximately correct (PAC) learning algorithm that implements the MIN-FEATURES bias. It is proved that any such algorithm requires

$$\Theta\left(\frac{1}{\epsilon}\ln\frac{1}{\delta}+\frac{1}{\epsilon}\left[2^p+p\ln n\right]\right)$$

training examples to PAC-learn a binary concept involving p input features (out of a space of n input features) with accuracy parameter ϵ and confidence parameter δ.

Note in this bound that the total number of available features n appears only logarithmically. Hence, if there are k irrelevant features, it only costs us a factor of $\ln k$ training examples to detect and eliminate them from consideration.

Following this sample complexity analysis, we turn to the problem of implementing the MIN-FEATURES bias. This bias can be implemented in two steps: First, we identify the smallest subset of features that is sufficient to construct a hypothesis consistent with the given training examples. Then, we apply some learning procedure (e.g., ID3 [23]) that focuses on just those chosen features. For identification of the smallest sufficient subset of features we describe two algorithms. The first algorithm, FOCUS-1, is a straightforward algorithm that runs in time quasi-polynomial in the appropriate learning parameters. The second algorithm, FOCUS-2, is a more sophisticated algorithm which is empirically shown to be substantially faster than FOCUS-1.

For situations where implementing the exact MIN-FEATURES bias is computationally unaffordable, we introduce the Mutual-Information-Greedy, Simple-Greedy and Weighted-Greedy algorithms, which apply efficient heuristics for approximating the MIN-FEATURES bias. Unlike FOCUS-1 and FOCUS-2, these algorithms employ *greedy* heuristics that trade optimality for computational efficiency. Experimental studies show that these heuristics approximate the MIN-FEATURES bias to various levels of success. Particularly, the Weighted-Greedy algorithm is shown to provide an excellent approximation of the FOCUS algorithms but with substantially lower computational costs.

At first glance, it may appear that there are already many algorithms that approximate the MIN-FEATURES bias. For example, ID3 [23] has a bias in favor of small decision trees, and small trees would seem to test only a subset of the input features. In the experimental part of this work, we compare ID3 and FRINGE [21] to our algorithms. The experiments demonstrate that ID3 and FRINGE do not provide good approximations to the MIN-FEATURES bias—these algorithms often produce hypotheses as output that are much more complex (in terms of the number of input features used) than the hypotheses found by the algorithms we propose. Indeed, there are some cases in which ID3 and FRINGE miss extremely simple hypotheses.

These results suggest that the FOCUS and the Weighted-Greedy algorithms will require fewer training examples and generalize more correctly than ID3 in domains where the MIN-FEATURES bias is appropriate. We believe there are many such domains. For example, in many practical applications, it is often not known exactly which input features are relevant or how they should be represented. The natural response of users is to include all features that they believe could possibly be relevant and let the learning algorithm determine which features are in fact worthwhile.

Another situation in which many irrelevant features may be present is when the same body of training data is being used to learn many different binary functions. In such cases, one must ensure that the set of features measured in the data is sufficient to learn all of the target functions. However, when learning each individual function, it is likely that only a small subset of the features will be relevant. This applies, for example, to the task of learning diagnosis rules for several different diseases from the medical records of a large number of patients. These records usually contain more information than is actually required for describing each disease. Another example (given in [16]) involves pattern recognition tasks in which feature detectors automatically extract a large number of features for the learner's consideration, not knowing which might prove useful.

The task of selecting a subset of the available features that meets a given criterion has long been known in the field of pattern recognition as the problem of "feature selection" or "dimensionality reduction." However, most feature selection criteria in pattern recognition are defined with respect to a specific classifier or group of classifiers. For example, [14] shows methods for selecting a small subset of features that optimizes the expected error of the nearest neighbor classifier. Similar work has addressed feature selection for the Box classifier [11], the linear classifier [12] and the Bayes classifier [22]. Other work (aimed at removing feature redundancy when features are highly correlated) is based on performing a principal components analysis to find a reduced set of new uncorrelated features defined by *combining* the original features using the eigenvectors [18, 19]. To our knowledge, the problem of finding the smallest subset of Boolean features that is sufficient to construct a consistent hypothesis—which is the topic of this paper—has not been addressed.

2 Preliminaries

For each $n \geq 1$, let $\{x_1, x_2, \cdots, x_n\}$ denote a set of n Boolean features and U_n denote the set $\{0,1\}^n$ of all assignments to these features—the set of *instances*. A *concept* c is a subset of U_n (i.e., all positive instances of c). A Boolean formula f *represents* a concept c if $f(X) = 1$ for all $X \in c$ and $f(X) = 0$ otherwise.

For an instance $X \in U_n$, let $X^{(i)}$, for $1 \leq i \leq n$, denote the value of the feature x_i in the instance X. A feature x_i is said to be *relevant* to a concept c if there exist two instances $X_1 \in c$ and $X_2 \notin c$ such that $X_1^{(i)} \neq X_2^{(i)}$ and $\forall_{j \neq i}\ X_1^{(j)} = X_2^{(j)}$. The feature x_i is said to be *irrelevant* otherwise.

We assume an arbitrary probability distribution D on U_n. The *error* between two concepts c and h is the sum of the probability of all the instances in the symmetric difference of c and h, or formally

$$error_D(c,h) = \sum_{X \in\ c \oplus h} Pr_D[X]$$

where $\oplus$ indicates symmetric set difference. For $0 < \epsilon < 1$, a concept h is said to be ϵ-*close* to a concept c with respect to D if $error_D(c,h)$ is at most ϵ.

An *example* of c is a pair of the form $\langle X, class \rangle$, where $X \in U_n$ and *class* is the label $+$ if $X \in c$ and the label $-$ otherwise. The example is called *positive* in the first case, and *negative* in the second case. A *sample* of a concept c is a multi-set of instances drawn randomly (with replacement) according to D, and labeled according to c. The *size* of the sample is the number of these examples.

A *learning algorithm* is an algorithm that takes as input a sample of some unknown concept (to be called the *target* concept), and returns as output some concept $h \subseteq U_n$. The sample given to a learning algorithm is usually called the *training sample* and each example in the sample is called a *training example*. h is called the *hypothesis* of the algorithm. If h classifies all the examples in the training sample as given in the sample, then h is said to be a *consistent* hypothesis. That is, h is consistent with a sample S if for each example $\langle X, + \rangle \in S, X \in h$ and for each example $\langle X, - \rangle \in S, X \notin h$. An algorithm that on any training sample returns a consistent hypothesis is called a *consistent* algorithm. In other words, a consistent learning algorithm is an algorithm whose hypothesis never disagrees with the training sample.

In this work, we adopt the notion of Probably Approximately Correct (PAC) learning as defined by Blumer et al. [6]. Let L be a learning algorithm and let ϵ and δ be such that $0 < \epsilon, \delta < 1$. We say that "L *PAC learns* (or simply, *learns*) a concept c with respect to parameters ϵ and δ using a sample of size m" if, with probability at least $(1-\delta)$, L returns as an hypothesis a concept that is ϵ-close to c when the algorithm is given a sample of c of size m drawn under any fixed but unknown probability distribution D. In other words, let S_D^m denote a sample of m examples of c drawn randomly under probability distribution D, and let $L(S_D^m)$ denote the hypothesis returned by L when given S_D^m. With respect to ϵ and δ, to say that the algorithm L learns the concept c using a sample of size m, it is required that for any D

$$Pr[error_D(c, L(S_D^m)) \leq \epsilon] \geq 1 - \delta$$

where the probability is taken over all samples of c of size m. ϵ and δ are called the *accuracy* and *confidence* parameters, respectively.

The complexity of a concept, denoted $s(c)$, is defined to be the minimum number of bits needed to encode the concept with respect to some encoding scheme. The encoding scheme we use in this work will be introduced in Section 4. We let $C_{n,s}$ denote the set of all concepts of complexity at most s defined on $\{x_1, x_2, \cdots, x_n\}$. The *sample complexity* of a learning algorithm L is the minimum $m(n, s, \epsilon, \delta)$ such that L learns every concept in $C_{n,s}$ with respect to ϵ and δ using a sample of size m.

Given a training sample S and a set of features Q, a *sufficiency test* is a procedure for checking whether Q is sufficient to form an hypothesis consistent with S. The sufficiency test can be implemented simply by checking whether the sample contains a pair $\langle X_1, + \rangle$ and $\langle X_2, - \rangle$ of positive and negative examples such that X_1 and X_2 have the same values for all the features in Q. If such a pair appears, then Q cannot discriminate all of the positive examples from all of the negative examples. In general, Q is a sufficient set if and only if no such a pair appears in the training sample.

Given a sufficient subset of features, it is easy to construct a consistent hypothesis. For example, the algorithm ID3 [23] can be applied to the training sample but restricted to consider only the features in the given subset. Hence, in the rest of this paper, finding a *solution* will be taken to mean identifying a subset of features sufficient to form a hypothesis consistent with the given training sample.

