AN ABSTRACT OF THE THESIS OF

Hussein Saleh Almuallim for the degree of Doctor of Philosophy

in Computer Science presented on April 14, 1992.

Title: Concept Coverage and Its Application to Two Learning Tasks

Abstract approved.

Thomas G. Dietterich

The *coverage* of a learning algorithm is the number of concepts that can be learned by that algorithm from samples of a given size for given accuracy and confidence parameters. This thesis begins by asking whether good learning algorithms can be designed by maximizing their coverage. There are three questions raised by this approach: (i) For given sample size and other learning parameters, what is the largest possible coverage that any algorithm can achieve? (ii) Can we design a learning algorithm that attains this optimal coverage? (iii) What is the coverage of existing learning algorithms?

This thesis contributes to answering each of these questions. First, we generalize the upper bound on coverage given in [Dietterich 89]. Next, we present two learning algorithms and determine their coverage analytically. The coverage of the first algorithm, Multi-Balls, is shown to be quite close to the upper bound. The coverage of the second algorithm, Large-Ball, turns out to be even better than Multi-Balls in many situations. Third, we considerably improve upon Dietterich's limited experiments for estimating the coverage of existing learning algorithms. We find that the coverage of Large-Ball exceeds the coverage of ID3 [Quinlan 86] and FRINGE [Pagallo and Haussler 90] by more than an order of magnitude in most cases. Nevertheless, further analysis of Large-Ball shows that this algorithm is not likely to be of any practical help. Although this algorithm learns many

concepts, these do not seem to be very interesting concepts.

These results lead us to the conclusion that coverage maximization alone does not appear to yield practically-useful learning algorithms. The results motivate considering the *biased-coverage* under which different concepts are assigned different *weight* or *importance* based on given background assumptions.

As an example of the new setting, we consider learning situations where many of the features present in the domain are irrelevant to the concept being learned. These situations are often encountered in practice. For this problem, we define and study the *MIN-FEATURES bias* in which hypotheses definable using a smaller number of features involved are preferred. We prove a tight bound on the number of examples needed for learning. Our results show that, if the MIN-FEATURES bias is implemented, then the presence of many irrelevant features does not make the learning problem substantially harder in terms of the needed number of examples. The thesis also introduces and evaluates a number of algorithms that implement or approximate the MIN-FEATURES bias.

Concept Coverage and Its Application to Two Learning Tasks

by

Hussein Saleh Almuallim

A THESIS

submitted to

Oregon State University

in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

Completed April 14, 1992

Commencement June 1993

APPROVED:

Redacted for Privacy

Professor of Computer Science in charge of major

Redacted for Privacy

Head of Department of Computer Science

Redacted for Privacy

Dean of Graduate School

Date thesis presented April 14, 1992

Typed by Angel Smith for Hussein Saleh Almuallim

ACKNOWLEDGEMENTS

I would like to express my deep appreciation to my major advisor Prof. Tom Dietterich for all the help and support he provided me with during the course of this research.

I thank Prof. Prasad Tadepalli for many helpful discussions and Prof. Bella Bose for clarifying many issues in the literature of Error-Correction Codes, and for his continuous encouragement. I am also grateful to Prof. Paul Cull for introducing references on the isomorphism of Boolean functions, and for the careful comments on an earlier draft of this thesis.

I am indebted to my dear friend, Dr. Sulaiman Al-Bassam, for many valuable technical discussions. I thank Robert Rowley for a useful discussion on symmetric groups and Prof. Mina Ossiander of the Math department for verifying some of my mathematical proofs.

Special thanks are due to Dr. Ghulum Bakiri for many insightful discussions, to Dr. Nick Flann for comments on an earlier draft of this thesis, and to other students of Prof. Dietterich: Giuseppe Cerbone, Dietrich Wettschereck, Vijay Srinivasan, for all kinds of help and enjoyable company.

The University of Petroleum and Minerals-Saudi Arabia provided me with a generous scholarship to do my Ph.D. in the US. I should not forget to thank Dean Mohammed Al-Suwayyel for his efforts which made this scholarship possible despite the lack of budget. This work was also supported partially by the NSF under grant number IRI-86-57316 (Presidential Young Investigator Award) through my major advisor.

My parents, brothers, and sister were an endless source of love and support to me. The last, and surely the most important, note of thanks belongs to my dearest wife—Setsuko, for all her warm support and encouragement throughout my Ph.D. student life.

The remaining errors and ambiguities of this thesis are my sole responsibility.

Table of Contents

List of Figures

List of Tables

Concept Coverage and Its Application to Two Learning Tasks

Chapter 1

Introduction

Training (as opposed to programming) machines to do a certain job has long been an attractive goal for a large number of researchers. The field of inductive machine learning is concerned with the study of systems that attempt to infer a general rule for performing some task after being exposed to a sequence of "situation-response" pairs. Each such "situation-response" pair contains a description of a specific situation, along with the desired response by the system in this situation. These pairs are usually called *examples*, and the process of inferring a general rule out of these examples is called *generalization* or *inductive learning* form examples.

Imagine that you want to teach somebody who does not know what "flower" means how to discriminate between objects that are flowers and those that are not. Rather than providing the person with a detailed description of what flowers look like (using whatever communication language is available), one would prefer, instead, to show her/him a collection of examples of flowers and a collection of "non-flowers" and (reasonably) hope that it will not take long before she/he figures out her/his own mechanism to recognize flowers.

In this example, the first choice is analogous to the conventional "programming" approach, while the latter represents the "training" approach.

Although programming computers has almost always been the way to have them handle various tasks, in many cases this approach has proven to be difficult, ineffective and sometimes not even feasible. This is especially true for classification tasks performed by human experts. Just as you find it hard to come up with a concise definition for "flowers", in many practical domains it is hard for skilled experts to provide their knowledge about how they make their judgments, in a concrete and feasibly programmable form. On the other hand, it is often relatively easy to generate examples (or "situation-response" pairs) by watching how these experts respond to various instances of the problem. In these kinds of situations, the training, or the inductive machine learning approach has definite advantages.

Consider, for example, the medical diagnosis problem. Suppose that our goal is to build a system that looks at a list of symptoms and laboratory tests for some patient, and then makes a prediction of whether or not this patient has typhoid fever. While it is hard to *compile* the knowledge doctors use in diagnosing this disease into a readily programmable form to build the desired system, it is quite manageable to collect a list of patients' data, where each patient is labeled by doctors as to whether she/he has the disease. As we did in our "flowers" example, we may hope that proper training using such data will lead to some satisfactory classification rule for typhoid fever diagnosis.

As another example, consider the processing of credit card applications [Carter and Catlett 87]. Experts in a credit card company examine the information available about an applicant to decide whether or not to offer her/him a credit card. A training example in this case is composed of a list of information about a previous applicant, and a label telling whether or not approving the applicant's request was the right decision based on the recent behavior of the applicant.

Naturally, the goal of the inductive machine learning research is not to attack each of these problems individually. That is, finding an ad-hoc procedure tailored

for each particular application is not a desired approach. As it is the case in other branches of science, only results with a reasonable level of generality are to be sought. Though the above tasks (recognizing flowers, diagnosis of typhoid fever and credit card approval) may seem to be totally different from each other, they do in fact share a basic underlying problem of a special structure. The abstract form of these problems, as formulated in the next section, is what is usually considered in the research of inductive machine learning and what will be studied in this thesis.

1.1 Stating the Learning Problem

Suppose f is a $\{0,1\}$-valued function defined over a set of n variables $\{x_1, x_2, \cdots x_n\}$, where each of these variables represents some property or *feature* of the objects in the domain and the value of the function for an object gives the proper *classification* for that object. For instance, the variables in our "diagnosis of typhoid fever" example may represent features like the patient's blood pressure, temperature, sex and so on. In this case, for some assignment X_1 to the variables $\{x_1, x_2, \cdots x_n\}$ representing some patient's case, $f(X_1)$ takes the value 1 if the patient has the disease, or 0 otherwise. A pair $\langle X_1, f(X_1)\rangle$ then represents a "situation-response" pair, or an *example* of f. A collection $S = \{\langle X_1, f(X_1)\rangle, \langle X_2, f(X_2)\rangle, \langle X_3, f(X_3)\rangle, \cdots, \langle X_m, f(X_m)\rangle\}$ is usually called a *sample* of the function f, and m, the number of examples in the sample, is called the *size* of the sample.

Using these definitions, the inductive learning problem may (loosely) be stated as follows:

Given : A sample of f of a reasonable size,

Find : A procedure that simulates f.

f is usually called the *target concept*, and the output of the learning system is called the *hypothesis* of the system.

Of course, the learning system is not expected to see all of the possible examples of the domain. In practice, only a sample of a small size (compared to the whole space of examples) can be given to the learning system. This makes capturing the *exact* target concept f too challenging. Therefore, exactness is usually given up and the goal becomes only to arrive at some hypothesis that is "good-enough"; i.e. some $\hat{f}$ that satisfactorily approximates f, where "good-enough" is based on some threshold usually called the *accuracy level* or the *accuracy parameter.*

Although such relaxation of the success criterion eases the problem significantly, it turns out that, as stated, the learning task is still too hard to tackle. The difficulty is due to the lack of guarantees concerning the *quality* of the sample; that is, the fact that some samples may be more informative than others. The system may be getting a sample which contains mostly unimportant examples of the target concept; i.e. examples that do not capture the essence of that concept. Finding a good hypothesis starting from such *non-representative* samples is certainly hard. To account for such a possibility, we make one more relaxation: For the learning system to be successful in learning f, we only require that the hypothesis $\hat{f}$ of the system be close-enough to f for *nearly all* the samples of f. That is, we are allowing the system to be arbitrarily wrong occasionally, as long as this happens only rarely. This is governed by what is usually called the *confidence level* or the *confidence parameter.*

To illustrate these two kinds of relaxation in the success criterion of learning, assume, for example, an accuracy level of 95% and a confidence level of 99% are desired in learning some target concept f. Suppose we randomly draw a very large number of samples of f each of size m, and formed a hypothesis from each sample by feeding the sample to some learning system L. If 99% or more of these hypotheses are 95% correct (or better), then we say that the learning system L is successful in learning the concept f using m examples for the above-mentioned accuracy and confidence levels. Conversely, if these requirements are not satisfied, then this constitutes a failure of the system in learning f, for the given parameters.

Typically, the confidence and accuracy levels are desired to be as close to 100% as possible. Nevertheless, the higher these levels, the more examples, and the more time needed to process these examples will be required for learning.

This approach to inductive learning of concepts was first proposed and studied by Valiant in his influential paper [Valiant 84]. In his work, it is assumed that objects in the universe are encountered according to some fixed but unknown probability distribution. This underlying probability distribution is used in two ways:

- First, examples of the target concept are gathered according to this distribution.
- Second, the quality of an hypothesis is based on the same distribution. Specifically, the the error of an hypothesis is defined as the probability of encountering an object that is incorrectly classified by the hypothesis.

The idea behind assuming the same distribution during both the *training* and *testing* phases is to make it a "fair game". The probability distribution measures the significance of each object. Important objects are more likely to be encountered, and thus, have greater influence on the quality of the hypothesis. This is compensated for by the fact that these are more likely to appear in the sample as well. The converse is again true for those unimportant objects.

To illustrate this point, suppose that the target concept is to recognize "Keys that are used to open doors". Normally, it is safe to assume that keys are made of metal. However, although rather unusual, doors with electronic locks that are opened using magnetic plastic cards can sometimes be seen in sophisticated buildings or high-class hotels. Suppose now that a learning system was trained on examples of this target concept that are picked under the "usual distribution" of keys. Since it is very unlikely that the system will ever encounter any plastic key, it may well be expected that the system will end up with something like "made-of-metal" as a condition for an object to be a key. This should be reasonable if the

system is to be used in the same environment since plastic keys are again rarely encountered. On the other hand, after being trained in a normal environment, the system will fail badly if it were tested in a building where many of the keys are made of plastic. Such a test, however, is considered unfair because the system is being tested in an environment different from the one it was trained in.

Learning in Valiant's sense as described above is usually called *Probably Approximately Correct (PAC) Learning.* In this term (which is due to Angluin [Angluin 88]), the word "Approximately" corresponds to allowing a limited amount of error in the hypothesis as given by the accuracy parameter, while the word "Probably" indicates that good approximation is required only with high probability, and not definitely, as specified by the confidence parameter.

In this thesis, we adopt PAC learning as the criterion for successful learning of a target concept. We restrict our attention, however, to the case of Boolean concepts only. That is, we assume that the target concept and the features of the domain can take only the values "true" and "false". The formal definitions and terminology concerning PAC learning will be given in Chapter 2.

1.2 Inductive Bias of Learning Algorithms

Suppose a learning system is to learn an unknown concept f, defined on n Boolean variables. There is a total of 2^n examples of which the system will be given, say m examples, where m is considerably smaller than 2^n. The learning problem can then be viewed as the task of completing a partial truth table of the function being learned. The system is given an incomplete truth table of f in which the value of f is listed only for m assignments out of the 2^n possible assignments of the n variables. The task is to complete the table by "guessing" the value of f for the remaining $2^n - m$ assignments.

Now, if f can be any target concept, then for any assignment X for which the value of f is not given, $f(X)$ has an equal chance of being 0 or 1. This means that

there exist as many as 2^{2^n-m} candidate hypotheses that all agree with the given m examples.

For a learning algorithm to eventually select a single hypothesis as its output, the designer of the algorithm must, in one way or another, set some basis for the algorithm to choose among the many available alternatives. Such a basis is known in the machine learning community as the *bias* of the learning algorithm [Mitchell 80, Utgoff 86]. Mitchell defines bias as follows:

> "We use the term bias to refer to any basis for choosing one generalization (hypothesis) over another, other than strict consistency with the observed training instances."

In constructing a learning algorithm, bias is usually implemented in one of two ways:

- Restricting the space of hypotheses.
- Imposing a preference ordering on that space.

In the **Restricted Hypothesis Space Bias**, a language that is syntactically incapable of representing all of the possible hypotheses is used to represent the output of the learning algorithm. In this way, the algorithm is forced to focus only on those hypotheses representable in this language, and to discard all other hypotheses.

For example, hypotheses may be represented as pure conjuncts of literals (i.e., pure monomials), or as k-dnf formulae, where a k-dnf formula is a disjunction of conjuncts such that each conjunct has at most k literals, for some constant k. Another example, is to use some fixed artificial neural network architecture with adjustable weights, such as a perceptron or a feedforward net with a fixed number of hidden units. In all of these cases, any hypothesis that can not be expressed using these representations is ruled out from the learning algorithm's consideration.

In the **Preference Bias**, no restrictions on the expressiveness of the language are forced. Instead, some preference ordering is defined on the space of all hypotheses, and then, among all the candidate hypotheses, the learning algorithm is to choose the most preferred one according to this ordering.

An example of this bias is to use decision trees as the representation language. It is known that any Boolean function can be represented as a decision tree. The preference ordering may be based on the *simplicity* of a decision tree measured, for example, by the number of nodes in the tree.

The designer of a learning algorithm must also specify how the algorithm is to handle possible conflicts among candidate hypotheses. These are situations in which there are two or more candidate hypotheses that are

- representable in the output language in the restricted hypothesis space bias case, or
- equally preferable in the preference bias case.

All such choices in the selection of the hypothesis other than what is suggested by the sample are, of course, included in the bias of the learning algorithm.

With the above definition of bias, we can see that a complete description of some bias determines a mapping from the space of samples to the space of hypotheses. A learning algorithm which follows a given bias is just implementing the mapping defined by that bias. Thus, ignoring computational aspects, specifying a complete bias is equivalent to constructing a learning algorithm.

In many cases, however, only the higher level of the hypothesis selection process is considered important. For example, we may desire that a learning algorithm chooses a hypothesis representable as a decision tree with a minimal number of test nodes, without exactly specifying how to resolve ties between different decision trees with the same number of test nodes. An *incomplete* bias of this kind does not specify a unique learning algorithm but rather corresponds to a collection or a class of algorithms. In selecting the hypothesis, these algorithms have the

common property of satisfying the requirements outlined by the bias. However, the choice among equally preferable hypotheses is handled differently by each algorithm. These algorithms are said to be *implementing* the bias, in the sense that they never violate the hypothesis selection rule prescribed by the bias.

1.3 Learning Behavior of an Algorithm

Unfortunately, even with the relaxations in the criterion for successful learning given in the PAC learning framework, studies in this field show that we cannot hope for a "universal" learning algorithm that can PAC learn (from a reasonably sized sample) all the possible concepts there are, for reasonable accuracy and confidence levels.

In fact, Dietterich [Dietterich 89] has shown that the number of concepts that can be learned from a sample of a reasonable size forms only a small fraction of the concept space. This topic will be discussed in detail in Chapter 3. For now, we will just provide some insight of where these limitations stem from.

When a learning algorithm is given a sample of m examples of some target concept f, the job of the algorithm is to construct an hypothesis that *guesses* the class of each of the remaining $2^n - m$ objects that are not included in the sample. Assuming that f is an arbitrary function, each of these objects has an equal chance of being classified as 0 or 1. Since in the PAC learning framework, only a few mistakes are allowed, the learning algorithm has to be correct for almost all of the unknown $2^n - m$ values of f. Consequently, unless we are extremely *lucky*, the algorithm is highly unlikely to end up with an hypothesis that is an acceptable approximation of the target concept f.

In other words, given a small sample, there will be a large number of candidate hypotheses agreeing with the available examples. As far as the sample is concerned, all these candidates look equally good. However, one can show that the vast majority of these hypotheses are very bad approximations of the target concept f,

and consequently, this means there is a very small chance of being close enough to f.

One way or another, the algorithm has to *bet* on one of the possible hypotheses, say $\hat{f}$. The algorithm's choice of $\hat{f}$ will then divide the space of concepts into two groups. Those that are close enough to $\hat{f}$ (according to the designated accuracy parameter) and those that are not. If the target concept happened to be in the first group, then the algorithm is successful. Otherwise, the algorithm will fail in approximating the target concept. The problem here is that the first group is substantially smaller than the latter one, and thus, the chance for the learning algorithm to succeed is severely limited.

The fact that in PAC learning it is acceptable to have a totally wrong hypothesis within some small probability does not greatly change the final picture: Given only a sample of reasonable size, any algorithm can learn only a limited fraction of the whole concept space, regardless of the bias implemented by the algorithm.

Normally, the set of concepts learned by an algorithm grows as the sample size increases and/or as the desired accuracy and confidence levels decrease. This set of learned concepts characterizes what we call the *Learning Behavior* of the algorithm. That is, the learning behavior is a description of what target concepts an algorithm does/does not successfully learn for given sample size, accuracy level and confidence level.

Naturally, it is the way a learning algorithm chooses its hypothesis that determines the set of concepts learned by the algorithm. The learning behavior of an algorithm is, therefore, a consequence of the bias implemented by the algorithm. The converse, however, is not necessarily true. Specifying a particular learning behavior that we desire a learning algorithm to achieve does not immediately suggest what bias should be followed by the algorithm.

This leads us to the important distinction between the *desired* learning behavior as the *specifications* to be met in designing a new learning algorithm, and the bias *chosen* by the learning algorithm designer to fulfill this learning behavior

as required. It is unfortunate that separating the desired learning behavior from the bias has often been ignored in the inductive machine learning literature. The term "bias" has largely been used in a confusing way to denote both notions. The effect of this intermixing between the two notions is evident in the design process of many well-known learning algorithms whose learning behavior has never been characterized. More details on this subject is given in the next section.

1.4 Designing Learning Algorithms

Historically, the development of inductive learning algorithms has been a two-step process:

1. Select a hypothesis space along with some representation scheme that *appears* to be appropriate to practical learning problems as seen by the algorithm designer's intuition. This might be a restricted language, such as using perceptrons or pure conjunctive Boolean formulas. Or it might be the use of a more powerful language, which can represent any hypothesis, combined with some preference ordering.

2. Develop an algorithm to find some hypothesis within the chosen representation scheme that is consistent with the given collection of examples.

Two well-known algorithms developed following this approach are ID3 [Quinlan 86] and Error-Backpropagation [Rumelhart *et al.* 86].

ID3 builds a decision tree consistent with the sample. It attempts to come up with a "small" tree by applying a greedy heuristic to decide what to test in each node of the tree. (See Section 2.5 for more details about this algorithm.) The tendency toward smaller decision trees is based on what is known as Occam's Razor which asserts that simpler hypotheses are more credible, and on the intuition that using decision trees seems suitable as the representation language for various practical applications.

Error-Backpropagation is an algorithm that adjusts the weights in a feed-forward artificial neural network in order to arrive at some configuration that is consistent with the given sample. This approach is motivated by the hypothesized similarity between this architecture and the human brain.

A shortcoming of the above approach in designing learning algorithms is that there is no separation between a *specification* of the desired learning behavior of the algorithm and its *implementation*. Specifically, the learning behavior of these algorithms is shaped implicitly and only as a side-effect of the bias originating from the particular choices made by the designer in the two steps above. Consequently, the learning behavior of these algorithms is often unclear, and, hence, it is difficult to tell in advance whether an algorithm is appropriate for a given learning problem.

Simply put, the above approach is to construct learning algorithms based on some *sensible* ideas and *then* see what these algorithms actually achieve. Clearly, this approach is ignoring a very basic principle of design. Namely, one should first specify the objectives of the "product" to be designed. Accomplishing all (or sometimes most) of these objectives as economically as possible then becomes the designer's challenge.

Recently, this issue has been raised by some authors (e.g., [Buntine 90], [Wolpert 90]) who have advocated the following new procedure:

1. Specify the desired learning behavior, i.e. what the algorithm should be able to learn.

2. Based on Step 1, adopt some bias over the space of hypotheses and select some scheme for representing the hypotheses.

3. Design an *efficient* algorithm that implements the bias, or at least approximates it.

The last two steps simply mean to select some bias and turn it into a learning algorithm (as implicitly done in the previous approach). What is different now

is that the chosen bias and its implementation must achieve the desired learning behavior (or at least approximate it) using as little resources as possible, where resources in this case are the number of examples and the amount of computation consumed by the resulting algorithm.

By having the desired learning behavior precisely stated in the first step, we have a clear idea of what the objectives are. We believe that this procedure in designing learning algorithms is useful in many aspects:

- It allows formal analysis of the learning problem. This can lead to algorithms with provable guarantees on achieving the desired learning behavior, or it can produce negative results that save us the effort of pursuing unachieveable goals.
- The goal as stated in the desired learning behavior, provides us with a criterion with which to measure performance. This criterion enables us to contrast various competing algorithms, and provides a measure for optimality.
- Studying practically motivated learning behaviors is likely to lead to useful algorithms in application domains where these learning behaviors are justified.

1.5 Goal and Topic of the Thesis

The goal of this thesis is to pursue the procedure outlined in the previous section for designing learning algorithms. In particular, we investigate two important learning problems:

1. Learning as many target concepts as possible from a fixed amount of data.
2. Learning from data containing many irrelevant information.

The objective is to design learning algorithms that perform well with respect to the learning behaviors suggested by the above learning problems. Let us look

more deeply at each of these problems.

(1) Maximum Coverage Learning: When we say that a given learning algorithm learns some target concept f, then this, of course, means that if f is the target concept in some application domain, then this learning algorithm is going to do a successful job in that domain. Therefore, intuitively, one may have a justification to believe that the more target concepts learned by an algorithm, the wider the range of applications this algorithm is expected to be successful in.

Motivated by the above understanding, Dietterich [Dietterich 89] raised the topic of designing learning algorithms that learn the maximum possible number of target concepts for given sample size, accuracy and confidence levels. In other words, the learning behavior of an algorithm is quantified into a single amount by counting how many concepts are learned by the algorithm.

In this thesis, we call this quantity the *coverage* of the algorithm. Designing learning algorithms with the maximum possible coverage is the objective of this part of the thesis.

(2) The MIN-FEATURES Learning Behavior: In many situations in machine learning of concepts, one has to deal with training data containing many features that are irrelevant to the concept being learned. In this kind of situation, the concepts defined over many features are less likely to be the correct target concept. We should, thus, seek learning algorithms whose learning behavior is shaped accordingly. That is, a good algorithm in these cases is the one that pays more attention to learning concepts that have a smaller number of features involved.

The objective of the second part of the thesis is to design learning algorithms that are tuned to the above learning behavior which we call the MIN-FEATURES learning behavior.

1.6 Contribution of the Thesis

The contribution of this thesis can be summarized as follows.

(1) Maximum Coverage Learning: This thesis contributes to the study of maximum coverage learning in the following aspects:

- First, we generalize the proof Dietterich [Dietterich 89] gives for an upper bound on the coverage of any learning algorithm.
- Second, we present a family of algorithms which we call the "Balls" algorithms, and give formal analysis of the coverage of these algorithms. Particularly, we prove that the number of examples required by one of our algorithms (the Multi-Balls algorithm) is within a constant factor of the number of examples required by an optimal algorithm to achieve the same coverage.
- And third, we introduce techniques for reducing the computational costs involved in the experimental measurement of the coverage of existing learning algorithms. These techniques make it possible to extend the limited experiments reported in [Dietterich 89] for measuring the coverage of some of the well-known learning algorithms.

Our results show that the coverage of the existing learning algorithms we tested is far below what can actually be achieved by our Balls algorithms. These results could mean that existing learning algorithms can be subject to significant improvement. Nevertheless, we also found that there exist trivial algorithms (e.g. the Large-Ball algorithm) that can attain high coverage while learning only very uninteresting collections of concepts. The fact that algorithms with practically good reputation are outperformed (in terms of coverage) by such trivial algorithms is quite surprising since it implies that higher coverage does not necessarily mean a practically better algorithm. These results indicate the necessity of revising the definition of coverage as a measure of performance for learning algorithms. Particularly, the results suggest that ignoring *which* concepts are actually learned and

merely counting the *number* of concepts learned is inappropriate. Instead, it seems essential that we specify, as a component of the desired learning behavior, the *relative significance* or *importance* of the target concepts and then seek an optimal algorithm within that setting.

(2) The MIN-FEATURES Learning Behavior: The contribution of this thesis to the study of learning in the presence of many irrelevant features includes the following:

- First, we define the *MIN-FEATURES bias* in which hypotheses with smaller number of features involved are preferred. We prove an upper bound on the number of examples sufficient for learning when the *MIN-FEATURES bias* is followed and show that this bound matches (within a constant factor) the number of examples needed by *any* learning algorithm, regardless of whether or not it follows this bias. Our results show that the presence of many irrelevant features does not make the learning problem substantially harder in terms of the number of examples needed for learning.

- Second, we assess the computational complexity of implementing the MIN-FEATURES bias by linking this problem to the well studied problem known as the MINIMUM-SET-COVER problem [Garey and Johnson 79].

- Third, we present FOCUS-1, a simple algorithm that implements the MIN-FEATURES bias. We show that this algorithm runs in time quasi-polynomial in the appropriate learning parameters.

- Fourth, we introduce FOCUS-2 which is a more efficient implementation of the MIN-FEATURES bias, discuss its time complexity and empirically compare its running time to that of FOCUS-1.

- Fifth, we experimentally evaluate the performance of ID3 [Quinlan 86] and FRINGE [Pagallo and Haussler 90] in learning from training samples with

many irrelevant features. Comparisons with the FOCUS algorithms show that—contrary to common belief—these algorithms are badly affected by the presence of irrelevant features.

- Sixth, we introduce the Mutual-Information-Greedy, Simple-Greedy and Weighted-Greedy algorithms. These are three efficient algorithms that approximate the MIN-FEATURES bias and that may be used in practical situations where FOCUS-1 and FOCUS-2 are computationally unaffordable.

- Finally, we present empirical results comparing the performance of the above algorithms in learning in the presence of many irrelevant features. Our experiments show that that the Weighted-Greedy algorithm provides an excellent approximation of the FOCUS algorithms but with substantially less computational costs.

1.7 A Guide to the Thesis

In Chapter 2, we give a collection of definitions and notation to be used in the rest of the thesis. We also describe the learning algorithms ID3 [Quinlan 86] and FRINGE [Pagallo and Haussler 90] which will be used in most of the experiments reported later in the thesis.

Chapter 3 summarizes our results on maximum coverage learning.

Our work on the MIN-FEATURES learning behavior is presented in Chapters 4 and 5. In Chapter 4, we discuss the problem of implementing the exact form of the MIN-FEATURES bias, whereas in Chapter 5, we look at efficient heuristics for approximating this bias.

Finally, in Chapter 6, we conclude with a list of related open problems suggested by the course of research followed in this thesis.

Chapter 2

Background

This chapter introduces the terminology and background knowledge that will be needed throughout the thesis. It is recommended that the reader start with this chapter before proceeding to Chapter 3. Sections 2.1 through 2.3 are particularly necessary for understanding the material of the subsequent chapters.

2.1 Basic Definitions

In this thesis we assume that *objects* are described using a finite set of Boolean variables, i.e. variables that take either *true* or *false* as a value. Each such Boolean variable will be referred to as a *feature*. The parameter $n \geq 1$ will be exclusively used to denote the number of features describing the objects, and the features will usually be named $x_1, x_2, \cdots, x_n$. For convenience, we will let 1 and 0 denote the feature values *true* and *false*, respectively. An object can, therefore, be viewed abstractly as a $\{0,1\}$ assignment to the n features, or equivalently, as an n-bit string. The set of all possible objects, or the *universe*, is denoted U_n and is equivalent to $\{0,1\}^n$.

A *concept* is a classification rule that partitions the universe into objects that belong to the concept and those that do not. In other words, a concept is just an arbitrary set $c \subseteq U_n$. Equivalently, a concept can be viewed as a function

$c : \{0,1\}^n \rightarrow \{0,1\}$, where for any object $X \in U_n$, $c(X) = 1$ if X is in c, and 0 otherwise.

A Boolean formula f is said to be *representing* a concept c if for every object X, $c(X) = 1$ if and only if f is satisfied by the truth assignment implied by X.

A concept c such that $c(X) = 0$ for every object $X \in U_n$, will be called the *nil* or the *empty* concept. Conversely, a concept c such that $c(X) = 1$ for every object $X \in U_n$, will be called the *true* concept. It should be clear that the *nil* and the *true* concepts can be represented by (the Boolean formulas) 0 and 1, respectively.

By C_n we denote a set of concepts that are all defined over n features. An infinite union of such sets for $n \geq 1$ is called a *class of concepts.* That is, a class of concepts $\mathcal{C}$ is defined as $\bigcup_{n \geq 1} C_n$, where $C_n \subseteq 2^{U_n}$ for each $n \geq 1$. Note that a class of concepts is defined over all values of n whereas a set of concepts is defined for a specific number of features (i.e., with respect to a fixed value of n).

A class of concepts is usually associated with a complexity measure which defines the *complexity* of each concept in the class. The complexity of a concept, denoted s, is usually defined as the minimum number of bits needed to encode the concept under some fixed *representation* or *encoding scheme.* Such representation scheme is given as a pair $\mathcal{L}, \sigma$, where $\mathcal{L}$ is a language (a set of strings) and σ is a mapping

$$\sigma : \mathcal{L} \rightarrow \mathcal{C}$$

such that for every $l \in \mathcal{L}$, $\sigma(l)$ determines a unique concept in $\mathcal{C}$ and for every $c \in \mathcal{C}$, there exists some string $l \in \mathcal{L}$ for which $\sigma(l) = c$. [1]

For a given class of concepts, if $\mathcal{C}$ is the set of all possible concepts (that is, if $C_n = 2^{U_n}$ for all $n \geq 1$) then the class is said to be a *complete* class of concepts. Otherwise, the class is to be called a *restricted* class of concepts.[2]

---

[1] Formally, σ is called an *onto* function from $\mathcal{L}$ to $\mathcal{C}$.