3 The MIN-FEATURES Bias

The MIN-FEATURES bias can be stated simply. Given a training sample S of some unknown target concept c, let V be the set of all possible hypotheses consistent with S. (V is sometimes called the *version space*; [17]) Let H be the subset of V whose elements have the fewest relevant features. The MIN-FEATURES bias chooses its guess, h, from H arbitrarily.

Note that the MIN-FEATURES bias is "incomplete" in the sense that it does not necessarily lead to a unique hypothesis. Therefore, to construct a learning algorithm we need an additional mechanism to enable the algorithm to choose among those alternatives that are equally good under the MIN-FEATURES bias. Regardless of what mechanism is employed, an algorithm is said to be *implementing* the MIN-FEATURES bias as long as it never outputs an hypothesis that is defined on a number of features larger than the minimum possible, i.e. it never "violates" the MIN-FEATURES bias.

4 Sample Complexity Analysis

Given that we wish to implement and analyze the MIN-FEATURES bias, the first step is to choose a representation for hypotheses. We will represent a concept c by the concatenation of two bit vectors R_c and T_c. R_c is an n-bit vector in which the i-th bit is 0 if and only if x_i is irrelevant to c. T_c is the right-most column of the truth table of a Boolean function f that represents c defined only on those features in $\{x_1, x_2, \cdots, x_n\}$, whose corresponding bits in R_c are set to 1. Following Blumer et al. [7], we will let the complexity $s(c)$ for concept c be the number of bits needed to encode c using the above bit-vector representation. This measure has the desired property that $s(c_1) < s(c_2)$ iff the number of relevant features of c_1 is less than the number of relevant features of c_2. Specifically, if c has p relevant features then $s(c) = n + 2^p$.

Example: Let $n = 5$ and let c be a concept represented by $x_1 \vee x_3$. Then, $R_c = 10100$ and $T_c = 0111$. Hence, the complexity of c is 9. □

The goal of this section is to analyze the sample complexity—that is, the number of training examples required to ensure PAC learning of an arbitrary target concept of complexity s. The sample complexity of a learning algorithm is expressed in terms of the total number of features n, the complexity of the target concept s, the accuracy parameter ϵ, and the confidence parameter δ.

To derive an upper bound on the sample complexity of learning algorithms that implement the MIN-FEATURES bias, we make use of the following well-known lemma given by Blumer et al. [7].

Lemma 1 *Let C be a set of concepts and let L be a consistent learning algorithm which always chooses its hypothesis from C. Then, under any probability distribution D, any ϵ and δ such that $0 < \epsilon, \delta < 1$ and any concept $c \in C$, a sample of c of size*

$$\frac{1}{\epsilon}\left(\ln |C| + \ln \frac{1}{\delta}\right)$$

is sufficient to guarantee that L returns an hypothesis that is ϵ-close to c with probability at least $1 - \delta$.

This result is useful in deriving upper bounds on the sample complexity in cases where the size of the hypothesis space explored by the learning algorithm can be computed analytically. This is the case for algorithms that implement the MIN-FEATURES bias. Using the above lemma, we can state the following upper bound on the sample complexity of such algorithms.

Theorem 1 *Let $C_{n,s}$ denote the class of concepts defined on n features with complexity at most s. Then, under any probability distribution D, any $n \geq 1$, any ϵ and δ such that $0 < \epsilon, \delta < 1$ and any concept $c \in C_{n,s}$, a sample of c of size*

$$\frac{1}{\epsilon} \ln \frac{1}{\delta} + \frac{1}{\epsilon} \left[\log_2(s-n) \ln n + (s-n) \ln 2\right]$$

is sufficient to guarantee that any algorithm implementing the MIN-FEATURES bias will return an hypothesis that is ϵ-close to c with probability at least $1-\delta$.

Proof: By definition, the number of features needed to define the hypothesis of any algorithm implementing the MIN-FEATURES bias is less than or equal to the number of features relevant to the target concept. Therefore, for any target concept of complexity at most s, the hypothesis space for any algorithm that implements the MIN-FEATURES bias is contained in $C_{n,s}$. Every concept in $C_{n,s}$ is defined on at most $p = \log_2(s-n)$ features out of a total of n features. The number of concepts one can define on at most p specific features is 2^{2^p}. Since there are $\binom{n}{p}$ ways to choose p out of n features, it must be true that

$$\begin{aligned} |C_{n,s}| &\leq \binom{n}{p} 2^{2^p} \\ &\leq n^p 2^{2^p} \\ &= n^{\log_2(s-n)} 2^{(s-n)}. \end{aligned}$$

Substituting this in the sufficient sample size given by Lemma 1 completes the proof. □

It is interesting to note that the number of examples sufficient for learning grows only logarithmically in the number of irrelevant features and linearly in the complexity of the concept.

We now show that this bound is tight by exhibiting an identical lower bound using the methods developed by Blumer et al. [6] exploiting the Vapnik-Chervonenkis dimension (VC-dimension).

The VC-dimension of a class of concepts C is defined to be the largest integer d such that there exists a set of d instances that can be labeled by the concepts in C in all the 2^d possible ways. In other words, suppose S is a fixed set of instances, and let $\ell(S)$ denote an arbitrary labeling of the instances in S as positive or negative. Note that there exist a total of $2^{|S|}$ such labelings. If for any $\ell(S)$ we can find some $c \in C$ that gives the same labeling to the elements in S as given by $\ell(S)$, then we say that the set S is *shattered* by C. The VC-dimension of C is just the size of the largest such S, that is the size of the largest set of instances that is shattered by C.

Blumer et al. show that the number of examples needed for learning any class of concepts strongly depends on the VC-dimension of the class [6]. Specifically, Ehrenfeucht et al. [9] prove the following:

Theorem 2 *Let C be a class of concepts and $0 < \epsilon, \delta < 1$. Then, any algorithm that learns every concept in C with respect to ϵ, δ and any probability distribution must use a sample of size*

$$\Omega\left(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{VCdim(C)}{\epsilon}\right).$$

To apply this result, we must first derive a lower bound on the VC-dimension of $C_{n,s}$, the set of concepts over n features having complexity less than or equal to s. Note that if we construct a set of instances S that is shattered by a given class of concepts C, then $|S|$ immediately serves as a lower bound on the VC-dimension of C. The following lemma is based on such an argument. We give a method for constructing a set of instances that is shattered by $C_{n,s}$ and let the size of that set be the desired lower bound.

Lemma 2 *Let $C_{n,s}$ be as in Theorem 1. Then*

$$VCdim(C_{n,s}) \geq \max\left\{\frac{1}{8}\log_2(s-n)\log_2 n,\ s-n\right\}.$$

Before proving this lemma, we will need to make the following definitions.

Definition: Let $\alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_r$ be r Boolean variables. For any $0 \leq d \leq r$, by $\mathcal{F}_{r,d}$ let us denote the union of the following two sets:

1. The set of all Boolean functions defined on $\alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_{r-d}$.

2. The set of the d singleton Boolean functions $\alpha_{r-d+1}, \alpha_{r-d+2},\ \alpha_{r-d+3},\ \cdots,\ \alpha_r$.

Thus, the cardinality of $\mathcal{F}_{r,d}$ is just $2^{2^{r-d}} + d$. $\square$

Definition: The bit-vector $V = (v_0, v_1, \cdots, v_{t-1})^T$ is called the *bit-vector representation* of the Boolean function $f(\alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_r)$ if V matches the right-most column of the truth table of f. Boolean operations $\vee$, $\wedge$ and $\neg$ are defined on bit-vectors by applying the operations bit-wise on the components of these vectors. $\square$

It should be clear that a Boolean function f can be expressed in terms of a set of Boolean functions $g_1, g_2, \cdots, g_z$ if and only if the bit-vector that represents f can be obtained by Boolean operations on the bit-vectors that represent the functions $g_1, g_2, \cdots, g_z$.

Definition: Let f be any Boolean function defined on the variables $\alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_r$. Then, rewriting f as

$$\begin{aligned}&f(\alpha_1, \alpha_2, \cdots, \alpha_i, \cdots, \alpha_r) = \\ &\quad(\neg\alpha_i \wedge f(\alpha_1, \alpha_2, \cdots, \alpha_{i-1}, 0, \alpha_{i+1}, \cdots, \alpha_r)) \vee (\alpha_i \wedge f(\alpha_1, \alpha_2, \cdots, \alpha_{i-1}, 1, \alpha_{i+1}, \cdots, \alpha_r))\end{aligned}$$

will be called *unfolding* of the function f on the variable α_i. $\square$

Lemma 3 *For any $r \geq 1$ and any d such that $0 \leq d \leq r$, any Boolean function $f(\alpha_1, \alpha_2, \alpha_3,$ $\cdots,\ \alpha_r)$ can be expressed in terms of at most*

$$2^d + d$$

elements of $\mathcal{F}_{r,d}$.