[2] The usual practice in the computational learning theory research is to define a class of concepts by specifying a language $\mathcal{L}$ for representing the class. In this case, any concept that can be represented by some string in $\mathcal{L}$ is in the class, and vice versa. For example, if $\mathcal{L}$ is the set

An *example* of a concept is an object along with a label that indicates whether or not the object is in the concept. For an object X, this will be denoted as a pair $\langle X, c(X)\rangle$. An example is said to be *positive* when $c(X) = 1$, i.e. when X belongs to c, or *negative* otherwise.

We will always assume some probability distribution D over the space of objects, and denote the probability of an object X under this distribution by $Pr[X]$. A *sample* of a concept c is just a collection

$$\langle X_1, c(X_1)\rangle, \langle X_2, c(X_2)\rangle, \langle X_3, c(X_3)\rangle, \cdots, \langle X_m, c(X_m)\rangle$$

of examples of c. The parameter m will exclusively be used to denote the *sample size*, i.e. the number of the examples in a sample.

Sampling means to randomly draw a sample of a concept. This is usually done *with replacement.* That is, examples are drawn independently from each other, and thus, the same example may appear in a sample more than once. In some special cases, we will also consider sampling *without replacement* where examples in the sample are all distinct.

For a specific sample S of size m of some concept c, $Pr[S]$ denotes the probability of getting S as the result of sampling, where this probability is taken over the space of all possible samples of c of size m. Note that such a space depends on whether sampling is done with or without replacement, which will always be clear from the context.

A *learning algorithm* is an algorithm that takes as input a sample of some unknown concept, and returns as output some concept $h \subseteq U_n$. A learning algorithm L can, therefore, be viewed as a mapping $L : \mathcal{S} \rightarrow 2^{U_n}$, where $\mathcal{S}$ is the space of all samples of all the concepts in 2^{U_n}, for $n \geq 1$.

---

of all monomials (pure conjunctions of literals, where a literal is a feature or its negation), then $\mathcal{L}$ defines the class of concepts in which each concept can be represented as a monomial. Note that this is a restricted class of concepts since not every concept can be represented by a monomial. In contrast, if $\mathcal{L}$ is the set of all DNF formulae, then $\mathcal{L}$ defines a complete class of concepts since every Boolean concept can be represented as a DNF formula.

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 on a sample S, a learning algorithm L returns an hypothesis h, then we say that L *maps* S *to* h. The set of hypotheses that may ever be returned by a learning algorithm is to be called the *hypothesis space* of the algorithm.

If L is a learning algorithm that on any training sample S returns an hypothesis that classifies all the training examples of S as originally given in S, then L is said to be a *consistent* learning algorithm. In other words, a consistent learning algorithm is an algorithm whose hypothesis never disagrees with the training sample.

The *error* between two concepts h and c under the probability distribution D over the space of objects is the probability that a randomly drawn object is classified positive by either c or h and negative by the other. This is equivalent to the sum of the probability of all objects on which c and h disagree; or formally

$$error(c,h) = \sum_{X:c(X)\neq h(X)} Pr[X]$$

where $Pr[X]$ denotes the probability of the object X under the distribution D.

If $error(c,h) \leq \epsilon$, then we say that c is ϵ-*close* to h, or *within* ϵ *from* h. A concept c that is not ϵ-close to h is said to be ϵ-*far* of h.

2.2 PAC Learning and Polynomial Learnability

Suppose c is some unknown concept and let D be the probability distribution over the space of objects. For sample size m and fixed constant ϵ such that $0 < \epsilon < 1$, the success of a learning algorithm L in learning the concept c is based on the following random experiment:

1. Draw a random sample S of size m of the concept c, under the probability distribution D.

2. Let h be the hypothesis of the learning algorithm L when given S as the training sample.

3. If $error(c, h) \leq \epsilon$, report *Success*, otherwise report *Failure*. The error here is defined under the same probability distribution D under which the sample S was drawn in Step 1.

This procedure defines a random variable with two outcomes: *Success* and *Failure*. The probability of getting *Success* measures how successful the algorithm L is in learning the concept c using a sample of size m drawn under the probability distribution D.

Now, let W_c^L denote the event that the outcome of the above random experiment is *Success*. The probability of this event is, therefore, given by

$$Pr[W_c^L] = \sum_{S \in \mathcal{S}_{L,c}} Pr[S] \tag{2.1}$$

where $\mathcal{S}_{L,c}$ is the set of all samples of c of size m on which L returns an hypothesis that is ϵ-close to c, and $Pr[S]$ is the probability of drawing a sample S under the probability distribution D. In other words, each sample S that is mapped by L to an hypothesis that is ϵ-close to c contributes an amount of $Pr[S]$ to the summation of $Pr[W_c^L]$ given by equation (2.1).

Now, the decision of whether or not L is successful in learning c is determined by comparing $Pr[W_c^L]$ to a threshold $(1 - \delta)$, where $0 < \delta < 1$. Specifically, if

$$Pr[W_c^L] \geq 1 - \delta \tag{2.2}$$

then we say that L *learns* c with respect to the given values of ϵ and δ, from a sample of size m under the probability distribution D. ϵ and δ are called the *accuracy* and *confidence parameters*, respectively.

A learning algorithm is evaluated based on two types of resources: the number of examples and the amount of computation required to learn. Naturally, the amount of these resources sufficient to make an algorithm learn a particular concept

c depends on the values of ϵ and δ. The higher the desired accuracy and confidence (i.e., the lower the values of ϵ and δ), the more costly the algorithm is expected to be. Moreover, when c can be any concept in some set C_n of concepts, then this may also depend on n and the complexity of the concept, $s = \mathit{complexity}(c)$.

More formally, we consider two types of complexity of a learning algorithm L with respect to a class of concepts $\mathcal{C}$:

1. The *Sample Complexity* : This is the smallest $m(n, s, \epsilon, \delta)$ such that for any $n \geq 1$, any c in $C_n \subset \mathcal{C}$ of complexity s, any ϵ and δ such that $0 < \epsilon, \delta < 1$, and any probability distribution D over U_n, L learns c with respect to ϵ, δ and D from a sample of size $m(n, s, \epsilon, \delta)$.

2. The *Computational Complexity* : This is the number of computational steps (as a function of n, s, m, ϵ and δ) required by the algorithm to achieve the same condition in (1) above.

Note that the sample and computational complexities are defined above as the worst-case over all probability distributions. These definitions, however, can also be specialized to some specific probability distribution by removing the "any probability distribution" requirement.

What we have described so far is usually called the *Probably Approximately Correct (PAC)* learning model. This model was introduced by Valiant [Valiant 84] and has been widely adopted in the theoretical machine learning research [COLT88, COLT89, COLT90, COLT91]. In this model, the overall objective is to identify whether it is feasible to learn a given class of concepts. The *polynomial learnability* of a class of concepts is based on the existence of a learning algorithm whose sample and computational complexities in learning the concepts of the class are each bounded by some polynomial in $n, s, \frac{1}{\epsilon}$ and $\frac{1}{\delta}$.

Many restricted classes of concepts have been shown to be polynomially learnable. Examples of these classes are the classes of monomials, k-cnf, k-cnf [Valiant 84, Haussler 88] and k-decision-lists [Rivest 87]. Negative results have also been proven

for classes such as the class of μ-formuale and Boolean threshold functions [Kearns *et al.* 87]. On the other hand, the polynomial learnability of many complete classes (e.g. DNF, CNF) are still open problems [Valiant 84, Haussler 88, Verbeurgt 90].

2.3 Probability Distribution Over the Space of Objects

In this thesis, we will consider two cases:

1. The distribution-free case where no assumptions are made about the probability distribution over the space of objects.

2. The uniform distribution case where the probability of drawing each object is equally likely.

For the above two cases, sampling is usually assumed to be with replacement. That is, examples are drawn independently from each other, and thus, the same example may appear more than once in a sample. However, when the uniform distribution over the space of objects is assumed, we will sometimes consider sampling without replacement. This means that each example can be drawn only once, and that a sample of size m contains exactly m distinct examples. It should be noted that in practical settings, the training sample contains only a small proportion of the space of all objects (i.e., $m \ll 2^n$). Assuming the uniform distribution over the space of objects, the probability that the same object appears more than once in a randomly drawn sample is ignorably small. Therefore, the results obtained for the uniform distribution case should not be greatly affected by whether sampling is done with or without replacement.

2.4 Isomorphism in the Space of Boolean Functions

Let PN^n be the set of all operators that permute and/or negate the Boolean variables $\{x_1, x_2, \cdots, x_n\}$. For example, $\gamma \in PN^3$ may be the operator that maps

$\{x_1, x_2, x_3\}$ to $\{x_3, \neg x_1, \neg x_2\}$. Applying this operator to a Boolean function g of these three variables gives

$$\gamma(g(x_1, x_2, x_3)) = g(x_3, \neg x_1, \neg x_2)$$

which is another Boolean function of x_1, x_2 and x_3. In this case, we say that $g(x_3, \neg x_1, \neg x_2)$ is *isomorphic* to $g(x_1, x_2, x_3)$ under PN^3. In general, for any two Boolean functions f_1 and f_2 defined over n Boolean variables, we say that f_1 is isomorphic to f_2 under PN^n if and only if there exists some $op \in PN^n$ such that $op(f_1) = f_2$. Note that, if such op exists, then f_2 is also isomorphic to f_1 since, by definition, the inverse of op must be in PN^n.

Now, imagine that every operator in PN^n was applied to some Boolean function h_1, and let $Closure(h_1)$ be the set of the resulting functions. Clearly, for any $p, q \in Closure(h_1)$, p is isomorphic to q. Conversely, any concept $r \notin Closure(h_1)$ cannot be isomorphic to any concept in $Closure(h_1)$. Suppose we again apply all the operators in PN^n to another function $h_2 \notin Closure(h_1)$, and let the resulting set of functions be $Closure(h_2)$. Clearly, it must be true that $Closure(h_1) \bigcap Closure(h_2) = \phi$. Thus, continuing in this fashion, the space of all Boolean functions is partitioned into disjoint sets, such that every two functions are isomorphic if and only if they belong to the same set. Each of such sets will be called an *equivalence class* under PN^n.

Defining equivalence classes in the above manner is interesting to us here because many learning algorithms behave identically in learning all the concepts (functions) that belong to the same equivalence class. Thus, if the goal is to evaluate the performance of a learning algorithm for all the space of concepts, then all we need is to test this algorithm on a *representative* from each equivalence class (which can be *any* concept in the equivalence class). In the following, let us us see how a a complete list of representatives con be computed.

Let T_c be the truth table representation of a concept c. If we consider a fixed ordering on the space of objects, then, of course, the right-most column of T_c is

sufficient to represent c. Let us call the 2^n-bit string obtained by transposing this column the *bit-vector representation* of c. For example, suppose $n = 3$ and let $c_1 = x_1 \wedge x_2$ and $c_2 = \neg x_1 \neg x_2 \neg x_3$. Then, these concepts are represented as $c_1 = 00000011$ and $c_2 = 10000000$.

The bit-vector representation of a concept can also be viewed as a 2^n-bit integer. Let us call this the *integer-value* of a concept. This imposes a natural ordering ">" on the space of concepts where $concept_1 > concept_2$ if the integer-value of $concept_1$ is larger than that of $concept_2$. In the above example, the integer values of c_1 and c_2 are 3 and 128, respectively, and thus, we can say that $c_2 > c_1$.

Notice that the representative of an equivalence class can by any concept in that class. Suppose that we choose to let the concept with the highest integer-value be the representative for each class. Then, a straightforward method for finding a complete list of representatives is as follows:

> For each concept c_i, if there exists some $op \in PN^n$ such that $op(c_i) > c_i$, then c_i is not a representative. Otherwise, include c_i in the set of representatives.

Note that the number of concepts to be tested is 2^{2^n}, and thus, the above procedure is manageable for n only up to 4, where the number of concepts is $2^{2^4} = 65536$. However, for $n = 5$, this number becomes more than 4 billion which makes it necessary to seek a more efficient method. A reasonable procedure that works for this case is given in Figure 1.

First, let us define expanding a concept c to its *Successors* as follows: Let us number the bits in the bit-vector representation of c from left to right starting from 0 (that is, the left-most bit is the 0-th bit and the right-most bit is the 2^n-1-th bit). Let j be the position of the right-most 1 in c. Then the successors of c are all the concepts generated by setting the k-th bit of c to 1, for $k = j+1, j+2, \ldots, 2^n - 1$. If the right-most bit is 1 (i.e., if $j = 2^n - 1$), then c has no successors.

For example, if $c = 10001000$, then $Successors(c)$ gives the concepts 10001100,

Algorithm: Find-Representatives(n)

1. $REP = \phi$
2. Insert the concept $\overbrace{000\ldots00}^{2^n}$ in Q

   /* Q is a first-in-first-out queue */
3. Repeat until Q is empty
   - 3.1. Let c be the concept at the head of Q
   - 3.2. Remove c from Q
   - 3.3. If for all $op \in PN^n, op(c) \leq c$, then
     - 3.3.1. Add c to REP
     - 3.3.2. If $weight(c) < 2^{n-1}$ then insert $Successors(c)$ in Q
4. Return REP

end.

Figure 1. Algorithm for finding a set of representative concepts. The *weight* of a concept is the number of 1's in it.

10001010 and 10001001.

Now, suppose that we expand the concept $000\cdots000$ (the *nil* concept) to its successors, and then expand each of these to its successors, and so on until no more successors can be generated. The reader can check that this enumerates all the concepts in the space.

The idea of the algorithm of Figure 1 is to eliminate redundant tests by exploiting the following observation:

> If g is not a representative, then all the successors of g are not representatives either.

To see why this observation holds, let us define the *weight* of a concept as the number of 1's in that concept. It can easily be verified that if two concepts a and b have the same weight, and if $a > b$, then a is greater than any successor of b. Given that "a concept g is not a representative" implies that there exists some $\gamma \in PN^n$ such that $\gamma(g) > g$. Let $h = \gamma(g)$. It should be clear that h and g have the same weight since γ only permutes and/or negates the variables and thus only permutes the bits of g. Now, let g' be a successor of g and let $h' = \gamma(g')$. Since g' is a successor of g, g' is identical to g except that one bit in g is turned to 1 in g'. The same thing can also be said about h and h' (that is, h and h' are identical except for one bit which is 0 in h and 1 in h'). Therefore, the following must be true about h, h', g and g':

$h' > h$	since all bits are matching except one that is 1 in h' and 0 in h.
$h > g$	by definition of $h = \gamma(g)$ which is greater than g.
$h > g'$	because (i) g' is a successor of g, (ii) $h > g$, and (iii) h and g have the same weight.

These relations imply that $h' > g'$. But h' and g' are in the same equivalence class, and hence, g' is not the concept with the highest integer-value in the class—that is, g' not a representative.

In Figure 2, we show a trace run of the algorithm for $n = 3$. The tree of this

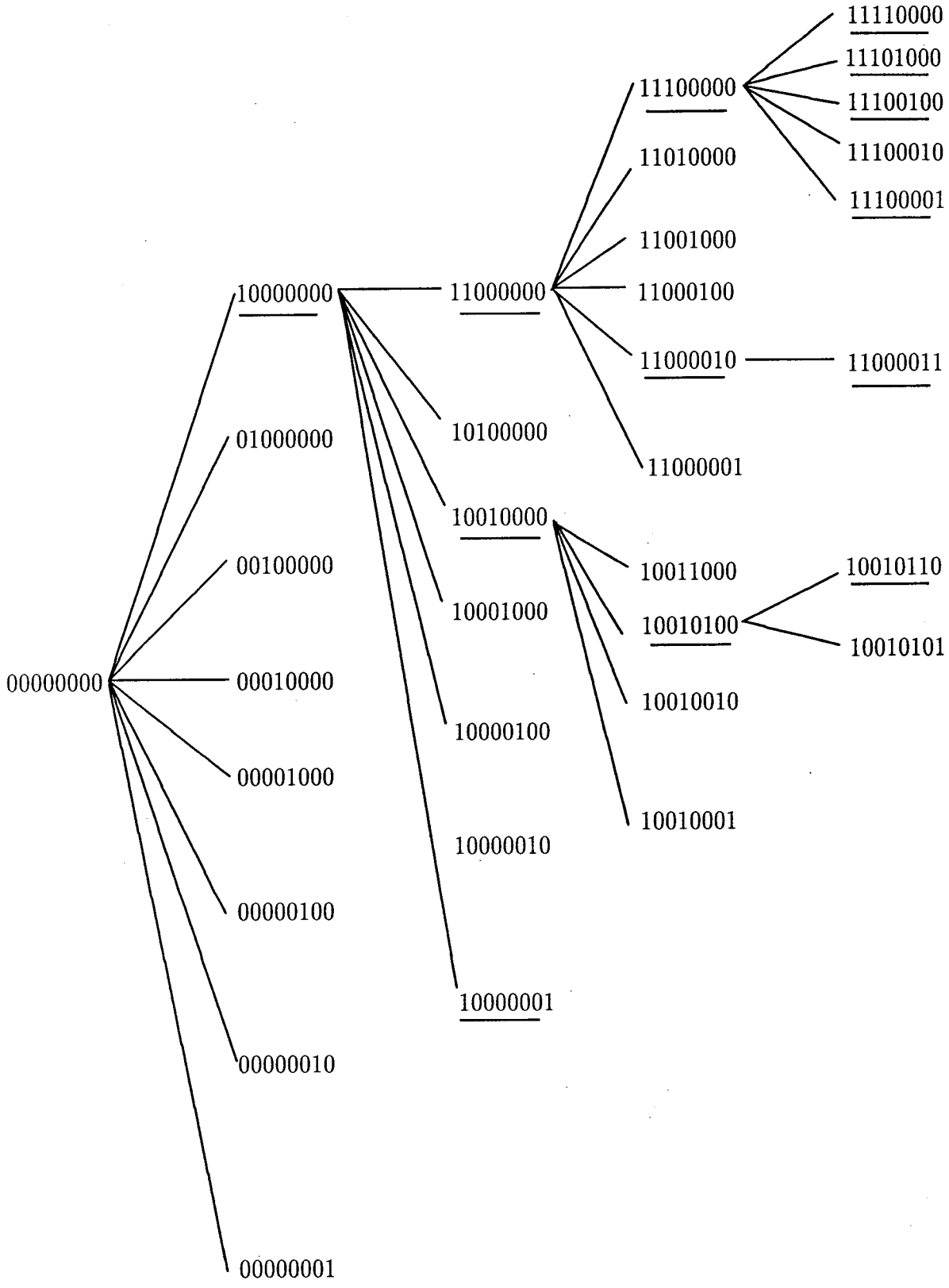


Figure 2. Finding a set of representatives for $n = 3$. Representatives are underlined.

figure is traversed by the algorithm in a breadth-first fashion. The algorithm starts by expanding 00000000 to its successors. Each of these are not representatives except 10000000. Therefore, only this concept is expanded. Only three successors of this concept are found representatives and thus only these are expanded, and so on.

Notice that we do not need to continue this procedure beyond the concepts of weight 2^{n-1} since the rest of the representatives (those of weight $2^{n-1}-1$ to 2^n) can be obtained simply by complementing the representatives of weight 0 to $2^{n-1}-1$. The representatives of weight $2^{n-1}-1$ to 2^n obtained in this way are not necessarily the largest ones in their classes, but this, of course, does not matter.

Now, suppose we let the set of operators be PNC^n which contains all the operators of PN^n in addition to an operator that complements a Boolean function. That is, an operator *op* such that for any function f, $op(f) = \neg f$. Then, *equivalence classes* can also be defined under PNC^n in the same manner as given above for PN^n, and representatives of these classes can be computed by the algorithm of Figure 1. In this case, however, we do not need to complement the representatives of weight 0 to $2^{n-1}-1$ as we did for the case of PN^n.

For a detailed discussion on isomorphism of Boolean functions (or on "permutation groups" in general), we refer the reader to [Harrison 65] and [Slepian53]. The algorithm of Figure 1, however, is not discussed in these references and is a contribution of this thesis.

Before concluding this section, we should mention that this algorithm helps in finding the set of representatives for n up to 5. For $n = 5$, an efficient C implementation of this algorithm produced the set of all 698,635 representatives for PN^5 in less than an hour of CPU time on a SUN Sparcstation-1. However, 5 variables is the maximum that we can handle. The immense number of representatives (more than 2×10^{14} for $n = 6$) prevents going beyond that.

2.5 The ID3 Learning Algorithm

The ID3 learning algorithm, introduced by Quinlan [Quinlan 86], takes as input a set of pre-classified training examples and produces a *decision tree* as its hypothesis. A decision tree is a tree structure in which

1. each leaf is labeled with a class; either $-$ or $+$,
2. each node is labeled with a feature, and
3. each edge is labeled with a value of the feature labeling the edge's parent node.

We consider here only the case where features are Boolean, and thus, each node has two out-going edges, one for each of the *true* and *false* values for the feature tested at that node.

To determine the class of a new object as predicted by a decision tree, we start at the root of the tree and follow the tree down until a leaf is reached. At each node, we branch on the *true* or the *false* out-going edge depending on the value of the feature of that node as given by the object to be classified. The class of the object is the label of the encountered leaf.

Given a training sample, it is easy to construct a decision tree consistent with the sample. This can be done recursively as follows: First, the simplest case is when all the examples have one common class. In this case we just "return that class"—that is, the tree consisting of a single node which is a leaf and is labeled either $+$ or $-$, depending on whether the examples were all positive or all negative. Else, if there is a mixture of positive and negative examples in the sample, then we pick up any feature x and split the training sample into two groups: the examples that have $x = 0$ and those that have $x = 1$. We then repeat the same procedure recursively on each of these groups. Then, we return a decision tree in which x is the root and the two decision trees returned by the recursive calls are the two subtrees of x.

The above is exactly what ID3 (Figure 3) does except that ID3 attempts to make the resulting decision tree as compact as possible by following an information-based heuristic for choosing the feature x (the root of the tree) in each recursive call. Under this criterion, the score of a feature x_i is computed as

$$-\frac{p_0+n_0}{p+n}\left[\frac{p_0}{p_0+n_0}\log_2\frac{p_0}{p_0+n_0}+\frac{n_0}{p_0+n_0}\log_2\frac{n_0}{p_0+n_0}\right]- \\ \frac{p_1+n_1}{p+n}\left[\frac{p_1}{p_1+n_1}\log_2\frac{p_1}{p_1+n_1}+\frac{n_1}{p_1+n_1}\log_2\frac{n_1}{p_1+n_1}\right] \quad (2.3)$$

where p, n are the number of positive and negative examples, p_0, n_0 the number of positive and negative examples in which $x_i = 0$, and p_1, n_1 the number of positive and negative examples in which $x_i = 1$. The above quantity estimates the expected number of features that are yet to be tested down the tree. Therefore, the algorithm chooses the feature for which the above quantity is minimized.

For a detailed discussion of ID3, we refer the reader to [Quinlan 86].

2.6 The FRINGE Learning Algorithm

FRINGE is an algorithm that dynamically creates and uses *new* features defined as Boolean combinations of the original (primitive) features. This algorithm was introduced by Pagallo and Haussler [Pagallo and Haussler 90] [3]. The algorithm begins with a set V of primitive features and employs ID3 to build a decision tree out of a set of training examples expressed only in terms of the features in V. Then, an *extract-feature* heuristic (described below) generates new features as Boolean combinations of the features that are tested in the nodes near the *fringe* of the tree. The set of new features is added to the feature set, V, and the training examples are augmented to include the newly defined features. ID3 is again called to build a decision tree using the expanded feature set, V. The process is repeated until one of the following occurs:

[3] Large part of this section was copied from [Bakiri 91]

Algorithm ID3 (*Sample*)

1. If all the examples are positive return +.
2. If all the examples are negative return −.
3. Compute the score for each feature as given in Equation (2.3).
4. Let x_i be a feature with the minimum score.
5. Let *Sample0* be the set of examples in which $x_i = 0$.
6. Let *Sample1* be the set of examples in which $x_i = 1$.
7. Return a tree in which

   x_i is the root

   ID3(*Sample0*) is the sub-tree down the edge labeled $x_i = 0$, and

   ID3(*Sample1*) is the sub-tree down the edge labeled $x_i = 1$.

end.

Figure 3. The ID3 algorithm.

- No new features are defined by the extract-feature procedure.
- The decision trees output by ID3 are identical for two successive iterations.
- A predefined maximum number of iterations is reached.
- A predefined maximum number of newly defined features is reached.

The *extract-feature* procedure takes as input a binary decision tree and outputs the set of *newly-defined features.* It scans the tree, and, for every positive leaf l (at depth > 1 from the root), it defines a new feature as a *conjunction* involving the features tested at the parent node, p, and the grand parent node, g, of the leaf l as follows:

let v_p and v_g be the test features at nodes p and g, respectively.
if l is on the right subtree of p and p is on the right subtree of g then
    define $v_p \wedge v_g$ as a new feature
else if l is on the right subtree of p and p is on the left subtree of g then
    define $v_p \wedge \neg v_g$ as a new feature
else if l is on the left subtree of p and p is on the right subtree of g then
    define $\neg v_p \wedge v_g$ as a new feature
else l must be on the left subtree of p and p on the right subtree of g
    define $\neg v_p \wedge \neg v_g$ as a new feature.

(The above assumes that the decision tree adopts the convention that the left edge represents the negative or 0 outcome of the feature tested at the node, and the right edge represents the positive or 1 outcome.)

The procedure generates simple features at each step, but it adaptively assembles more complex combinations of the attributes through the iterative process. So, the k^{th} iteration may produce features of size up to 2^k.

Details about the FRINGE algorithm can be found in [Pagallo and Haussler 90].

2.7 Useful Inequalities

In this section, we state four lemmas that we will repeatedly use throughout the thesis.

Consider a random experiment conducted by t independent trials, each of which results in a "success" with probability p or "failure" with probability $1-p$. If Y is the number of successes that occur in this experiment, then Y is called a *binomial* random variable with t trails and p as the ratio of success. The probability that Y takes a given value i, $0 \leq i \leq t$, is given by

$$\binom{t}{i} p^i (1-p)^{t-i}$$

The first two lemmas of this section give upper bounds on the probability that the value of Y goes far above or far below pt, the expected value of Y. These lemmas show that this probability decreases exponentially in the number of trials and in the squared difference between the value of Y and its expected value.

Lemma 2.1 (Hoeffding) *Let Y be a binomial random variable with t trials and p as the ratio of success. Then*

$$Pr[Y \geq rt] = \sum_{i=\lceil rt \rceil}^{t} \binom{t}{i} p^i (1-p)^{t-i} \leq e^{-2(r-p)^2 t}$$

for any r such that $p \leq r \leq 1$, and

$$Pr[Y \leq rt] = \sum_{i=0}^{\lfloor rt \rfloor} \binom{t}{i} p^i (1-p)^{t-i} \leq e^{-2(p-r)^2 t}$$

for any r such that $0 \leq r \leq p$.

A proof of this result can be found in [Hoeffding 63]. The second lemma, proven by Chernoff [Chernoff 52], gives a better bound on the same quantity for the cases where p is small.

Lemma 2.2 (Chernoff) *Let Y be a binomial random variable with t trials and p as the ratio of success. Then*

$$Pr[Y \geq rt] = \sum_{i=\lceil rt \rceil}^{t} \binom{t}{i} p^i (1-p)^{t-i} \leq e^{-\frac{(r-p)^2}{3p} t}$$

for any r *such that* $p \leq r \leq 1$, *and*

$$Pr[Y \leq rt] = \sum_{i=0}^{\lfloor rt \rfloor} \binom{t}{i} p^i (1-p)^{t-i} \leq e^{-\frac{(p-r)^2}{2p} t}$$

for any r *such that* $0 \leq r \leq p$.

The above two lemmas are helpful in stating closed form upper bounds on the probability that a binomial random variable greatly deviates from its expected value.

Next, the third lemma of this section, which will be frequently used in simplifying our results, is stated as follows.

Lemma 2.3 *For any* ϵ *such that* $0 < \epsilon < 1$

$$1 - \epsilon < e^{-\epsilon}$$

Proof:

For $\epsilon = 0$, $1 - \epsilon = e^{-\epsilon}$, and the derivative of $1 - \epsilon - e^{-\epsilon}$ is negative for $0 < \epsilon < 1$.

□

Quantities in the form $\sum_{i=0}^{\epsilon k} \binom{k}{i}$ are repeatedly encountered in the formal work of this thesis. The last lemma of this section (from [MacWilliams and Sloane 77], Corollary 9 of Chapter 10) helps in providing a convenient closed-form upper bound on such sum of binomials.

Lemma 2.4 *For any* $0 < \epsilon < \frac{1}{2}$,

$$\sum_{i=0}^{\epsilon k} \binom{k}{i} \leq 2^{kH(\epsilon)}$$

where $H(\epsilon) = -\epsilon \log_2 \epsilon - (1-\epsilon) \log_2 (1-\epsilon)$.

Proof:

For any integer r, it must be true that

$$\begin{aligned} 2^{r(1-\epsilon)k} \sum_{i=0}^{\epsilon k} \binom{k}{i} &\leq \sum_{i=0}^{\epsilon k} 2^{r(k-i)} \binom{k}{i} \\ &\leq \sum_{i=0}^{k} 2^{r(k-i)} \binom{k}{i} \\ &= \sum_{j=0}^{k} 2^{rj} \binom{k}{k-j} \\ &= \sum_{j=0}^{k} 2^{rj} \binom{k}{j} \\ &= (1+2^r)^k \end{aligned}$$

Thus,

$$\begin{aligned} \sum_{i=0}^{\epsilon k} \binom{k}{i} &\leq \frac{(1+2^r)^k}{2^{r(1-\epsilon)k}} \\ &= \left[2^{-(1-\epsilon)r} + 2^{\epsilon r}\right]^k \end{aligned}$$

Now, letting $r = \log_2 \frac{1-\epsilon}{\epsilon}$ (which is clearly positive for $0 < \epsilon < \frac{1}{2}$), the last quantity is equal to

$$\begin{aligned} & \left[2^{-(1-\epsilon)\log_2(1-\epsilon)+(1-\epsilon)\log_2\epsilon} + 2^{\epsilon\log_2(1-\epsilon)-\epsilon\log_2\epsilon}\right]^k \\ = & \left[2^{-(1-\epsilon)\log_2(1-\epsilon)-\epsilon\log_2\epsilon+\log_2\epsilon} + 2^{-(1-\epsilon)\log_2(1-\epsilon)+\log_2(1-\epsilon)-\epsilon\log_2\epsilon}\right]^k \\ = & \left[2^{H(\epsilon)}\left\{2^{\log_2\epsilon} + 2^{\log_2(1-\epsilon)}\right\}\right]^k \\ = & \; 2^{kH(\epsilon)}[\epsilon + 1 - \epsilon]^k \\ = & \; 2^{kH(\epsilon)} \end{aligned}$$

which shows the lemma. □

Chapter 3

On Learning More Concepts

3.1 Motivation

Research in computational learning theory (e.g., [Valiant 84], [COLT88]–[COLT91], [Natarajan91]) has provided many insights into the capabilities and limitations of inductive learning from examples. However, an important shortcoming of most work in this area is that it focuses on learning concepts drawn from *prespecified classes* of concepts (e.g., linearly separable functions, k-DNF formulae). This style of research begins by choosing a concept class and then finding a polynomial bound—called the *sample complexity*—such that if a sample of size larger than the sample complexity is available, any concept from the concept class that is consistent with the sample will be approximately correct with high probability.