Proof:

By unfolding f on α_r we get

$$(\neg\alpha_r \wedge f(\alpha_1, \alpha_2, \cdots, \alpha_{r-1}, 0)) \vee (\alpha_r \wedge f(\alpha_1, \alpha_2, \cdots, \alpha_{r-1}, 1))\ .$$

f is now expressed in terms of α_r and two functions defined on $\alpha_1, \alpha_2, \cdots, \alpha_{r-1}$. Unfolding these two functions again on α_{r-1}, f will be expressed in terms of α_{r-1}, α_r and four functions defined on $\alpha_1, \alpha_2, \cdots, \alpha_{r-2}$. Unfolding d times in this manner, f will be expressed in terms of $\alpha_{r-d+1}, \alpha_{r-d+2}, \alpha_{r-d+3}, \cdots, \alpha_r$ and 2^d functions defined on $\alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_{r-d}$. All of these are in $\mathcal{F}_{r,d}$, and thus the lemma follows. $\square$

Proof of Lemma 2:

Let $p = \log_2(s-n)$. We prove the lemma by constructing a set of instances that is shattered by $C_{n,s}$ and has cardinality that is both $\geq 2^p$ and $\geq \frac{1}{8}p\log_2 n$. We will represent this set of instances as a matrix M of width n, in which each row represents an instance and each column corresponds to one of the n features. In other words, the entry $M_{i,j}$ is the value of the feature x_j in the i-th instance in the collection of instances being constructed. An arbitrary labeling of these instances is viewed as a bit-vector. The proof proceeds by showing that, for any labeling of the instances, the corresponding bit-vector can be expressed using Boolean operations $(\neg, \vee, \wedge)$ over at most p out of the n columns of the matrix M. This, of course, means that for every labeling ℓ of the instances, there exists a Boolean function definable over only p features of $x_1, x_2, \cdots, x_n$ (that is, a concept in $C_{n,s}$) which labels the instances as given by ℓ. Hence, the set of instances represented by M is shattered by $C_{n,s}$

Part 1: To show that $VCdim(C_{n,s}) \geq s-n$, we construct a matrix M of n columns and $s-n = 2^p$ rows as follows. To begin, we fill the first p columns of the matrix as we ordinarily fill any truth table of a function defined over p Boolean variables. That is, the first p columns of M will look like

$$\overbrace{\begin{array}{ccc} 000 & \cdots & 000 \\ 000 & \cdots & 001 \\ 000 & \cdots & 010 \\ 000 & \cdots & 011 \\ & \cdots & \\ & \cdots & \\ 111 & \cdots & 110 \\ 111 & \cdots & 111 \end{array}}^{p}$$

The remaining columns of M (if any) can be filled arbitrarily.

Let V be a bit-vector representing an arbitrary labeling of the 2^p instances represented by M. Obviously, V represents some Boolean function defined over p variables. Therefore, V can be expressed using Boolean operations on the first p columns of M as constructed above. This shows that this set of instances is shattered by $C_{n,s}$ and hence, $VCdim(C_{n,s}) \geq 2^p = s-n$.

Part 2: Now we show that $VCdim(C_{n,s}) \geq \frac{1}{8}p\log_2 n$. In the following, we will assume that $p \geq 2$ and $n \geq 4$. Showing the lemma for $p \leq 1$ and/or $n \leq 3$ is straightforward.

Let $d = \lfloor \log_2 p \rfloor - 1$ and $r = \lfloor \log_2 p + \log_2 \log_2 n \rfloor - 2$. We construct a matrix M of n columns and 2^r rows as follows:

Let $\alpha_1, \alpha_2, , \cdots, \alpha_r$ be a set of r Boolean variables, and let $\mathcal{F}_{r,d}$ be as defined immediately prior to this proof. Construct a matrix N by letting the bit-vector representation of each function in $\mathcal{F}_{r,d}$ constitute a column of the matrix (that is, we concatenate the bit-vector representations of all the functions in $\mathcal{F}_{r,d}$). Clearly, for any function $g \in \mathcal{F}_{r,d}$ such that $g \notin \{\alpha_{r-d+1}, \alpha_{r-d+2}, \cdots, \alpha_r\}$, $\neg g$ (the complement of g) must also be in $\mathcal{F}_{r,d}$. For each such g, suppose we remove from N the column of either g or $\neg g$. (It does not matter which one is removed.) Since the cardinality of $\mathcal{F}_{r,d}$ is just $2^{2^{r-d}} + d$, the number of columns that remain in N is exactly

$$\frac{2^{2^{r-d}}}{2} + d \; . \tag{1}$$

We let these constitute the first $\frac{2^{2^{r-d}}}{2} + d$ columns in M, and fill the rest of the n columns of M (if any) arbitrarily.

We have to show the following two claims:

Claim 1: The number of columns of M does not exceed n. That is, the quantity given by Equation 1 is bounded by n.

Claim 2: An arbitrary bit-vector of length 2^r can be expressed using Boolean operations over at most p columns of M, and hence the set of instances represented by M is shattered by $C_{n,s}$.

Claim 1 is shown as follows:

$$\begin{aligned} \frac{2^{2^{r-d}}}{2} + d &= \frac{2^{2^{\lfloor \log_2 p + \log_2 \log_2 n \rfloor - 2 - \lfloor \log_2 p \rfloor + 1}}}{2} + \lfloor \log_2 p \rfloor - 1 \\ &\leq \frac{2^{2^{\log_2 p + \log_2 \log_2 n - 2 - \log_2 p + 2}}}{2} + \log_2 p - 1 \\ &= \frac{n}{2} + \log_2 p - 1 \leq n \; . \end{aligned}$$

Claim 2 is shown using Lemma 3 which says that any Boolean function defined over $\alpha_1, \alpha_2, , \cdots, \alpha_r$ can be expressed using at most $2^d + d$ functions of $\mathcal{F}_{r,d}$. Remember that M is constructed such that for any $f \in \mathcal{F}_{r,d}$, there exists a column V in M such that either V or $\neg V$ represents f. This implies that any arbitrary bit-vector of length 2^r can be expressed using Boolean operations over at most $2^d + d$ columns of M. Note that

$$2^d + d \leq 2^{d+1} \leq 2^{\log_2 p} = p$$

and thus, Claim 2 follows.

The proof is completed by noting that the number of rows in M is

$$2^r = 2^{\lfloor \log_2 p + \log_2 \log_2 n \rfloor - 2} \geq 2^{\log_2 p + \log_2 \log_2 n - 3} = \frac{1}{8} \log_2 (s-n) \log_2 n$$

as desired. □

We can now state the lower bound on the sample complexity as follows:

Theorem 3 *To learn an arbitrary concept in $C_{n,s}$ with respect to ϵ, δ and any probability distribution, any algorithm must use a sample of size*

$$\Omega \left(\frac{1}{\epsilon} \ln \frac{1}{\delta} + \frac{1}{\epsilon} \left[\ln(s-n) \ln n + s - n \right] \right) .$$

Proof: Follows immediately from Theorem 2 and Lemma 2. □

Note that this lower bound is within only a constant factor of the upper bound given by Theorem 1. These results show that the presence of many irrelevant features does not make the learning task substantially more difficult, at least in terms of the number of examples needed for learning, since the sample complexity grows only logarithmically in the number of irrelevant features.

5 Implementing the MIN-FEATURES Bias

Now that we have analyzed the sample complexity of the MIN-FEATURES bias, we turn to the problem of implementing the MIN-FEATURES bias. We first note that one can quite efficiently test whether a given subset of features is sufficient to construct a hypothesis consistent with a given set of examples, implying that the expensive part of implementing the MIN-FEATURES bias lies in the *search* for the smallest sufficient set of features. We then discuss two algorithms for implementing such search.

5.1 Testing the Sufficiency of a Subset of Features

Given a training sample S and a subset of features Q, it is easy to judge whether Q is sufficient to construct an hypothesis consistent with S. All we need to do is search in the sample for two examples that have the same values for all the features in Q but have conflicting classes. The subset Q is sufficient to construct a consistent hypothesis if and only if such a pair cannot be found in the training sample. This procedure runs in time quadratic in the number of examples in the sample.

A more efficient sufficiency test can be done by performing successive partitioning of the the training sample based on a feature arbitrarily chosen from Q each time (see Figure 1). A subset in which all the examples are of the same class does not need to be partitioned (Step 1). Insufficiency is reported if and only if we exhaust Q and there still exists a subset containing a mixture of positive and negative examples (Step 6).

Note that to perform partitioning using a feature at any stage, we need to visit each example at most once. The total running time for this procedure is therefore $O(|S||Q|)$, that is, only linear.

5.2 The Set Covering Viewpoint

The task of finding the smallest sufficient subset of features for a given training sample can be viewed as a set covering problem. For convenience, let us call this task the "MIN-FEATURES problem". To link this problem to the minimum set cover problem we make the following definition.

Definitions: A *conflict* generated by a pair of examples $\langle X_1, +\rangle$ and $\langle X_2, -\rangle$ is an n-bit vector $a = \langle a_1 a_2 \cdots a_n \rangle$ such that

$$a_i = \begin{cases} 0 & \text{if } X_1^{(i)} = X_2^{(i)} \\ 1 & \text{if } X_1^{(i)} \neq X_2^{(i)} \end{cases}$$

for $i = 1, 2, \cdots, n$, where $X^{(i)}$ denotes the value of the i-th feature in the instance X.

We say that the feature x_i *covers* the conflict a if and only if a_i is 1. A set of features Q covers a set of conflicts A if every conflict in A is covered by at least one feature in Q.

It should be noted that a conflict is generated only from a pair of examples of *conflicting* classes. □

With this definition, the analogy between the minimum set covering and the MIN-FEATURES problems should be clear. Given a training sample, we can generate the set of all conflicts from these examples. Each feature covers a subset of the generated conflicts, and the goal is to find the smallest subset of features that together cover all the conflicts.[1]

Example: Let the training sample be

$$\begin{array}{ll} \langle 010100, + \rangle & \langle 011000, - \rangle \\ \langle 110010, + \rangle & \langle 101001, - \rangle \\ \langle 101111, + \rangle & \langle 100101, - \rangle \end{array}$$

Then, the conflicts generated from this sample are

$$\begin{array}{lll} a_1 = \langle 001100 \rangle & a_4 = \langle 101010 \rangle & a_7 = \langle 110111 \rangle \\ a_2 = \langle 111101 \rangle & a_5 = \langle 011011 \rangle & a_8 = \langle 000110 \rangle \\ a_3 = \langle 110001 \rangle & a_6 = \langle 010111 \rangle & a_9 = \langle 001010 \rangle \end{array}$$

It can be checked that a subset such as $\{x_1, x_3, x_4\}$ is sufficient to form a consistent hypothesis (e.g., $\bar{x}_1 \bar{x}_3 \vee (\bar{x}_3 \oplus x_4)$), and that all subsets of features of cardinality less than 3 are insufficient. □

The minimum set cover problem is known to be NP-hard [10]. At first glance, this may appear to mean that it is not possible to implement the MIN-FEATURES bias in polynomial time (unless P=NP). However, one should be careful before drawing such a conclusion. The NP-hardness result for the minimum set cover problem means that if the number of the available sets is n and the number of elements to be covered is ℓ, then one cannot find an optimal solution in time polynomial in n and ℓ, unless P=NP. The usual practice in computational learning theory is to include s, the complexity of the target concept, among the parameters to measure the computational and sample complexities of learning algorithms. For a target concept of p relevant features out of n available features, s is equal to $2^p + n$. Therefore, assuming that p is the size of the smallest cover, we are concerned with the question of finding an optimal solution for the minimum set cover problem in time polynomial in n, ℓ and 2^p.

To our knowledge, there are no known complexity results of this type for the minimum set cover problem, and thus, whether or not the MIN-FEATURES bias can be implemented in time polynomial in the concept complexity and other learning parameters remains an interesting open problem.

In the rest of the paper, we describe a collection of algorithms that implement and approximate the MIN-FEATURES bias.

5.3 The FOCUS Algorithms

In Subsection 5.1, we have seen that one can efficiently test whether a given subset of features is sufficient to construct an hypothesis consistent with a training sample. Since it is assumed that the target concept has relatively few relevant features, a straightforward algorithm to solve the MIN-FEATURES problem is to try all subsets of features of size 0, 1, 2, 3, $\cdots$ and so on. Because the subsets are tested in order of increasing size, this is guaranteed to produce the smallest sufficient set of features. We will call this procedure FOCUS-1.

Suppose this algorithm is given m examples of a target concept that has p relevant features, and thus, has complexity $s = 2^p + n$. Then, the algorithm will be examining all the subsets of features of size 0, 1, 2, 3 $\cdots$ up to at most p. Therefore, the number of sufficiency tests performed will be at most

$$\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots\cdots + \binom{n}{p} = O(n^p)\ .$$

As shown in Subsection 5.1, each sufficiency test costs time $O(mp)$, and thus, FOCUS-1 has a worst-case time complexity of $O(mpn^p)$. In terms of s, this gives $O(m \log(s-n) n^{\log(s-n)})$. This is quasi-polynomial in n and s, but clearly it will be impractical for large values of n and s.

The major computational cost of the above simple algorithm stems from the rather *blind* search it performs in the space of feature subsets. Consider again the example given in Subsection 5.2. In this example, FOCUS-1 tests all the 22 subsets of features of size 0, 1 and 2, and some of the 20 subsets of size 3 before returning a solution. By doing so, FOCUS-1 is not exploiting all the information given in the training sample. Consider, for instance, the conflict $a_1 = \langle 001100 \rangle$. This conflict tells us that any sufficient set of features must contain x_3 or x_4 in order to cover the conflict. Hence, none of the sets $\{x_1\}$, $\{x_2\}$, $\{x_5\}$, $\{x_6\}$, $\{x_1, x_2\}$, $\{x_1, x_5\}$, $\{x_1, x_6\}$, $\{x_2, x_5\}$, $\{x_2, x_6\}$, $\{x_5, x_6\}$ can be solutions. Therefore, all of these sets can immediately be ruled out of the algorithm's consideration. Many other subsets can be similarly ruled out based on the other conflicts.

Figure 2 shows the FOCUS-2 algorithm, which takes advantage of this observation. In this algorithm, we use a first-in-first-out queue in which each element denotes a subspace of the space of all feature subsets. Each element has the form $M_{A,B}$, which denotes the space of all feature subsets that include all of the features in the set A and none of the features in the set B. Formally,

$$M_{A,B} = \{T \mid T \supseteq A, T \bigcap B = \phi, T \subseteq \{x_1, x_2, \cdots, x_n\}\}.$$

For example, the set $M_{\phi,\phi}$ denotes all possible feature subsets, the set $M_{A,\phi}$ denotes all feature subsets that contain at least all the features in A, and the set $M_{\phi,B}$ denotes all feature subsets that do not contain any features in B.

The main idea of FOCUS-2 is to keep in the queue only the *promising* portions of the space of feature subsets—i.e. those that *may* contain a solution. Initially, the queue contains only the element $M_{\phi,\phi}$, which represents the whole power set. In each iteration in Step 4, the space represented by the head of the queue is partitioned into disjoint subspaces, and those subspaces that cannot contain solutions are pruned from the search.

Consider again the conflict $a_1 = \langle 001100 \rangle$. Suppose the current space of possible feature subsets is $M_{\phi,\phi}$. We know that any sufficient feature subset must contain either x_3 or x_4. We can incorporate this knowledge into the search by refining $M_{\phi,\phi}$ into the two subspaces $M_{\{x_3\},\phi}$ (all feature subsets that contain x_3) and $M_{\{x_4\},\{x_3\}}$ (all feature subsets that contain x_4 and do not contain x_3). Note that the second subscript of M is used to keep the various subspaces disjoint.

Clearly, conflicts with fewer 1's in them provide more constraints for the search than conflicts with more 1's. Hence, if the head of the queue is $M_{A,B}$, then the algorithm (Step 4.3) searches for a conflict a such that

- a is not covered by any of the features in A, and
- the number of 1's corresponding to features that are not in B is minimized.

This conflict is then incorporated in the search in Step 4.4.

In detail, here is how FOCUS-2 behaves on the example given above. As shown in Figure 3, the algorithm starts by processing $M_{\phi,\phi}$. The conflict $a_1 = \langle 001100 \rangle$ is selected in Step 4.3 and $M_{\phi,\phi}$ is replaced by $M_{\{x_3\},\phi}$ and $M_{\{x_4\},\{x_3\}}$. Next, for $M_{\{x_3\},\phi}$, the conflict $a_8 = \langle 000110 \rangle$ is selected, and $M_{\{x_3,x_4\},\phi}$ and $M_{\{x_3,x_5\},\{x_4\}}$ are added to the queue. $M_{\{x_4\},\{x_3\}}$ is then processed with $a_9 = \langle 001010 \rangle$ and $M_{\{x_4,x_5\},\{x_3\}}$ is inserted. Finally, when $M_{\{x_3,x_4\},\phi}$ is processed with $a_3 = 110001$, the algorithm terminates in Step 4.4.1 before adding $M_{\{x_1,x_3,x_4\},\phi}$ to the queue, since $\{x_1, x_3, x_4\}$ is a solution.