Work of the above type usually leads to a learning algorithm that is *specialized* in learning the prescribed class of concepts, and an upper bound on the number of training examples required by the algorithm to guarantee successful learning of every concept in the class.

For real-world applications, such findings can be viewed as follows: A learning algorithm L designed to learn a class of concepts $\mathcal{C}$ is guaranteed to succeed[4] in application domains in which the target concept belongs to $\mathcal{C}$ (i.e., the restrictions

[4]Note that the guarantees are on being approximately correct with high confidence.

used to define $\mathcal{C}$ are satisfied by the target concept), provided that a sufficient number of training examples is given to the algorithm. Of course, no such guarantees are given if the target concept is not in $\mathcal{C}$.

This naturally means that one should seek algorithms that learn concept classes that are as large (i.e., less restricted) as possible. Rivest [Rivest 87], for instance, mentions this goal most explicitly by saying:

> "One goal of research in machine learning is to identify the largest possible class of concepts that are learnable from examples."

This goal is also declared (although less explicitly) in many papers in the related literature ([Valiant 84], [Natarajan 87], [COLT88]–[COLT91]).

Nevertheless, it is a well-known fact that learning larger classes of concepts necessarily requires a larger number of training examples [Blumer *et al.* 87a]. Such trade-off between the size of the class of concepts being learned and the required number of training examples dictates how far one can go in attempting to learn larger and larger classes of concepts.

Traditionally, this issue has been addressed by identifying new classes of concepts that are as large as possible but still require a training sample of size bounded by some polynomial. For example, as an improvement of the learnability results of k-dnf and k-cnf classes[Valiant 84], Rivest defined a new class of concepts which he called k-DL (k-Decision-Lists) that is a superset of both k-dnf and k-cnf, and gave an algorithm that learns this class in polynomial time from a polynomial number of examples [Rivest 87].

Such an approach, however, does not enjoy a great practical merit. In fact, the idea of learning prescribed classes of concepts in general suffers from two important problems:

- Training examples are usually hard to obtain. In a typical inductive learning task, one has only a limited number of training examples, much less than the polynomial bounds provided by learning theory.

- The concept class is usually unknown. In most application settings, there is often considerable flexibility (and concomitant lack of prior knowledge) concerning the choice of which concept class to explore. In fact, many of the concept classes studied in computational learning theory have never been supported by any practical justification.

Due to these difficulties, the learning algorithms and sample complexity bounds developed in computational learning theory have rarely been of practical value.

Recently, an alternative theoretical framework was introduced [Dietterich 89]. Instead of fixing a class of concepts and then deriving the sample complexity, this framework turns the problem around by asking: Given a *fixed* number of training samples, what is the largest collection of concepts that some algorithm can learn? The intuition behind this framework is that, in the absence of additional information, one should prefer the learning algorithm that has the highest chance of learning the unknown concept—that is, the algorithm that learns the largest number of concepts. In short, this framework could provide an approach to discovering an "optimal" bias for inductive learning in the absence of prior knowledge.

The goal of this chapter is to explore this approach. We define the *coverage* of a learning algorithm to be the number of concepts learned by the algorithm from a given sample size (and other relevant parameters). There are three questions raised by this approach:

1. For given sample size m, accuracy parameter ϵ and confidence parameter δ, what is the largest possible coverage that any algorithm can achieve?

2. Can we design a learning algorithm that attains this optimal coverage?

3. What is the coverage of existing inductive learning algorithms?

Dietterich's work included an upper bound on the number of concepts any learning algorithm can learn for given n, m, ϵ and δ. It also included empirical

measurement of the performance of some existing learning algorithms. These results were obtained by running exhaustive experiments on the space of concepts defined on 3 Boolean features. The prohibitive computational costs involved in these experiments prevented carrying out the same experiments on a larger number of features.

The contribution of this thesis to the study of maximum coverage learning can be summarized as follows:

- Generalizing Dietterich's upper bound: The upper bound given in [Dietterich 89] is shown for the case where the training examples are drawn without replacement, i.e. all training examples are assumed to be distinct. We generalize this upper bound to the case where sampling is done with replacement. In addition, we provide a closed-form formula for this bound.

- Learning algorithms with nearly-optimal coverage: We introduce new learning algorithms and show analytically that these algorithms achieve high coverage. An algorithm called "Multi-Balls" is shown to require a number of examples that is within a constant factor of that required by an optimal algorithm to achieve a given coverage.

- Extended experimental results: We introduce new techniques for reducing the computational costs involved in the empirical measurement of coverage. Applying these techniques, we report experimental results that are analogous to those of [Dietterich 89] but carried out over the space of concepts defined on 5 Boolean features.

3.2 Definitions and Notation

Throughout this chapter, we will always assume the uniform distribution over U_n. However, all our results can be easily extended to the distributions where the probability is 0 on a subset of U_n and uniform on the rest. This is done by

substituting the number of instances in U_n having non-zero probability in place of every occurrence of 2^n in the results.

We consider both cases of sampling with and without replacement, where this will always be clear from the context. Note that we usually assume that the sample is relatively small compared to the whole space of examples (i.e., $m \ll 2^n$), and thus, the difference between the two cases of sampling with or without replacement is not significant.

Because of the uniform distribution assumption, it is convenient to view a concept as a point in a 2^n-dimensional cube, i.e. a bit vector of length 2^n. For example, if $n = 4$ and c is the concept given by the formula $\neg x_1 \neg x_2 \neg x_3 \neg x_4 \vee x_1 x_2$, then c corresponds to the bit vector

$$1000\ 0000\ 0000\ 1111$$

where the i-th bit is just the value of $c(X_i)$ for X_i being the assignment to x_1, x_2, x_3 and x_4 as specified by the binary representation of i in the obvious manner. In other words, this is just the (transposed) right-most column of the truth table of the concept.

The *Hamming distance* (or simply, the *distance*) between two concepts c_1 and c_2, denoted $distance(c_1, c_2)$, is just the number of non-matching bits in the bit vector representations of c_1 and c_2. Thus, the error between two concepts (as defined previously in Section 2.1) is equivalent to the distance between the two concepts divided by 2^n. That is,

$$error(c_1, c_2) = distance(c_1, c_2)2^{-n}.$$

For a concept c and a fraction α, $0 \leq \alpha \leq 1$, we define a *ball* of concepts with *center* c and *radius* $\lfloor \alpha 2^n \rfloor$ as the set of all concepts that have distance at most $\lfloor \alpha 2^n \rfloor$ from c. More formally

$$Ball(c, \alpha) = \{c' \mid distance(c, c') \leq \lfloor \alpha 2^n \rfloor\}.$$

It should be clear that for any two concepts c_1, c_2 in a ball of radius r, $distance(c_1, c_2) \leq 2r$. It must also be clear that the number of concepts in a ball is given by

$$|Ball(c, \alpha)| = \sum_{i=0}^{\lfloor \alpha 2^n \rfloor} \binom{2^n}{i}.$$

When a sample S is consistent with two concepts c_1 and c_2, we say that S is *shared* by c_1 and c_2. If the distance between the two concepts is d, then this means that there are $2^n - d$ objects on which both concepts agree and d objects on which they disagree. A sample that is shared by c_1 and c_2 must contain only examples from the $2^n - d$ on which the two concepts agree. Clearly, the larger the value of d the smaller the number of shared samples of any given size.

The disagreement between a sample S and a concept c is defined as follows:

$$disagreement(c, S) = \sum_{\langle X,\ class \rangle \in S} I(X,\ class, c)$$

where $I(X,\ class, c) = 1$ if $c(X) \neq\ class$, and 0 otherwise. In other words, the disagreement between c and S is just the number of examples in S that are misclassified by c. Note that for the case of sampling with replacement, the same example may appear more than once in the sample. The above definition states that *each* occurrence of an example contributes an increment of 1 to the disagreement count.

For sample size m, accuracy parameter ϵ and confidence parameter δ, we will use the term *coverage* of a learning algorithm to mean the number of concepts learned by the algorithm with respect to m, ϵ and δ, under the uniform distribution.

The accuracy and confidence parameters ϵ and δ are generally in the range $0 < \epsilon, \delta < 1$. In practice, however, only values that are very close to zero are interesting. For this reason, we will sometimes derive results that explicitly assume for instance that $0 < \epsilon < \frac{1}{4}$ and/or $0 < \delta < \frac{1}{2}$, with the understanding that these are practically reasonable assumptions. Also, to simplify our results, we will only consider the values of ϵ such that $\epsilon 2^n$ is a positive integer. All our results, however, can easily be modified to include the cases where $\epsilon 2^n$ is not an integer. Finally, another reasonable assumption that we will frequently base our results on is that

the training sample constitutes only a small fraction of the space of all objects (e.g. $m < \frac{1}{4}2^n$).

3.3 Upper Bound on Coverage

In our search for learning algorithms with optimal coverage, the first problem we should address is to find out how large the coverage of any learning algorithm can be. An upper bound of this type has been proven by Dietterich in [Dietterich 89] for the case of sampling without replacement. In the following, we generalize Dietterich's result and show that the same upper bound also holds for the case where sampling is done with replacement. In addition, we provide a closed-form formula for this bound.

Theorem 3.1 *Assuming that* $0 < \epsilon < \frac{1}{2}$ *and that* $m < (1 - 2\epsilon)2^n$, *for both cases of sampling with and without replacement, the coverage of any learning algorithm under the uniform distribution can not exceed*

$$\frac{2^m \sum_{i=0}^{\epsilon 2^n} \binom{2^n - m}{i}}{1 - \delta}$$

concepts, for sample size m, *accuracy parameter* ϵ *and confidence parameter* δ.

Theorem 3.2 *For* $0 < \epsilon < \frac{1}{4}$ *and* $m < \frac{1}{4}2^n$, *the quantity of Theorem 3.1 is further bounded above by*

$$\frac{2^{(1-\epsilon \log_2 e)m+1} \binom{2^n}{\epsilon 2^n}}{1 - \delta} \leq \frac{2^{(1-1.44\epsilon)m+1+H(\epsilon)2^n}}{1 - \delta}$$

where e *is the base of the natural logarithm, and* $H(\epsilon) = \epsilon \log_2 \frac{1}{\epsilon} + (1 - \epsilon) \log_2 \frac{1}{1-\epsilon}$.

The following lemmas are needed to prove the above two theorems.

Lemma 3.1 *For any set* S *containing* k *distinct training examples and any concept* $h \in 2^{U_n}$, *the number of concepts that are consistent with the training examples in* S *and within* ϵ *of* h *is at most*

$$\sum_{i=0}^{\epsilon 2^n} \binom{2^n - k}{i}.$$

Proof:

In how many ways can we construct a concept c such that c is consistent with S and within ϵ of h? For the k bits of c that correspond to the k examples of S, we have no choice but to follow the classification of these examples as given in S. For the remaining $2^n - k$ bits, we can afford to disagree with h on at most $\epsilon 2^n$ bits. Thus, there are at most

$$\sum_{i=0}^{\epsilon 2^n} \binom{2^n - k}{i}$$

ways to construct such c, and the lemma follows. □

Lemma 3.2 *For any $0 < \epsilon < \frac{1}{2}$ and any integer j such that $0 \leq j \leq \epsilon 2^n$, the quantity*

$$2^k \binom{2^n - k}{j}$$

is monotonically increasing in k in the range $1 \leq k < (1 - 2\epsilon)2^n$.

Proof:

$$\begin{aligned}
\frac{2^{k+1}\binom{2^n-k-1}{j}}{2^k\binom{2^n-k}{j}} &= 2 \cdot \frac{(2^n - k - 1)(2^n - k - 2)\cdots(2^n - k - j)}{(2^n - k)(2^n - k - 1)\cdots(2^n - k - j + 1)} \\
&= 2 \cdot \frac{2^n - k - j}{2^n - k} \\
&= 2 \cdot (1 - \frac{j}{2^n - k}) \\
&> 2 \cdot (1 - \frac{\epsilon 2^n}{2\epsilon 2^n}) \quad \text{(by substituting } j = \epsilon 2^n \text{ and } k = (1 - 2\epsilon)2^n \text{)} \\
&= 1
\end{aligned}$$

This shows the lemma. □

Lemma 3.3 *For any ϵ such that $\epsilon < \frac{1}{2}$ and such that ϵk is an integer*

$$\sum_{i=0}^{\epsilon k} \binom{k}{i} \leq \frac{1 - \epsilon}{1 - 2\epsilon} \binom{k}{\epsilon k}.$$

Proof:

The left-hand side of the inequality is

$$\sum_{i=0}^{\epsilon k} \binom{k}{i} = \binom{k}{\epsilon k} + \binom{k}{\epsilon k - 1} + \cdots \binom{k}{1} + \binom{k}{0}.$$

If we divide the second term by the first, the third by the second, the fourth by the third, and so on, we get

$$\frac{\epsilon k}{k - \epsilon k + 1}, \frac{\epsilon k - 1}{k - \epsilon k + 2}, \frac{\epsilon k - 2}{k - \epsilon k + 3}, \cdots \frac{2}{k-1}, \frac{1}{k}.$$

Note that these ratios are monotonically decreasing. Therefore, we can write

$$\begin{aligned}
\sum_{i=0}^{\epsilon k} \binom{k}{i} &\leq \binom{k}{\epsilon k}\left[1 + \frac{\epsilon k}{k - \epsilon k + 1} + (\frac{\epsilon k}{k - \epsilon k + 1})^2 + \cdots + \right. \\
&\qquad \left. (\frac{\epsilon k}{k - \epsilon k + 1})^{\epsilon k - 1} + (\frac{\epsilon k}{k - \epsilon k + 1})^{\epsilon k}\right] \\
&= \binom{k}{\epsilon k} \sum_{i=0}^{\epsilon k} \left[\frac{\epsilon k}{k - \epsilon k + 1}\right]^i \\
&\leq \binom{k}{\epsilon k} \sum_{i=0}^{\infty} \left[\frac{\epsilon k}{k - \epsilon k + 1}\right]^i \\
&\leq \binom{k}{\epsilon k} \sum_{i=0}^{\infty} \left[\frac{\epsilon}{1 - \epsilon}\right]^i \\
&= \frac{1 - \epsilon}{1 - 2\epsilon}\binom{k}{\epsilon k}.
\end{aligned}$$

The last step follows by viewing $\sum_{i=0}^{\infty} \left[\frac{\epsilon}{1-\epsilon}\right]^i$ as an infinite sum of a geometric series. This evaluates to $\frac{1-\epsilon}{1-2\epsilon}$ provided that $\epsilon < \frac{1}{2}$. □

Lemma 3.4 *Provided that* $\epsilon < \frac{1}{4}$ *and* $m < \frac{k}{4}$,

$$\sum_{i=0}^{\epsilon k} \binom{k-m}{i} \leq 2e^{-\epsilon m}\binom{k}{\epsilon k}.$$

Proof:

First, note that

$$\frac{\binom{k-m}{\epsilon k}}{\binom{k}{\epsilon k}} = \frac{(k-m)!}{(k - \epsilon k - m)!(\epsilon k)!} \cdot \frac{(k - \epsilon k)!(\epsilon k)!}{k!}$$

$$
\begin{aligned}
&= \frac{(k-\epsilon k)(k-\epsilon k-1)\cdots(k-\epsilon k-m+1)}{k(k-1)\cdots\cdots(k-m+1)} \\
&\leq (1-\epsilon)^m \\
&\leq e^{-\epsilon m} \quad \text{(by Lemma 2.3).} \qquad (3.4)
\end{aligned}
$$

Now, the quantity $\sum_{i=0}^{\epsilon k}\binom{k-m}{i}$ can be rewritten as

$$
\begin{aligned}
\sum_{i=0}^{(\frac{\epsilon k}{k-m})(k-m)}\binom{k-m}{i} &\leq \frac{1-\frac{\epsilon k}{k-m}}{1-2\frac{\epsilon k}{k-m}}\binom{k-m}{\epsilon k} \quad \text{(by Lemma 3.3)} \\
&= \frac{k-m-\epsilon k}{k-m-2\epsilon k}\binom{k-m}{\epsilon k} \\
&\leq \frac{k-m-\epsilon k}{k-m-2\epsilon k}e^{-\epsilon m}\binom{k}{\epsilon k} \text{ (by Equation (3.4)).}
\end{aligned}
$$

The lemma follows by verifying that

$$\frac{k-m-\epsilon k}{k-m-2\epsilon k} \leq 2$$

when $\epsilon < \frac{1}{4}$ and $m < \frac{k}{4}$. This is equivalent to

$$k-m-\epsilon k \leq 2k-2m-4\epsilon k$$

or

$$k \geq m+3\epsilon k\,,$$

which is satisfied by the assumptions of the lemma. □

Proof of Theorem 3.1:

Let L be any learning algorithm and let $C \subseteq 2^{U_n}$ be the set of concepts learned by L for the given learning parameters. Let us denote by p_c the probability that a sample of size m for a concept c be mapped by L to some hypothesis within ϵ of c. By definition, for any $c \in C$, p_c must be at least $1-\delta$. Thus, it must be true that

$$\sum_{c\in C} p_c \geq |C|(1-\delta)$$

and therefore

$$|C| \leq \frac{1}{1-\delta}\sum_{c\in C} p_c \leq \frac{1}{1-\delta}\sum_{c\in 2^{U_n}} p_c$$

since $C \subseteq 2^{U_n}$. In the following, we let

$$W = \sum_{c \in 2^{U_n}} p_c \, .$$

The theorem holds by proving an upper bound on W as follows.