Using FOCUS-2, the number of sufficiency tests is only 7. By comparison, FOCUS-1 must perform at least 23 sufficiency tests (to test each of the 22 subsets of size up to 2, and at least one of the 20 subsets of size 3). Because FOCUS-2 only prunes subspaces that cannot possibly cover all of the conflicts, it is sound and complete—it will not miss any sufficient feature subsets. Furthermore, because it considers the subspaces $M_{A,B}$ in order of increasing size of A, it is guaranteed to find a sufficient subset with the smallest possible size. Finally, of course, the number of sufficiency tests performed by FOCUS-2 will typically be much less (and certainly never more) than the number of tests performed by FOCUS-1. This makes FOCUS-2 substantially faster than FOCUS-1, as will be demonstrated by the execution-time comparisons reported in Subsection 7.2.

6 Approximating the MIN-FEATURES Bias

Exact implementation of the MIN-FEATURES bias in domains with many features can be computationally infeasible. In such cases, one may be willing to employ efficient heuristics that provide good but not necessarily optimal solutions. In this section, we describe three such algorithms. Each of these algorithms implements an iterative procedure where, in each iteration, the feature that seems most promising is added to the partial solution. This continues until a sufficient set of features is found. The only difference between the three algorithms is the criterion used in selecting the best feature in each iteration.

6.1 The Mutual-Information-Greedy (MIG) Algorithm

For a given set of features Q, imagine that the training sample is partitioned into $2^{|Q|}$ groups such that the examples in each group have the same truth assignment to the features in Q. Let p_i and n_i denote the number of positive and negative examples in the i-th group, respectively, and define the *entropy* of Q as

$$Entropy(Q) = -\sum_{i=0}^{2^{|Q|}-1} \frac{p_i + n_i}{|Sample|}\left[\frac{p_i}{p_i+n_i}\log_2\frac{p_i}{p_i+n_i} + \frac{n_i}{p_i+n_i}\log_2\frac{n_i}{p_i+n_i}\right] \quad (2)$$

with the convention that $a\log_2 a = 0$ when $a = 0$.

In the Mutual-Information-Greedy algorithm, the feature that leads to the minimum entropy when added to the current partial solution is selected as the best feature. To implement this algorithm, one needs to avoid explicit partitioning of the training sample into $2^{|Q|}$ groups. We explain how this can be done in the appendix.

6.2 The Simple-Greedy (SG) Algorithm

The Simple-Greedy algorithm is based on the well-known greedy heuristic for the minimum set cover problem [8]. Starting with the set of all conflicts, the algorithm chooses each time the feature that covers the largest number of conflicts that are not yet covered. The conflicts that are covered by this feature are then removed from the set of conflicts. The process is repeated until all conflicts are removed.

To implement the Simple-Greedy algorithm, however, one should avoid explicit generation of the set of all conflicts (which requires time quadratic in the number of examples). A detailed discussion is given in the appendix.

6.3 The Weighted-Greedy (WG) Algorithm

In the Simple-Greedy algorithm, every conflict contributes a unit increment to the score of each feature that covers it. In the Weighted-Greedy algorithm, the increment instead depends on the total number of features that cover the conflict. The intuition is that if a feature uniquely covers a conflict, then that feature *must* be part of the solution set of features. If A_{x_i} is the set of conflicts covered by a feature x_i, then the score of x_i is computed as

$$score_{x_i} = \sum_{a \in A_{x_i}} \frac{1}{\#\text{ of features covering } a \ - 1}.$$

Under this heuristic, the *credit* a feature x_i receives for covering a conflict a, is inversely proportional to the number of *other* features that cover a. If only a few other conflicts cover a then x_i receives high credit for covering a. In the extreme case where a is exclusively covered by x_i, the score of x_i becomes ∞. This causes the feature to be included in the solution with certainty.

7 Experimental Results

In this section, we compare our exact (FOCUS-1 and FOCUS-2) and approximate (MIG, SG, and WG) algorithms to two well-known algorithms, ID3 and FRINGE. The goal of the comparisons is to determine whether our algorithms show better generalization performance than ID3 and FRINGE in situations where the MIN-FEATURES bias is appropriate. A second goal of the comparisons is to determine how well the three heuristic algorithms approximate the exact algorithms.

We chose ID3 and FRINGE because these algorithms appear to have biases similar to the MIN-FEATURES bias. In particular, these algorithms attempt to construct "small" decision trees. The algorithms construct the decision tree top-down (i.e., starting at the root), and they terminate as soon as they find a tree consistent with the training examples. Features tested at each node are chosen according to their *estimated relevance* to the target concept, measured using the mutual information criterion. Obviously a tree with k internal nodes can test no more than k features, and hence, it is reasonable to expect that ID3 and FRINGE will perform well in cases where MIN-FEATURES is appropriate.

It should be noted that FOCUS-1, FOCUS-2, MIG, SG, and WG are not complete learning algorithms—rather they are preprocessors that provide us only with a set of features sufficient to construct a consistent hypothesis. To construct an actual hypothesis, we first filter the training examples to remove all features not selected during preprocessing. Then we give the filtered examples to ID3 to construct a decision tree. Hence, we will refer to the complete algorithms as FOCUS+ID3, MIG+ID3, SG+ID3, and WG+ID3. In most of the experiments, we do not distinguish between FOCUS-1 and FOCUS-2, because they give identical results. Our version of ID3 performs no windowing or pruning and employs the information gain (mutual information) criterion to select features. Our version of FRINGE is terminated after at most 10 iterations.

7.1 Sample Complexity and Accuracy Comparisons

We report here two kinds of experiments. In the first experiment, we are interested in the *worst-case* performance over a class of concepts, where each concept is definable over at most

p out of n features (and hence, the MIN-FEATURES bias is appropriate). As our measure of performance, we employ the sample complexity—the minimum number of training examples needed to ensure that every concept in the class can be learned in the PAC sense [6]. We estimate the sample complexity with respect to fixed learning parameters p, n, ϵ, and δ and with training samples drawn according to the uniform distribution.

In the second experiment, we are interested in the *average-case* performance of the algorithms. We randomly generate a collection of concepts such that each involves only a few features among many available ones. We then measure the accuracy rate of each algorithm while progressively increasing the size of the training sample—i.e. by plotting the learning curve for each of the concepts under consideration.

EXPERIMENT 1: Sample Complexity. The goal of this experiment is to estimate the minimum number of examples that enables each algorithm to PAC learn all the concepts of at most 3 relevant features out of n available features for $n = 8, 10$ and 12. We assume the uniform distribution over the space of instances. To decide whether an algorithm L learns a concept c for sample size m, we generate 100,000 random samples of c of size m. We conclude that c is learned by L if and only if for at least 90% of these samples L returns a hypothesis that is at least 90% correct.[2]

The minimum value of m that enables each algorithm to learn all the concepts under consideration is reported. Thus, the quantity measured here can be viewed as an empirical estimate of the *sample complexity* of each algorithm under the uniform distribution for $\epsilon = \delta = 0.1$.

The results of this experiment are shown in Figure 4.

EXPERIMENT 2: Learning Curve. The purpose of this experiment is to perform a kind of "average-case" comparison between the algorithms. The experiment is conducted as follows. First, we randomly choose a concept such that it has only a few relevant features among many available ones. We then run each of the algorithms on randomly-drawn training samples of this concept and plot the accuracy of the hypothesis (i.e., the percentage of the examples correctly classified by the hypothesis) returned by each algorithm against the training sample size. This is repeated for various sample sizes for the same concept.

The above procedure was applied on 100 randomly selected concepts each having at most 5 relevant features out of 16 available features. For each of these concepts, the sample size m was varied from 20 to 120 examples. For each value of m, the accuracy rate was averaged over 100 randomly drawn training samples.

Figure 5 shows a pattern typical of all learning curves that we observed.

As a way to combine the results of the 100 concepts, we have measured the difference in accuracy between FOCUS+ID3 and each of the other algorithms for each sample size, and averaged that over all 100 target concepts. The result is shown in Figure 6.

DISCUSSION: The performance of the algorithms tested in the above experiments can be summarized as follows:

1. Each of the three heuristics improved the performance of ID3 when learning in the presence of irrelevant features.

2. Weighted-Greedy gave the overall best approximation to the MIN-FEATURES bias. The performance of this algorithm was quite close to that of FOCUS both in the worst and average cases.

3. Mutual-Information-Greedy and Simple-Greedy are very much alike. These algorithms maintained reasonable average-case performance, but exhibited rather bad worst-case performance.

4. Finally, FRINGE showed poor average-case performance, but its worst-case performance is almost as good as Weighted-Greedy and substantially better than Mutual-Information-Greedy and Simple-Greedy. In terms of the computational costs, however, FRINGE is much more expensive than any of the three heuristics considered here.

7.2 Execution-Time Comparisons

In this subsection, we compare the computational costs of FOCUS-1, FOCUS-2 and Weighted-Greedy measured as the number of sufficiency tests and the amount of CPU-time required by each algorithm to return a solution. The three algorithms were implemented in C. Special attention was given to optimizing the implementation of FOCUS-1.

The experiments were conducted using target concepts that have only a few relevant features out of many available. The relative performance of the algorithms was greatly affected by the problem size measured as the number of the available features and the ratio of the relevant features to that number. However, when the problem size was reasonably large, the relative performance followed a consistent trend. This trend is illustrated by Table I where we give the result for a target concept with 9 relevant features out of 25 available features. Training examples were drawn with replacement under the uniform distribution and the training sample size was varied from 100 to 500. The numbers in this table are averaged over 10 runs for each training sample size.

Overall, we found that FOCUS-2 was several times faster than FOCUS-1 and that Weighted-Greedy was further many times faster than FOCUS-2. It is interesting to note that the number of sufficiency tests done by FOCUS-1 remains steady as the training sample size grows, since it blindly follows the same steps for any training sample. FOCUS-2, on the other hand, does a progressively smaller number of sufficiency tests as the number of training examples increases. This is because with a larger sample there is a greater chance of getting conflicts that are covered by only a few features, and consequently, a better chance for significant reduction in the number of sufficiency tests needed by FOCUS-2.

8 Summary and Conclusion

This paper defined and studied the MIN-FEATURES bias. Section 4 presented a tight bound on the number of examples needed to guarantee PAC-learning for any algorithm that implements MIN-FEATURES. Section 5 introduced two algorithms for implementing the

MIN-FEATURES bias. The first algorithm, FOCUS-1, is a straightforward algorithm that runs in time quasi-polynomial in the total number of features and the complexity of the target concept. The second algorithm, FOCUS-2, is empirically shown to be substantially faster than FOCUS-1. Section 6 introduced the algorithms Mutual-Information-Greedy, Simple-Greedy and Weighted-Greedy for approximating the MIN-FEATURES bias. Experiments of Section 7 showed that the Weighted-Greedy algorithm in particular provides excellent approximation of the exact FOCUS algorithms but with substantially less computational costs. The experiments also showed that both ID3 and FRINGE do not perform nearly as well in problems where the MIN-FEATURES bias is appropriate. These results suggest that one should not rely on ID3 or FRINGE to filter out irrelevant features. Instead, in situations where the MIN-FEATURES bias is appropriate, it is recommended that the Weighted-Greedy algorithm be applied to preprocess the training examples before invoking a decision-tree learning algorithm, such as ID3.

We conclude with the following open problems:

- Is the class $C_{n,s}$ polynomially learnable in the sense of [7]? Namely, does there exist a learning algorithm L such that, under any probability distribution, L PAC-learns with respect to parameters ϵ and δ an arbitrary concept in $C_{n,s}$ using a sample of size polynomial in $n, s, \frac{1}{\epsilon}$ and $\frac{1}{\delta}$, and such that L runs in time polynomial in $n, s, \frac{1}{\epsilon}$ and $\frac{1}{\delta}$?

  Note that this does not necessarily require that L implement the MIN-FEATURES bias *strictly*. For example, L could be a polynomial time algorithm that approximates the MIN-FEATURES bias quite closely so that its sample complexity is still polynomial. It is easy to show that the approximation algorithms given in this paper run in polynomial time. An important goal for future research is to prove formal results on the sample complexity of these and similar algorithms as an approach to answering the above question.

- The work reported in this paper assumes noise-free training data. In practice, however, training examples are often subject to various kinds of noise affecting the values of the features and/or the classification of the training examples [13, 15]. A direct way to deal with noise is to modify the given algorithms by relaxing the requirement of covering *all* the conflicts generated from the training data. That is, we search for a small set of features that may leave a certain percentage of the conflicts uncovered, where such percentage can be determined through cross-validation. Studying this and more sophisticated approaches to dealing with noise is an important topic for future research.

- A major limitation of this work is that it only applies to Boolean concepts. Many practical domains have multi-valued (discrete) features and continuous features as well as multiple classes. Extension of the methods in this paper to multi-valued discrete features and multiple classes is straightforward. However, extending the MIN-FEATURES bias to continuous features requires a different definition of "relevant feature".

Acknowledgement

This work was carried out at Oregon State University. The paper was revised while the first author was at NTT Network Information Systems Laboratories, Japan. The second author gratefully acknowledges the support of the National Science Foundation under Grant Number IRI-8657316.

References

[1] H. Almuallim, Exploiting Symmetry Properties in the Evaluation of Inductive Learning Algorithms: An Empirical Domain-Independent Comparative Study, Technical Report, 91-30-09, Dept. of Computer Science, Oregon State University, Corvallis, Oregon (1991).

[2] H. Almuallim and T. G. Dietterich, Learning With Many Irrelevant Features, in: *Proceedings AAAI-91,* Anaheim, California (1991) 547–552.

[3] H. Almuallim, Concept Coverage and Its Application to Two Learning Tasks, Ph.D. Thesis, Oregon State University, Corvallis, Oregon (1992).

[4] H. Almuallim and T. G. Dietterich, Efficient Algorithms for Identifying Relevant Features, in: *Proceedings of the Ninth Canadian Conference on Artificial Intelligence,* Vancouver, British Columbia (1992) 38–45.

[5] W. L. Buntine, Myths and legends in learning classification rules, in: *Proceedings AAAI-90,* Boston, Massachusetts (1990) 736–742.

[6] A. Blumer, A. Ehrenfeucht, D. Haussler and M. Warmuth, Learnability and the Vapnik-Chervonenkis Dimension, *J. ACM,* 36(4) (1989) 929–965.

[7] A. Blumer, A. Ehrenfeucht, D. Haussler and M. Warmuth, Occam's Razor, *Information Processing Letters,* 24 (1987) 377–380.

[8] V. Chvatal, A Greedy Heuristic For the Set Covering Problem, *Math. Oper. Res.*, 4(3) (1979) 233–235. Vol. 4, No. 3, 1979.

[9] A. Ehrenfeucht, D. Haussler, M. Kearns and L. G. Valiant, A General Lower Bound on the Number of Examples Needed for Learning, in: *Proceedings of the First Workshop on Computational Learning Theory* (Morgan Kaufmann, San Mateo, CA, 1988) 139–154.

[10] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, CA, 1979).

[11] M. Ichino, and J. Sklansky, Optimum Feature Selection by Zero-One Integer Programming, *IEEE Trans. Sys. Man & Cyb.,* SMC-14(5) (1984) 737–746.

[12] M. Ichino, and J. Sklansky, Feature Selection for Linear Classifiers, in: *The Seventh International Conference on Pattern Recognition,* (1984) 124–127.

[13] M. Kearns, and M. Li, Learning in the Presence of Malicious Errors, in: *Proceedings of the 20th Annual ACM Symposium on the Theory of Computation*, Chicago, Illinois (1988) 267–279.

[14] J. Kittler, Computational Problems of Feature Selection Pertaining to Large Data Sets, in: E. S. Gelsma and L. N. Kanal, eds., *Pattern Recognition in Practice,* (North-Holland, Amsterdam, 1980) 405–414.

[15] P. Laird, *Learning from Good and Bad Data* (Klawer Academic, Boston, Massachusetts, 1988).

[16] N. Littlestone, Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm, *Machine Learning,* 2 (1988) 285–318.

[17] T. M. Mitchell, Generalization as Search. *Artif. Intell.*, 18 (1982) 203-226.

[18] S. D. Morgera, Computational Complexity and VLSI Implementation of an Optimal Feature Selection Strategy, in: E. S. Gelsma and L. N. Kanal, eds., *Pattern Recognition in Practice II* (Elsevier Science Publishers B.V., North-Holland, 1986) 389–400.

[19] A. N. Mucciardi and E. E. Gose, A Comparison of Seven Techniques for Choosing Subsets of Pattern Recognition Properties, *IEEE Trans. Computers,* C-20(9) (1971) 1023–1031.

[20] P. M. Narendra and K. Fukunaga, A Branch and Bound Algorithm for Feature Subset Selection, *IEEE Trans. Computers,* C-26(9) (1977) 917–922.

[21] G. Pagallo and D. Haussler, Boolean Feature Discovery in Empirical Learning, *Machine Learning*, 5(1) (1990) 71–100.

[22] C. E. Queiros and E. S. Gelsma, On Feature Selection, in: *Proceedings of the Seventh International Conference on Pattern Recognition,* (1984) 128–130.

[23] J. R. Quinlan, Induction of Decision Trees, *Machine Learning,* 1(1) (1986) 81–106.

[24] D. Wolpert, A Mathematical Theory of Generalization: Parts I and II, *Complex Systems,* 4(2) (1990) 151–249.

Appendix

In Section 6, we gave sketches of the Mutual-Information-Greedy and the Simple-Greedy algorithms. These sketches are only meant to explain how those algorithms work. We describe here more efficient ways to implement the two algorithms.

In the description of the Mutual-Information-Greedy algorithm in Section 6, the entropy function for a subset Q is computed explicitly by partitioning the sample in each iteration. To be more efficient, one should exploit the following:

- Features are progressively added to the current solution Q. Once a feature is added to Q, it is never removed. Therefore, the splitting information from one iteration should be kept to be used for computing $Entropy(Q \bigcup \{x_i\})$ in the succeeding iteration.