Let $\mathcal{S}$ be the set of all possible outcomes of randomly drawing m objects from U_n. For any $s \in \mathcal{S}$, let $\sharp s$ denote the number of distinct objects in s. Note that $\sharp s = m$ for every $s \in \mathcal{S}$ in the case of sampling without replacement, and that $1 \leq \sharp s \leq m$ in the case of sampling with replacement. Also, let $Pr[s]$ denote the probability of the outcome s.

Now, for some $s \in \mathcal{S}$, suppose t is a training sample obtained by arbitrarily labeling the objects in s as positive or negative, and let h be the hypothesis returned by L when given t. Mapping t to h contributes the amount of $Pr[s]$ to p_c for every concept c that is (i) consistent with t and (ii) within ϵ of h. Therefore, we can compute W by summing up this contribution for all possible outcomes $s \in \mathcal{S}$ and all possible ways of labeling these. That is,

$$W = \sum_{s \in \mathcal{S}} Pr[s] \sum_{t \in Labeling(s)} N(t)$$

where $Labeling(s)$ is the set of all training samples obtained by labeling s, and $N(t)$ is the number of concepts that are consistent with t and within ϵ from the hypothesis returned by L when given t. By Lemma 3.1, for any $t \in Labeling(s)$

$$N(t) \leq \sum_{i=0}^{\epsilon 2^n} \binom{2^n - \sharp s}{i}$$

which means that

$$\begin{aligned} W &\leq \sum_{s \in \mathcal{S}} \left\{ Pr[s] \times \mid Labeling(s) \mid \times \sum_{i=0}^{\epsilon 2^n} \binom{2^n - \sharp s}{i} \right\} \\ &= \sum_{s \in \mathcal{S}} \left\{ Pr[s] \times 2^{\sharp s} \times \sum_{i=0}^{\epsilon 2^n} \binom{2^n - \sharp s}{i} \right\} \\ &= \sum_{s \in \mathcal{S}} \left\{ Pr[s] \times \sum_{i=0}^{\epsilon 2^n} 2^{\sharp s} \times \binom{2^n - \sharp s}{i} \right\} \end{aligned}$$

For the case of sampling without replacement $\sharp s$ is always equal to m and for the case of sampling with replacement $\sharp s$ is at most m. Therefore, for both cases using Lemma 3.2 we get

$$\begin{aligned} W &\leq \sum_{s \in \mathcal{S}} Pr[s] \sum_{i=0}^{\epsilon 2^n} 2^m \binom{2^n - m}{i} \\ &= 2^m \sum_{i=0}^{\epsilon 2^n} \binom{2^n - m}{i} \sum_{s \in \mathcal{S}} Pr[s] \\ &= 2^m \sum_{i=0}^{\epsilon 2^n} \binom{2^n - m}{i} \end{aligned}$$

which proves the theorem. □

Proof of Theorem 3.2:

The first inequality follows directly from Lemma 3.4. The second inequality follows from Lemma 2.4. □

Theorems 3.1 and 3.2 show that given a training sample of a reasonable size, any learning algorithm can learn only a small proportion of the concept space. As a numerical example, consider the case where $n = 20, m = 100,000$ and $\delta = \epsilon = 0.05$. Note that these are all reasonable figures. While domains with much more than 20 features are not unusual, 100,000 examples is a rather large sample size. In this case $H(\epsilon) \approx 0.286$. The above results states that no learning algorithm can learn more than

$$\frac{2^{92,788+0.286 \times 2^{20}}}{0.95}$$

concepts. This is less than $\frac{1}{600,000}$ of the $2^{2^{20}}$ possible concepts definable over 20 features—a strikingly small fraction.

For the extreme case where $\delta = 0$, we can even derive a much tighter bound:

Theorem 3.3 *If $\delta = 0$ and $0 < \epsilon < \frac{1}{4}$, and if sampling is done with replacement under the uniform distribution, then for any $m \geq 1$, the coverage of any learning algorithm is at most $\sum_{i=0}^{\epsilon 2^n} \binom{2^n}{i}$ concepts, for accuracy parameter ϵ and confidence parameter δ.*

Proof:

Let L be any learning algorithm. Any two concepts c_1, c_2, such that $c_1 \neq \neg c_2$, must share some samples in common. Since $\delta = 0$, for L to learn both concepts, L must map *all* of these samples to some hypothesis h that is within ϵ of both concepts. For such an h to exist, it must be true that $distance(c_1, c_2) \leq 2\epsilon 2^n$.

Now, suppose that C is the set of concepts learned by L for $\delta = 0$, $\epsilon < \frac{1}{4}$ and some particular m. One of the following two cases must hold:

- There exists no concept c such that both c and $\neg c$ are in C. This implies that the distance between every pair of concepts is at most $2\epsilon 2^n$, and hence, $|C| \leq \sum_{i=0}^{\epsilon 2^n} \binom{2^n}{i}$ as desired.

- There exists some concept c such that $c, \neg c \in C$. Let c' be any concept such that $c' \neq c$ and $c' \neq \neg c$. c' must, therefore, share some samples with both c and $\neg c$. Thus, for c' to be learned by L, it must be within $2\epsilon 2^n$ of both c and $\neg c$. However, this is impossible since $\epsilon < \frac{1}{4}$. This implies that C contains no concepts other than c and $\neg c$ which means a coverage of only 2.

This proves the theorem. □

Theorem 3.3 suggests that the upper bound of Theorem 3.1 is not tight when δ is very small. It also implies that the degree of freedom provided by the confidence parameter, δ, in the PAC definition is very important. Any algorithm that does not exploit this freedom to output (with probability δ) a totally incorrect hypothesis can have only very limited coverage.

3.4 The Ball(s) Family of Learning Algorithms

Given these upper bounds on coverage, can we design algorithms that achieve these coverages?

Designing such learning algorithms means

- Step 1: Deciding (based on the values of n, m, ϵ and δ) which concepts are to be learned by the algorithm and which concepts are to be given up.

- Step 2: Constructing an appropriate sample-to-hypothesis mapping that results in learning the concepts chosen in Step 1.

Note that our ability to accomplish Step 2 depends on our choices in Step 1. The goal is, of course, to make the number of concepts desired to be learned in Step 1 as large as possible. However, if this number is too large, then we may not be able to construct a successful sample-to-hypothesis mapping in Step 2. In addition, Step 2 also depends on *which* concepts are chosen to be learned in Step 1, and not only the number of these. That is, a bad choice of the collection of concepts to be learned could prevent constructing an appropriate mapping even if this collection is small.

Let us be more specific. Let c_1 and c_2 be two concepts with distance d, and suppose that we desire to construct a learning algorithm L that learns both c_1 and c_2. To do this, we must consider how L should treat every possible sample consistent with c_1, c_2, or both.

Obviously, any sample that is consistent with only one of c_1 or c_2 can be mapped to some hypothesis that is within ϵ of the consistent concept. The question is how to map a sample that is consistent with both c_1 and c_2. Now, if $\frac{d}{2^n} \leq 2\epsilon$, then there exists some concept h that is within ϵ of both c_1 and c_2. In this case, all we need to do is to map the sample to h. However, if $\frac{d}{2^n} > 2\epsilon$, then there exists no concept that is within ϵ of both c_1 and c_2, and therefore, we must map the sample either in favor of c_1 or in favor of c_2, but not both. In these cases, if the correct concept is c_2 and we map the sample in favor of c_1, we will commit a *mistake*, and we can only afford to do this with probability δ. The probability of getting a sample of size m that is consistent with both c_1 and c_2 is

$$\left(\frac{2^n - d}{2^n}\right)^m$$

when sampling is with replacement and

$$\frac{\binom{2^n-d}{m}}{\binom{2^n}{m}}$$

when sampling is without replacement. These quantities are decreasing as d increases, so if we choose d sufficiently large, we can keep the probability of committing a mistake below δ.

In short, if c_1 and c_2 are close together, then there is no problem, because we can choose an h ϵ-close to both. Conversely, if they are far apart, there is also no problem, because the probability of committing a mistake can be bounded by δ. This suggests that a good strategy for designing learning algorithms with high coverage is to choose a collection of concepts that is as large as possible, such that the distance between each pair of concepts in the collection is either

- sufficiently large to suppress the probability of getting a sample consistent with both concepts, or
- within $2\epsilon 2^n$, so that we can find concept(s) within ϵ of both concepts.

This means that the concepts to be learned must be clustered as one or more balls in the space of concepts. What we need to do in order to construct an appropriate algorithm is to keep the radius of each ball small enough, and at the same time, make the distance between the centers of the balls large enough. In the following subsections, we describe and analyze the coverage of some learning algorithms that follow this approach.

3.4.1 The Inconsistent Multi-Balls Algorithm

As a trivial case, suppose that we want to learn the set of all concepts that are within ϵ of a fixed concept c—the set $Ball(c, \epsilon)$. This is achieved simply by returning c as the hypothesis regardless of the sample. This leads to a coverage of $\sum_{i=0}^{\epsilon 2^n} \binom{2^n}{i}$.

Algorithm Two-Balls (*Sample*)

1. If *disagreement*(*Sample*, c_1) < *disagreement*(*Sample*, $\neg c_1$), return c_1.
2. Else return $\neg c_1$. Break ties arbitrarily.

end.

Figure 4. The Two-Balls algorithm. c_1 is a built-in constant concept.

Algorithm Majority-Class (*Sample*)

1. If the number of negative examples in *Sample* is more than the number of positive examples then return 0 (the *nil* concept).
2. Else return 1 (the *true* concept). Break ties arbitrarily.

end.

Figure 5. The Majority-Class algorithm.

A less trivial case is to learn 2 ϵ-balls of concepts. This is accomplished by the "Two-Balls" algorithm given in Figure 4. This algorithm returns, as the hypothesis, some fixed concept c_1 or its complement, whichever is *closer* to the training sample.

A special case of this algorithm is the "Majority-Class" algorithm (Figure 5) in which c_1 is set to be the *nil* concept, i.e. the concept which classifies all the instances as negative. It should be clear that the coverage of both the Two-Balls and Majority-Class algorithms must be equivalent.

Before going into the actual coverage analysis of these algorithms, it is interesting to look at the simple situation where we have only one example in the training sample (that is, $m = 1$). In this case, the hypothesis of Majority-Class will be the *nil* concept if the example is negative and the *true* concept if it is positive. Suppose

the target concept is any $c \in Ball(0, \epsilon)$. Then, the randomly drawn example is positive with probability at most ϵ. This means that the probability that a sample of c of size 1 is mapped to an ϵ-far hypothesis is at most ϵ. The same argument applies to all the concepts in $Ball(1, \epsilon)$. As a result, if $\epsilon \leq \delta$ then a sample of size 1 suffices to make Majority-Class learn all the $2\sum_{i=0}^{\epsilon 2^n} \binom{2^n}{i}$ concepts clustered around the concepts 0 and 1. This is, of course, true regardless of the value of n.

The following lemma computes the sample size sufficient to make the Two-Balls algorithm attain the same coverage for the cases where ϵ is not necessarily smaller than δ.

Lemma 3.5 *For accuracy and confidence parameters ϵ and δ, the coverage of the Two-Balls algorithm is*

$$2\sum_{i=0}^{\epsilon 2^n} \binom{2^n}{i}$$

when given a sample of size

$$m > \frac{12\epsilon}{(1-2\epsilon)^2} \ln\frac{1}{\delta}$$

assuming that sampling is with replacement under the uniform distribution and that $\epsilon < \frac{1}{2}$.

Proof:

Let c be a concept with distance at most $\epsilon 2^n$ from the *nil* concept. It is enough to show that the given sample size is sufficient to make the Majority-Class algorithm learn c.

Let Z denote the number of positive examples in a sample of c of size m. Since c has at most $\epsilon 2^n$ positive examples, Z can be viewed as a binomial random variable of m trials and at most ϵ as the ratio of success. A sample of c is mapped to the *true* (instead of the *nil*) concept only if it has more positive examples than negative examples. The probability of this is bounded above by

$$Pr[Z \geq \frac{m}{2}] \leq \sum_{i=\lceil \frac{m}{2} \rceil}^{m} \binom{m}{i} \epsilon^i (1-\epsilon)^{m-i} .$$

Using Chernoff's lemma (Lemma 2.2), this is at most

$$e^{-\frac{(1-2\epsilon)^2 m}{12\epsilon}} .$$

Therefore, c is learned if

$$e^{-\frac{(1-2\epsilon)^2 m}{12\epsilon}} < \delta ,$$

which is satisfied if

$$m > \frac{12\epsilon}{(1-2\epsilon)^2} \ln \frac{1}{\delta}$$

as desired. □

The condition on the sample size in the above lemma is easy to satisfy. To see this, note that when $\epsilon < \frac{1}{2}$, the term $\frac{12\epsilon}{(1-2\epsilon)^2}$ is monotonically increasing in ϵ. For the reasonable range of $0 < \epsilon \leq 0.1$, this term is at most 1.875. Therefore, we only need 1.875 $\ln \frac{1}{\delta}$ examples. Even when $\delta = 0.001$ (i.e. 99.9 % confidence), this amounts to a sample of only 13 examples. It is important to note that this is independent of the value of n. That is, these numbers hold no matter how large is the number of the features.

Since the sample size is usually much larger than this, a direct generalization of the above trivial cases is to attempt to learn as many balls of concepts as permitted by the sample size. The idea is to choose a collection of well-separated concepts (the centers of the balls) and attempt to learn all the concepts clustered around each of these centers. More specifically, we start by fixing two integers d and k, and then construct a set $\mathcal{H} = \{h_1, h_2, h_3, \cdots, h_k\}$ of k concepts such that the distance between each pair of concepts in $\mathcal{H}$ is at least d. Then given a training sample, the concept in $\mathcal{H}$ that has the minimum disagreement with the sample is returned as the hypothesis.