- If the examples in the i-th group of the splitted sample are all positive or all negative (i.e., if $p_i n_i = 0$.), then this group does not contribute to the summation in equation (2) since in this case
$$\frac{p_i}{p_i + n_i} \log_2 \frac{p_i}{p_i + n_i} = 0$$
and
$$\frac{n_i}{p_i + n_i} \log_2 \frac{n_i}{p_i + n_i} = 0 \ .$$
This is also true if the group is empty. As a result, one should ignore any group that is empty or having examples that are all of the same class.

Now, consider the following alternative implementation:

Algorithm: Mutual-Information-Greedy($Sample$)
1.$Q = \phi$, $V = \{x_1, x_2, x_3, \cdots, x_n\}$, $\mathcal{S} = \{Sample\}$.
2.Repeat until $\mathcal{S}$ is empty:
  2.1. $Best\text{-}Score = \infty$.
  2.2. For each $x_i \in V$:
    2.2.1. $Score = 0$.
    2.2.2. For each $s \in \mathcal{S}$:
      2.2.2.1. $p_0 = \#$ of positive examples in s with $x_i = 0$.
      2.2.2.2. $n_0 = \#$ of negative examples in s with $x_i = 0$.
      2.2.2.3. $p_1 = \#$ of positive examples in s with $x_i = 1$.
      2.2.2.4. $n_1 = \#$ of negative examples in s with $x_i = 1$.
      2.2.2.5. $e_0 = \frac{p_0}{p_0+n_0} \log_2 \frac{p_0}{p_0+n_0} + \frac{n_0}{p_0+n_0} \log_2 \frac{n_0}{p_0+n_0}$.
      2.2.2.6. $e_1 = \frac{p_1}{p_1+n_1} \log_2 \frac{p_1}{p_1+n_1} + \frac{n_1}{p_1+n_1} \log_2 \frac{n_1}{p_1+n_1}$.
      2.2.2.7. $Score(x_i) = Score(x_i) - \frac{p_0+n_0}{|\mathrm{Sample}|}[e_0 + e_1]$.
    2.2.3 If $Score(x_i) < Best\text{-}Score$ then
      2.2.3.1 $Best\text{-}Feature = x_i$.
      2.2.3.2 $Best\text{-}Score = Score$.

2.3. For each $s \in \mathcal{S}$:
  2.3.1. Partition s into s_0 and s_1, which are the sets of examples with *Best-Feature* $= 0$ and 1, respectively.
  2.3.2. Replace s in $\mathcal{S}$ by s_0 and s_1. However, if any of s_0 and s_1 is empty or contains only examples of the same class, then it should not be added to $\mathcal{S}$.
2.4. Remove *Best-Feature* from V.
2.5. Add *Best-Feature* to Q.
3.Return Q.

In this implementation, $\mathcal{S}$ is a set of sets that contains the splitting information of the training sample. Initially, $\mathcal{S}$ has only one element which is the training sample itself, and the current partial solution Q is empty. Then, whenever a new feature x_i is added to Q, each set s in $\mathcal{S}$ is partitioned into two sets s_0 and s_1. These are the sets of training examples in s that have $x_i = 0$ and 1, respectively. The element s is then replaced in the set $\mathcal{S}$ by s_0 and s_1. However, we neglect any of these two sets if it is empty or if it contains examples that are all of the same class. In this way, at any particular point of the algorithm, the set $\mathcal{S}$ contains only non-empty sets that have a mixture of positive and negative training examples. These are the only sets needed for entropy computation. The reader can confirm that the score of a feature x_i (as computed in Step 2.2.2.7 of the algorithm) is equivalent to the quantity *Entropy*$(Q \bigcup \{x_i\})$ when the loop of Step 2.2.2 terminates.

The Simple-Greedy algorithm can also be implemented more efficiently than described in Section 6 in a similar manner. Note that computing the score of a feature in this algorithm does not require the knowledge of the conflicts covered by the feature but only the *number* of these conflicts. For a training sample of a large size, constructing a list of all conflicts is rather expensive since it takes time quadratic in the number of examples in the sample. On the other hand, counting the number of conflicts generated from a given set of examples can be done in linear time since all what we need to do is to compute the number of positive and negative examples in the set and multiply these.

With this observation, we can compute the score of each feature as follows. Suppose that Q is the set of features that have been selected so far. For each of the $2^{|Q|}$ truth assignments to the features in Q, the training sample can be partitioned into $2^{|Q|}$ groups such that all the examples in the i-th group have the i-th truth assignment of the features in Q. Clearly, any conflict that is generated from two examples in two different groups must be covered by some feature in Q, and thus, the conflicts that are *yet to be covered* are only those generated from pairs of examples that belong to the same group. Therefore, to compute the score of a feature x_j, we only need to find the number of conflicts that remain uncovered after adding x_j to Q. This can be computed as follows:

$$score_{x_j} = \sum_{i=0}^{2^{|Q|}-1} p_i^{x_j=0} \cdot n_i^{x_j=0} + p_i^{x_j=1} \cdot n_i^{x_j=1} \qquad (3)$$

where $p_i^{x_j=a}$ and $n_i^{x_j=a}$ denote the number of positive and negative examples in the i-th group such that the value of the feature x_j in each example is a, for $a = 0$ or 1.

The above score computation can be achieved simply by replacing the steps 2.2.2.5 to 2.2.2.7 by the following:

$$2.2.2.5 \quad Score(x_i) = Score(x_i) + p_0 n_0 + p_1 n_1 \ .$$

Note that the feature with the least score (i.e., the one that leaves the least number of conflicts uncovered) is selected.

Texts of the footnotes

1. This description reduces the MIN-FEATURES problem to the minimum set cover problem. Reduction in the opposite direction is given in [3].

2. We have applied several techniques to reduce the immense amount of computation involved in this experiment. First, we exploited the fact that the algorithms are symmetric with respect to the permutation and negation of any subset of the features of the target concept [1]. Second, we started the experiment by running the algorithms on a reasonable number (a few thousands) of training samples per concept to get rough estimates on the sample complexity. These estimates were then refined by increasing the number of samples to 100,000, considering only the values in the neighborhood of the obtained estimates.

Figure Captions

Figure 1: Efficient implementation of sufficiency testing.

Figure 2: The FOCUS-2 algorithm.

Figure 3: An example of FOCUS-2. Rectangles indicate where the sufficiency tests occurred. The tree is traversed in a breadth-first fashion.

Figure 4: The number of examples needed for learning all the concepts with 3 relevant features out of 8, 10, and 12 available features. ID3 requires 2236 examples when the total number of features is 12.

Figure 5: Learning curve for the concept $f(x_1, \ldots, x_{16}) = x_1 x_2 x_3 \bar{x_4} \vee x_1 x_2 x_3 x_4 \bar{x_5} \vee x_1 x_2 \bar{x_3} x_4 x_5 \vee x_1 \bar{x_2} x_3 \vee x_1 \bar{x_2} \bar{x_3} \bar{x_4} \vee \bar{x_1} x_2 x_3 x_4 x_5 \vee \bar{x_1} \bar{x_2} x_3 x_4 x_5 \vee \bar{x_1} \bar{x_2} x_3 \bar{x_4} \vee \bar{x_1} \bar{x_3} x_4 x_5 \vee \bar{x_1} \bar{x_3} \bar{x_4} \bar{x_5}$ which has 5 relevant features out of 16.

Figure 6: The difference between the accuracy of FOCUS+ID3 and the accuracy of the other algorithms averaged over all the randomly chosen 100 target concepts.

Function Sufficient(*Features, Sample*)

1. If the examples in *Sample* are either all positive or all negative, return *true*.
2. If *Features* is empty return *false*.
3. Let x be any feature in *Features*.
4. Let *Sample0* be the set of all examples in which the value of the feature x is 0.
5. Let *Sample1* be the set of all examples in which the value of the feature x is 1.
6. If the results of

   Sufficient(*Features*$-\{x\}$, *Sample0*), and

   Sufficient(*Features*$-\{x\}$, *Sample1*)

   are both *true*, then return *true*, else return *false*.

Figure 1: Efficient implementation of sufficiency testing.

Algorithm FOCUS-2(*Sample*)
1. If all the examples in *Sample* have the same class, return ϕ.
2. Let G be the set of all conflicts generated from *Sample*.
3. $Queue = \{M_{\phi,\phi}\}$. /* This is a first-in-first-out data structure. */
4. Repeat
   4.1. Pop the first element in *Queue*. Call it $M_{A,B}$.
   4.2. Let $OUT = B$.
   4.3. Let a be the conflict in G not covered by any of the features in A, such that $|Z_a - B|$ is minimized, where Z_a is the set of features covering a.
   4.4. For each $x \in Z_a - B$
      4.4.1. If Sufficient($A \bigcup \{x\}$), return ($A \bigcup \{x\}$).
      4.4.2. Insert $M_{A \bigcup \{x\}, OUT}$ at the tail of *Queue*.
      4.4.3. $OUT = OUT \bigcup \{x\}$.

end.

Figure 2: The FOCUS-2 algorithm.

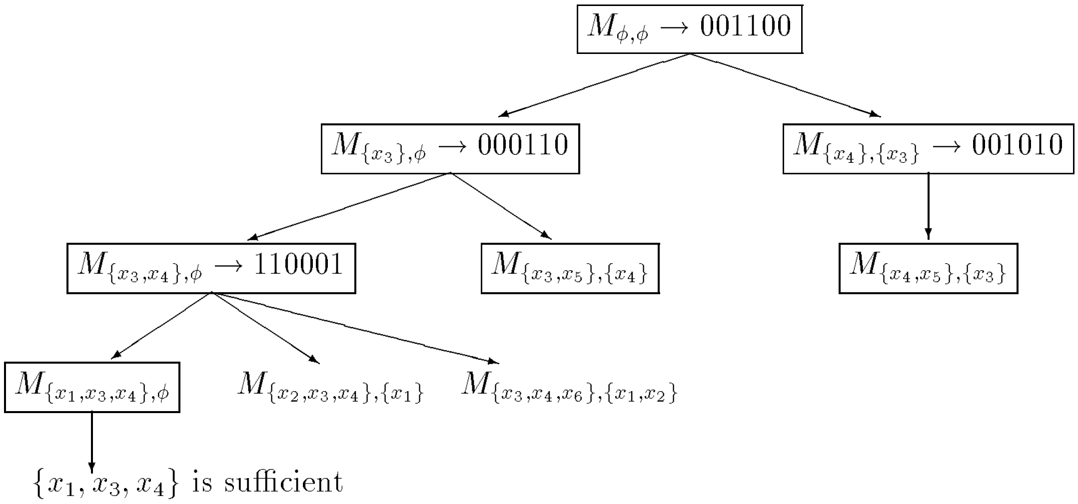

Figure 3: An example of FOCUS-2. Rectangles indicate where the sufficiency tests occurred. The tree is traversed in a breadth-first fashion.

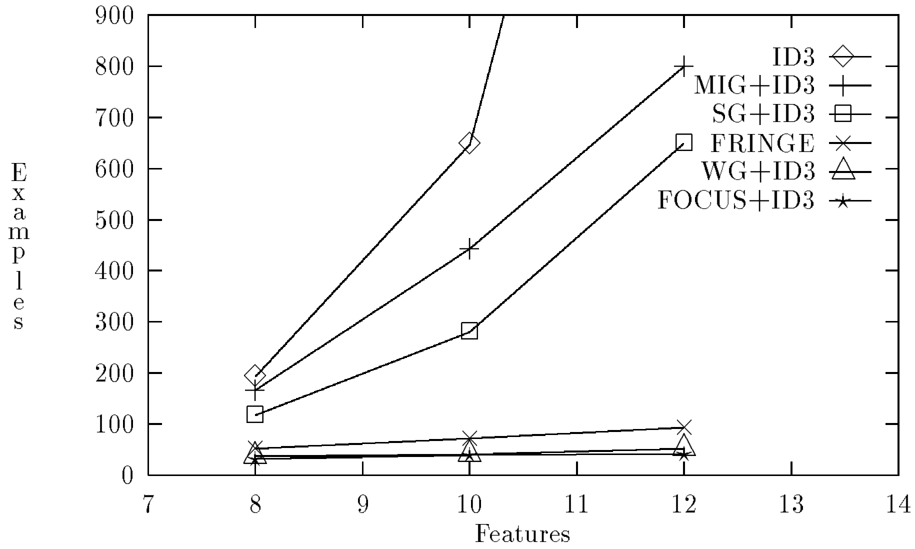

Figure 4: The number of examples needed for learning all the concepts with 3 relevant features out of 8, 10, and 12 available features. ID3 requires 2236 examples when the total number of features is 12.

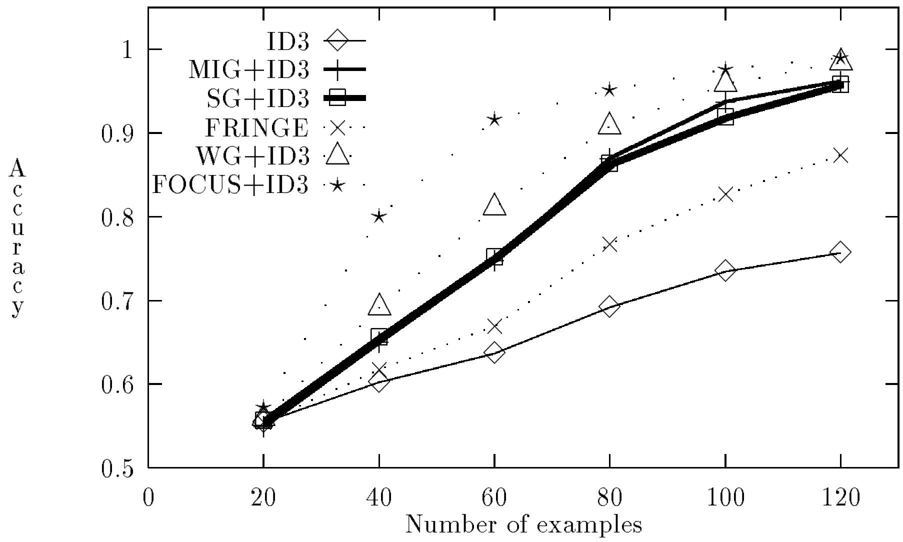

Figure 5: Learning curve for the concept $f(x_1, \ldots, x_{16}) = x_1 x_2 x_3 \bar{x}_4 \vee x_1 x_2 x_3 x_4 \bar{x}_5 \vee x_1 x_2 \bar{x}_3 x_4 x_5 \vee x_1 \bar{x}_2 x_3 \vee x_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 \vee \bar{x}_1 x_2 x_3 x_4 x_5 \vee \bar{x}_1 \bar{x}_2 x_3 x_4 x_5 \vee \bar{x}_1 \bar{x}_2 x_3 \bar{x}_4 \vee \bar{x}_1 \bar{x}_3 x_4 x_5 \vee \bar{x}_1 \bar{x}_3 \bar{x}_4 \bar{x}_5$ which has 5 relevant features out of 16.

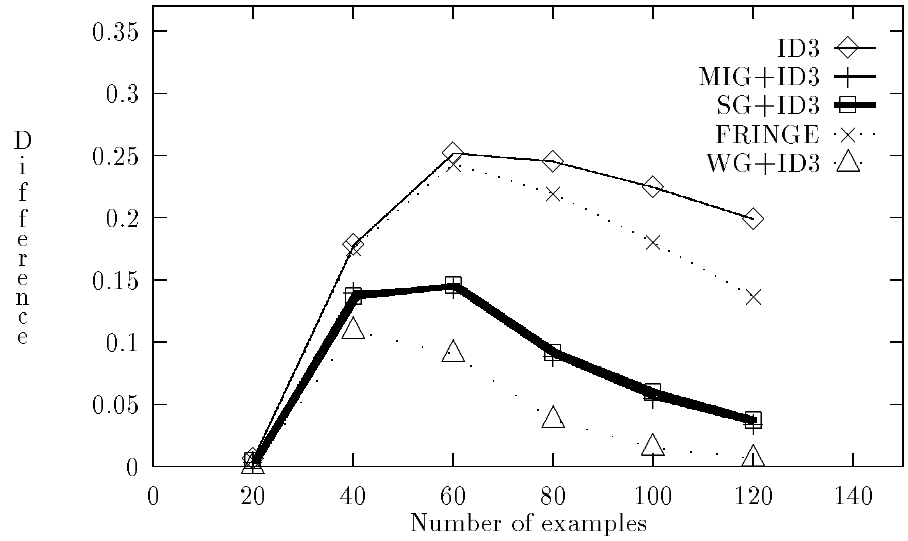

Figure 6: The difference between the accuracy of FOCUS+ID3 and the accuracy of the other algorithms averaged over all the randomly chosen 100 target concepts.

Table I: The number of sufficiency tests and CPU-time of FOCUS-1, FOCUS-2 and Weighted-Greedy for a target concept with 9 relevant features out of 25 available features.

Algorithm		Training Set Size				
		100	200	300	400	500
FOCUS-1	Sufficiency Tests	442189.4	1329330.5	2908068.9	2665664.0	2695393.0
	Time (seconds)	9.74	55.1	211.1	191.7	186.1
FOCUS-2	Sufficiency Tests	10593.1	9785.3	11339.5	8768.9	5582.3
	Time (seconds)	2.0	7.1	20.5	31.9	32.5
WG	Sufficiency Tests	8.5	10.1	10.7	10.5	11.0
	Time (seconds)	0.16	0.86	2.31	4.3	7.0