The goal of the algorithm is to learn all the concepts in $\bigcup_{h \in \mathcal{H}} Ball(h, \epsilon)$, which will give a coverage of $k \sum_{i=0}^{\epsilon 2^n} \binom{2^n}{i}$. The question is, of course, how to determine the appropriate values of d and k such that this goal is accomplished. Particularly, we need to worry about the following:

1. As explained earlier, d must be large enough so that the interaction between concepts in different balls is kept within what is allowed by the confidence parameter δ.

2. The number of concepts that we can construct such that the minimum distance between each pair is d drops sharply as d increases. Therefore, making d too large causes k (and hence, the coverage of the algorithm) to be too small.

In the following, we give a lemma and a corollary that are useful in choosing appropriate values for d and k.

Lemma 3.6 *For $0 < \epsilon < \frac{1}{4}$ and $2\epsilon < \alpha < \frac{1}{2}$, let $\mathcal{H} = \{h_1, h_2, \cdots, h_k\}$ be a set of concepts such that the distance between each pair $h_i, h_j \in \mathcal{H}$ is at least $d = \lceil \alpha 2^n \rceil$, and let $c \in Ball(h, \epsilon)$ for some $h \in \mathcal{H}$. Assume that L is a learning algorithm that on any sample S outputs some hypothesis $h_i \in \mathcal{H}$ that has minimal disagreement with S. Then, under the uniform distribution and when sampling is done with replacement, the probability that a sample of c of size m is mapped by L to an hypothesis other than h is at most*

$$\min_{0<\beta<1} \; k \cdot \left\{ e^{-2(1-\beta)^2\alpha^2 m} + e^{-\beta\frac{(\alpha-2\epsilon)^2}{2\alpha}m} \right\} \quad . \tag{3.5}$$

Proof:

Let $g \neq h$ be a *specific* concept in $\mathcal{H}$ and let us bound the probability that a sample of c is mapped to g (instead of h). We consider the case in which this probability is maximized. First, we assume that g is as close as possible to h—that is, $distance(h, g) = d$. Second, we place c to be as far as possible from h and as close as possible to g—that is, $distance(c, h) = \epsilon 2^n$ and $distance(c, g) = d - \epsilon 2^n$.

Let A be the set of objects where c agrees with g but not with h and let B be the set of objects where c agrees with h but not with g. Thus, $|A| = \epsilon 2^n$ and $|B| = d - \epsilon 2^n$. A sample S of c is mapped to g only if $disagreement(S, g) \leq$

disagreement(S,h). That is, only if S has more examples drawn from A than from B. Now, let us define the following three random variables:

- X: the number of examples in S drawn from A.
- Y: the number of examples in S drawn from B.
- $Z = X + Y$: the number of examples in S drawn from $A \bigcup B$.

Then, the probability that S is mapped to g is just $Pr[X \geq Y]$.

Note that Z is just a binomial random variable with m trials and $\frac{d}{2^n}$ as the ratio of success. Given that, out of the m examples in S, there are i examples drawn from $A \bigcup B$ (i.e. given that $Z = i$), the event $X \geq Y$ is equivalent to $Y \leq \frac{i}{2}$. In this case, Y can be viewed as a binomial random variable of i trials and $\frac{d-\epsilon 2^n}{d}$ as the ratio of success. Therefore, we can write

$$\begin{aligned} Pr[X \geq Y] &= \sum_{i=0}^{m} Pr[Z=i] Pr[X \geq Y | Z=i] \\ &= \sum_{i=0}^{m} Pr[Z=i] Pr[Y \leq \frac{i}{2} | Z=i] \\ &= \sum_{i=0}^{m} \left[\binom{m}{i} (\frac{d}{2^n})^i (\frac{2^n-d}{2^n})^{m-i} \sum_{j=0}^{\frac{i}{2}} \binom{i}{j} (\frac{d-\epsilon 2^n}{d})^j (\frac{\epsilon 2^n}{d})^{i-j} \right] . \end{aligned}$$

Applying Hoeffding's lemma (Lemma 2.1) on the inner summation, this is at most

$$\begin{aligned} & \sum_{i=0}^{m} \left[\binom{m}{i} (\frac{d}{2^n})^i (\frac{2^n-d}{2^n})^{m-i} e^{-2i(\frac{d-2\epsilon 2^n}{2d})^2} \right] \\ \leq & \sum_{i=0}^{\lfloor \beta \frac{d}{2^n} m \rfloor} \left[\binom{m}{i} (\frac{d}{2^n})^i (\frac{2^n-d}{2^n})^{m-i} e^{-2i(\frac{d-2\epsilon 2^n}{2d})^2} \right] + \\ & \sum_{i=\lceil \beta \frac{d}{2^n} m \rceil}^{m} \left[\binom{m}{i} (\frac{d}{2^n})^i (\frac{2^n-d}{2^n})^{m-i} e^{-2i(\frac{d-2\epsilon 2^n}{2d})^2} \right] \end{aligned}$$

for any $0 < \beta < 1$. Letting $i = 0$ in the first exponential and $i = \beta \frac{d}{2^n} m$ in the second, the above is at most

$$\sum_{i=0}^{\lfloor \beta \frac{d}{2^n} m \rfloor} \left[\binom{m}{i} (\frac{d}{2^n})^i (\frac{2^n-d}{2^n})^{m-i} \right] +$$

$$e^{-2\beta\frac{d}{2^n}(\frac{d-2\epsilon 2^n}{2d})^2 m} \sum_{i=\lceil\beta\frac{d}{2^n}m\rceil}^{m} \left[\binom{m}{i}(\frac{d}{2^n})^i(\frac{2^n-d}{2^n})^{m-i}\right]$$

$$\leq \sum_{i=0}^{\lfloor\beta\frac{d}{2^n}m\rfloor} \left[\binom{m}{i}(\frac{d}{2^n})^i(\frac{2^n-d}{2^n})^{m-i}\right] + e^{-2\beta\frac{d}{2^n}(\frac{d-2\epsilon 2^n}{2d})^2 m}.$$

Applying Hoeffding's lemma (Lemma 2.1) again on the first term, the above is at most

$$e^{-2(1-\beta)^2(\frac{d}{2^n})^2 m} + e^{-\beta\frac{(\frac{d}{2^n}-2\epsilon)^2}{2\frac{d}{2^n}}m}.$$

Since the above bound holds for any $0 < \beta < 1$, we can, of course, minimize over all β in that range and get

$$\min_{0<\beta<1} \left\{ e^{-2(1-\beta)^2(\frac{d}{2^n})^2 m} + e^{-\beta\frac{(\frac{d}{2^n}-2\epsilon)^2}{2\frac{d}{2^n}}m} \right\}.$$

This quantity is decreasing in $\frac{d}{2^n}$ when $\frac{d}{2^n} > 2\epsilon$. Thus, we can replace $\frac{d}{2^n}$ by α since $2\epsilon < \alpha \leq \frac{d}{2^n}$. Since g can be any of the k concepts in $\mathcal{H}$, multiplying the above probability by k gives the desired result. □

Note that d in Lemma 3.6 is expressed as a fraction α of 2^n. This result says that if the conditions of the lemma are met and if c is a concept in one of the k ϵ-balls, then the samples of c are usually mapped by L to the center of that ball except with a probability that is bounded by

$$k \cdot \left\{ e^{-2(1-\beta)^2\alpha^2 m} + e^{-\beta\frac{(\alpha-2\epsilon)^2}{2\alpha}m} \right\}$$

for *any* choice of β between 0 and 1. Thus, α and k must be chosen so that this quantity is at most δ. It is important to note that this probability is diminishing in α and m and independent of n.

Remember that our goal is to make k as large as possible. Note, however, that the number of concepts that we can construct such that the minimum pairwise distance is at least $\lceil\alpha 2^n\rceil$ decreases as α grows. Thus, making k larger requires making α smaller. This in turn may cause the probability of committing a mistake

to exceed δ. Thus, to find the "right" values of α and k, it is necessary to find out the relation between these two variables.

The problem of finding a collection of bit vectors that maintain a given minimum pairwise distance is well studied in the field of Error-Correcting Coding Theory. Specifically, the following theorem states how large k can (at least) be for a given value of α.

Theorem 3.4 (The Gilbert-Varshamov Bound) *There exist at least*

$$2^{l-r}$$

bit vectors of length l and minimum pairwise distance d, where r is any integer satisfying

$$\binom{l-1}{0} + \binom{l-1}{1} + \binom{l-1}{2} + \cdots + \binom{l-1}{d-2} < 2^r.$$

A proof of this theorem, in addition to a method of actually constructing the l-bit vectors as given above, can be found in many references in the literature of Error-Correcting Coding Theory (e.g., [Peterson and Weldon 72]).

For our purposes here, it is convenient to draw the following corollary from the Gilbert-Varshamov bound given above.

Corollary 3.1 *For any α, $0 < \alpha < \frac{1}{2}$, and any even positive integer l, there exist at least*

$$2^{l-\lceil H(\alpha)l \rceil}$$

bit vectors of length l and pairwise distance at least αl, where $H(\alpha) = \alpha \log_2 \frac{1}{\alpha} + (1-\alpha)\log_2 \frac{1}{1-\alpha}$.

Proof:

We just need to show that when $d = \lceil \alpha l \rceil$ and $r = \lceil H(\alpha)l \rceil$, the inequality of Theorem 3.4 is satisfied. The left-hand side of the inequality is just

$$\sum_{i=0}^{d-2} \binom{l-1}{i} < \sum_{i=0}^{\alpha l} \binom{l}{i}$$

Algorithm Multi-Balls (*Sample*, ϵ, δ)

1. Find α in the range $2\epsilon < \alpha < \frac{1}{2}$ such that

$$1 - H(\alpha) = 2[1 - \beta(\alpha)]^2 \alpha^2 \tfrac{m}{2^n} \log_2 e$$

where

$$H(\alpha) = \alpha \log_2 \tfrac{1}{\alpha} + (1-\alpha) \log_2 \tfrac{1}{1-\alpha}, \text{ and}$$
$$\beta(\alpha) = 1 + \tfrac{(\alpha-2\epsilon)^2}{8\alpha^3} - \sqrt{[1 + \tfrac{(\alpha-2\epsilon)^2}{8\alpha^3}]^2 - 1}\ .$$

2. Let $k = \left\lfloor 2^{2^n - H(\alpha)2^n} \times \frac{\delta}{2} \right\rfloor$.
3. Construct $\mathcal{H} = \{h_1, h_2, h_3, \cdots, h_k\}$ such that

$$\forall_{h_i, h_j \in \mathcal{H}}\ distance(h_i, h_j) \geq \lceil \alpha 2^n \rceil\ .$$

4. Return some hypothesis in $\mathcal{H}$ that has minimal disagreement with *Sample* (break ties arbitrarily).

end.

Figure 6. The Multi-Balls algorithm.

$$\leq 2^{H(\alpha)l} \qquad \text{(by Lemma 2.4)}$$
$$\leq 2^{\lceil H(\alpha)l \rceil}\ ,$$

which equals the right-hand side. This proves the corollary. □

Using Lemma 3.6 and Corollary 3.1, one can search for the appropriate value for α that leads to learning k different ϵ-balls, for k as large as possible. Let us now compute a lower bound on the coverage that can be achieved by this approach.

Figure 6 shows the "Multi-Balls" algorithm in which we give a specific way of choosing the value of α and k. To be able to give a lower bound on the coverage of this algorithm, we need the following definition.

Definition: For $0 \leq \theta \leq 1$ and $0 < \epsilon < \frac{1}{4}$, define $\rho(\theta, \epsilon)$ as:

$$\rho(\theta, \epsilon) = \frac{1 - H(\hat{\alpha})}{\theta}$$

for $\hat{\alpha}$ being the solution of the equation

$$1 - H(\alpha) = 2[1 - \beta(\alpha)]^2 \alpha^2 \theta \log e \tag{3.6}$$

in the range $2\epsilon < \alpha < \frac{1}{2}$, where $\beta(\alpha) = 1 + \frac{(\alpha-2\epsilon)^2}{8\alpha^3} - \sqrt{[1 + \frac{(\alpha-2\epsilon)^2}{8\alpha^3}]^2 - 1}$ and $H(\alpha) = \alpha \log_2 \frac{1}{\alpha} + (1-\alpha) \log_2 \frac{1}{1-\alpha}$. □

The following lemma ensures that the above ρ function is well-defined.

Lemma 3.7 *There exists a value for α in the range $2\epsilon < \alpha < \frac{1}{2}$ that satisfies Equation 3.6, provided that $0 < \epsilon < \frac{1}{4}$.*

Proof:

First, note that $H(\alpha)$ is monotonically increasing in the range $0 < \alpha < \frac{1}{2}$. Moreover, H is always strictly less than 1 except when $\alpha = \frac{1}{2}$, where H becomes exactly 1.

Now, call the left-hand side of Equation 3.6 $L(\alpha)$ and the right-hand side $R(\alpha)$. The reader can confirm the following:

- $L(2\epsilon) > 0$, since $2\epsilon < \frac{1}{2}$ by assumption.
- $R(2\epsilon) = 0$, since $\beta(2\epsilon) = 1$.
- $L(\frac{1}{2}) = 0$, since $H(\frac{1}{2} = 1$.
- $R(\frac{1}{2}) > 0$, since this is a product of positive quantities.

Therefore, $L(\alpha)$ and $R(\alpha)$ must intersect somewhere in the range $2\epsilon < \alpha < \frac{1}{2}$ and the lemma holds. □

Although $\rho(\theta, \epsilon)$ is not provided in closed form, it is easily computed for any given values of θ and ϵ using standard numerical methods. For illustration, Figure 7 plots this function over the range $0 \leq \theta \leq \frac{1}{2}$ for $\epsilon = 0.01, 0.05$ and 0.10.

Now, using the above definition of ρ, a lower bound on the coverage of Multi-Balls can be stated as follows.

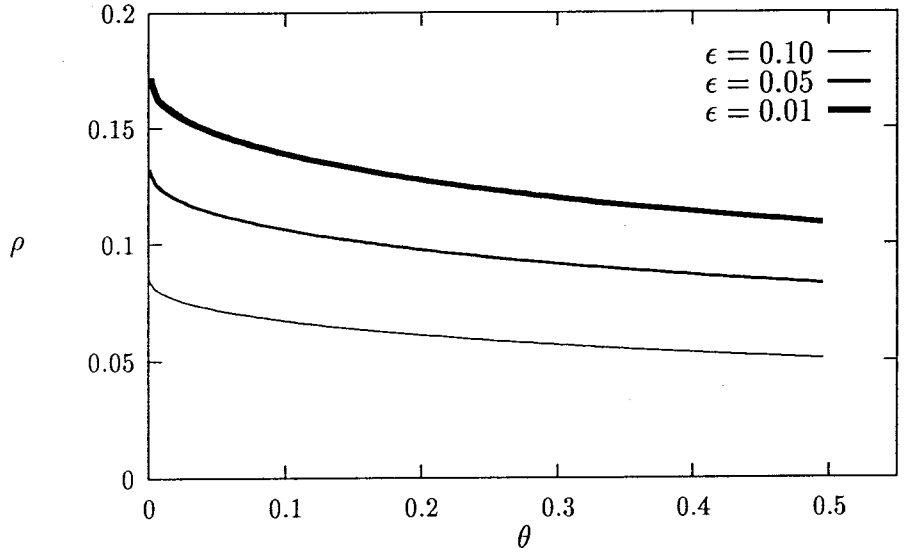


Figure 7. The function $\rho(\theta, \epsilon)$.

Theorem 3.5 *For $0 < \epsilon < \frac{1}{4}$ and $0 < \delta < 1$, the coverage of the Multi-Balls algorithm under the uniform distribution is at least*

$$\left\lfloor \frac{\delta}{2}\ 2^{\rho(\frac{m}{2^n},\epsilon)\, m} \right\rfloor \sum_{i=0}^{\epsilon 2^n} \binom{2^n}{i}$$

for sample size m, accuracy parameter ϵ and confidence parameter δ.

Proof:

First, the existance of α in the range $2\epsilon < \alpha < \frac{1}{2}$ satisfying the equation of Step 1 is guaranteed by Lemma 3.7. It is sufficient to show the following three claims for the values of α and k chosen by the algorithm in Steps 1 and 2:

Claim 1: Step 3 is executable. That is, for α and k as chosen in Steps 1 and 2, we can actually construct a set of k concepts such that the minimum pairwise distance is $\lceil \alpha 2^n \rceil$.

Claim 2: For any $h \in \mathcal{H}$ and any $c \in Ball(h, \epsilon)$, the probability that a sample of c of size m is mapped to an hypothesis other than h is at most δ.

Claim 3: The number of balls, k, is at least $\left\lfloor \frac{\delta}{2}\, 2^{\rho(\frac{m}{2^n},\epsilon)\, m} \right\rfloor$.

Claim 1 follows from Corollary 3.1 since k is set to $\left\lfloor 2^{2^n - H(\alpha)2^n} \times \frac{\delta}{2} \right\rfloor$ and thus

$$\begin{aligned} k &\leq 2^{2^n - H(\alpha)2^n} \times \frac{\delta}{2} \\ &= 2^{2^n - H(\alpha)2^n - \log_2 \frac{2}{\delta}} \\ &\leq 2^{2^n - \lceil H(\alpha)2^n \rceil + 1 - \log_2 \frac{2}{\delta}} \\ &< 2^{2^n - \lceil H(\alpha)2^n \rceil} \end{aligned}$$

since $\log_2 \frac{2}{\delta}$ is necessarily greater than 1.

For Claim 2, we use Lemma 3.6 which states that the probability of a mistake is bounded above by

$$k \cdot \left\{ e^{-2(1-\beta)^2\alpha^2 m} + e^{-\beta \frac{(\alpha-2\epsilon)^2}{2\alpha} m} \right\} \tag{3.7}$$

for *any* β in the range $[0, 1]$. Let us choose β such that

$$e^{-2(1-\beta)^2\alpha^2 m} = e^{-\beta \frac{(\alpha-2\epsilon)^2}{2\alpha} m}$$

which means that

$$2(1-\beta)^2\alpha^2 m = \beta \frac{(\alpha-2\epsilon)^2}{2\alpha} m \,. \tag{3.8}$$

Solving this quadratic equation for β gives the two solutions

$$1 + \frac{(\alpha-2\epsilon)^2}{8\alpha^3} \pm \sqrt{[1 + \frac{(\alpha-2\epsilon)^2}{8\alpha^3}]^2 - 1} \,. \tag{3.9}$$

It can be checked that Equation (3.8) has two roots, one in the range $[0, 1]$ and the other larger than 1. Thus, the smaller root in Equation (3.9), namely

$$1 + \frac{(\alpha-2\epsilon)^2}{8\alpha^3} - \sqrt{[1 + \frac{(\alpha-2\epsilon)^2}{8\alpha^3}]^2 - 1}$$

is between 0 and 1, and thus, is an appropriate value for β. Note that this is exactly the quantity $\beta(\alpha)$ in Step 1 of the algorithm.

Now, substituting for k and β in the bound of Lemma 3.6, the probability that Multi-Balls commits a mistake is at most

$$\min_{0<\beta<1} \; k \times \{ e^{-2[1-\beta]^2\alpha^2 m} + e^{-\beta \frac{(\alpha-2\epsilon)^2}{2\alpha} m} \}$$

$$\leq \frac{\delta}{2} \times 2^{2^n[1-H(\alpha)]} \times \{e^{-2[1-\beta(\alpha)]^2\alpha^2 m} + e^{-\beta(\alpha)\frac{(\alpha-2\epsilon)^2}{2\alpha}m}\}$$
$$= \frac{\delta}{2} \times 2^{2^n[1-H(\alpha)]} \times 2\, e^{-2[1-\beta(\alpha)]^2\alpha^2 m} .$$

The value of α is chosen in Step 1 such that

$$1 - H(\alpha) = 2[1-\beta(\alpha)]^2\alpha^2 \frac{m}{2^n} \log_2 e \ .$$

Therefore, continuing from above, the desired probability is at most

$$\delta \times 2^{2^n 2[1-\beta(\alpha)]^2\alpha^2 \frac{m}{2^n} \log_2 e} e^{-2[1-\beta(\alpha)]^2\alpha^2 m}$$
$$= \delta \times e^{2[1-\beta(\alpha)]^2\alpha^2 m} e^{-2[1-\beta(\alpha)]^2\alpha^2 m}$$
$$= \delta ,$$

which shows Claim 2.

Finally, Claim 3 follows immediately from the definition of ρ. Since the value of α found in Step 1 is equivalent to $\hat{\alpha}$ in the definition of ρ, we can write

$$\rho(\frac{m}{2^n}, \epsilon)\, m = \frac{1 - H(\alpha)}{\frac{m}{2^n}}\, m$$
$$= 2^n[1 - H(\alpha)] .$$

Thus, k is equivalent to $\left\lfloor \frac{\delta}{2}\ 2^{\rho(\frac{m}{2^n},\epsilon)\, m} \right\rfloor$, which completes the proof. □

More specific bounds on the coverage of Multi-Balls can be obtained from Theorem 3.5 if upper bounds on $\frac{m}{2^n}$ and ϵ are assumed. For example, if we are interested only in the range $0 < \epsilon \leq 0.05$ and $0 \leq \frac{m}{2^n} \leq 0.25$ (which are reasonable assumptions in practice), then the coverage of Multi-Balls as given by Theorem 3.5 is at least

$$\left\lfloor \frac{\delta}{2}\ 2^{0.094\, m} \right\rfloor \sum_{i=0}^{\epsilon 2^n} \binom{2^n}{i}$$

where the constant 0.094 is just the value of $\rho(0.25, 0.05)$. Note that the main difference between this and the upper bound of Theorem 3.1 is the coefficient of m in the exponent of 2. Therefore, for any fixed δ, this lower bound indicates that to achieve a given coverage, Multi-Balls requires a sample size that is within a constant factor of that required by an optimal learning algorithm.

Algorithm: Large-Ball (*Sample*)

1. Let c_1 be a constant concept.
2. Define the function s as follows:

$$s(X) = \begin{cases} 1 & \text{if } X \in \textit{Sample} \\ 0 & \text{otherwise} \end{cases}$$

3. Define the function p as follows:

$$p(X) = \begin{cases} 1 & \text{if } X \in \textit{Sample} \text{ and } X \text{ is a positive example} \\ 0 & \text{otherwise} \end{cases}$$

4. Return $h = (\neg s \wedge c_1) \vee p$.

end.

Figure 8. The Large-Ball learning algorithm.

3.4.2 The Consistent Large-Ball Algorithm

So far we have been trying to maximize the coverage by learning as many balls of concepts as possible, while fixing the radius of each ball at $\epsilon 2^n$. An alternative approach to increase the coverage is to learn ball(s) of concepts with larger radius. Because of the extremely high dimensionality of the space of concepts, any small increment in a ball's radius results in a huge increase in the number of concepts contained in the ball.

It turns out that learning a single ball of radius larger than $\epsilon 2^n$ is a surprisingly easy task. An algorithm that achieves this is the "Large-Ball" algorithm given in Figure 8. Using some fixed concept c_1, this algorithm returns an hypothesis h such that

1. For every object X that is not included in the sample, $h(X) = c_1(X)$.
2. For every object X that is included in the sample and is correctly classified by c_1, again $h(X) = c_1(X)$.

3. For every object X that is included in the sample and is misclassified by c_1, $h(X) = \neg c_1(X)$.

In other words, the algorithm works by modifying a *default* hypothesis c_1 so that the final hypothesis is fully consistent with the training sample. For example, suppose c_1 is the *nil* concept. Then this algorithm classifies all examples as negative unless they appeared as positive examples in the training sample!

In the following, we give two theorems that state the coverage of Large-Ball for the two cases of sampling with and without replacement.

Theorem 3.6 *For sample size m, accuracy parameter ϵ and confidence parameter δ, if sampling is done without replacement under the uniform distribution, then the coverage of the Large-Ball algorithm is*

$$\sum_{i=0}^{\epsilon 2^n+\beta} \binom{2^n}{i}$$

where β is the largest integer such that

$$\frac{\sum_{i=0}^{\beta-1} \binom{2^n-\epsilon 2^n-\beta}{m-i}\binom{\epsilon 2^n+\beta}{i}}{\binom{2^n}{m}} \leq \delta$$

Proof:

Without loss of generality, assume that c_1 of the algorithm is the *nil* concept. Suppose c is a concept such that $distance(c, c_1) = \epsilon 2^n + \beta$ for some integer β. That is, c has exactly $\epsilon 2^n + \beta$ distinct positive examples. The algorithm classifies all the examples as negative except those appearing in the training sample as positive examples. Therefore, any sample that has β or more positive examples is mapped by the algorithm to an hypothesis within ϵ of c. Thus, to prove the theorem, it is enough to show that the probability of getting a sample with less than β positive examples is just the right-hand side of the inequality of the theorem.

For any i, $0 \leq i \leq \epsilon 2^n + \beta$, there are $\binom{\epsilon 2^n+\beta}{i}$ ways of choosing i distinct positive examples of c, and $\binom{2^n-\epsilon 2^n-\beta}{m-i}$ to choose $m - i$ distinct negative examples of c.

Therefore, multiplying these, there are

$$\binom{\epsilon 2^n + \beta}{i}\binom{2^n - \epsilon 2^n - \beta}{m-i}$$

samples of c of size m containing exactly i positive examples. Dividing this by $\binom{2^n}{m}$, the total number of samples of size m, and summing for $i = 0$ to $\beta - 1$ gives the result. □

Now, let's try to estimate the value of β that satisfies the condition in Theorem 3.6. Suppose we fix ϵ, δ and m, and let n_1 and n_2 be two integers such that $n_1 > n_2$. Since m is fixed, the coverage when $n = n_1$ should be larger than or at least equivalent to the coverage when $n = n_2$ since the algorithm is given a larger proportion of the space of objects in the first case than in the latter case. In other words, we can say that if ϵ, δ and m are fixed, then by increasing n, the value of β that satisfies the condition decreases or at most stays the same. By letting $n \to \infty$ we get

$$\lim_{n\to\infty} \frac{\sum_{i=0}^{\beta-1}\binom{2^n-\epsilon 2^n-\beta}{m-i}\binom{\epsilon 2^n+\beta}{i}}{\binom{2^n}{m}} = \sum_{i=0}^{\beta-1}\binom{m}{i}\epsilon^i(1-\epsilon)^{m-i}$$

This follows from the well-known fact that the hypergeometric distribution (the left-hand side) turns to the binomial distribution (the right-hand side) in the limit (e.g. [Ross 88]). Using Chernoff's lemma (Lemma 2.2) we get

$$\sum_{i=0}^{\beta-1}\binom{m}{i}\epsilon^i(1-\epsilon)^{m-i} \leq e^{-\frac{(\epsilon m-\beta+1)^2}{2\epsilon m}}.$$

Setting this to be at most δ and solving for β we arrive at

$$\beta = \lfloor \epsilon m - \sqrt{2\epsilon m \ln\frac{1}{\delta}} + 1 \rfloor.$$

For fixed δ, this says that the radius of the ball of concepts learned by the Large-Ball algorithm grows linearly in ϵm.

Next, let us look at the coverage of Large-Ball when sampling is with replacement.

Theorem 3.7 *For sample size m, accuracy parameter ϵ and confidence parameter δ, if sampling is done with replacement under the uniform distribution, then the coverage of the Large-Ball algorithm is*

$$\sum_{i=0}^{\epsilon 2^n+\beta} \binom{2^n}{i}$$

where β it the largest integer such that

$$\sum_{k=0}^{\beta-1} \binom{\epsilon 2^n+\beta}{k} \left(\frac{2^n-\epsilon 2^n-\beta+k}{2^n}\right)^m \left\{1 - \sum_{i=1}^{k} \binom{k}{i} \left(\frac{2^n-\epsilon 2^n-\beta+k-i}{2^n-\epsilon 2^n-\beta+k}\right)^m (-1)^{i+1}\right\} \leq \delta.$$

Proof: Without loss of generality, assume that c_1 of the algorithm is the *nil* concept. Suppose c is a concept such that $distance(c, c_1) = \epsilon 2^n + \beta$ for some integer β. That is, c has exactly $\epsilon 2^n + \beta$ distinct positive examples. The algorithm classifies all examples as negative except those appearing in the training sample as positive examples. Therefore, any sample that has β or more positive examples is mapped by the algorithm to an hypothesis within ϵ of c. Thus, to prove the theorem, it is enough to show that the probability of getting a sample with less than β *distinct* positive examples is just the left-hand side of the inequality of the theorem. Note that, since sampling is done with replacement here, deriving this probability is more complicated than the case of Theorem 3.6, since the m examples in the sample are not necessarily distinct.

Let

- F be the set of negative examples of the concept c. Thus, $|F| = 2^n - \epsilon 2^n - \beta$.
- T be the set of positive examples of the concept of c. Thus, $|T| = \epsilon 2^n + \beta$.
- D be the (random) number of distinct positive examples in a random sample of size m of the concept c.

What we need to compute is just

$$Pr[D < \beta] = \sum_{k=0}^{\beta-1} Pr[D = k] \ .$$

Now, let $S \subset T$ be some *specific* set of k distinct positive examples of the concept c, and let R be the event of getting a sample of the concept c of size m that contains exactly S and any number of negative examples, but no positive examples not in S. Then

$$Pr[D = k] = \binom{\epsilon 2^n + \beta}{k} Pr[R] \ .$$

Now, R occurs if and only if both

- event $R1$: No positive examples from $T - S$ are in the sample, and
- event $R2$: Each positive example in S is present in the sample

are true. Therefore, $Pr[R] = Pr[R1 \cdot R2] = Pr[R1]Pr[R2|R1]$, and so finding $Pr[R1]$ and $Pr[R2|R1]$ is all what we need.

First, it is not difficult to see that

$$Pr[R1] = \left(\frac{2^n - \epsilon 2^n - \beta + k}{2^n}\right)^m \ .$$

Second, to find $Pr[R2|R1]$, let $A_i, i = 1, 2, \cdots k$, be the event that the positive example number i of S is *not* included in a sample of size m drawn from $S \cup F$. Then

$$Pr[R2|R1] = 1 - Pr\left[\bigcup_{i=1}^{k} A_i\right] \ .$$

By the principle of inclusion and exclusion (e.g. [Ross 88]), we get

$$Pr[\bigcup_{i=1}^{k} A_i] = \sum_i Pr[A_i] - \sum_{i \neq j} Pr[A_i A_j] + \cdots + (-1)^{k+1} Pr[A_1 A2 \cdots A_k] \ .$$

Now,

$$\begin{aligned} Pr[A_i] &= (\frac{2^n - \epsilon 2^n - \beta + k - 1}{2^n - \epsilon 2^n - \beta + k})^m, 1 \leq i \leq k \\ Pr[A_i A_j] &= (\frac{2^n - \epsilon 2^n - \beta + k - 2}{2^n - \epsilon 2^n - \beta + k})^m, 1 \leq i, j \leq k \text{ and } i \neq j \\ &\cdot \\ &\cdot \\ &\cdot \\ Pr[A_1 A_2 \cdots A_k] &= (\frac{2^n - \epsilon 2^n - \beta}{2^n - \epsilon 2^n - \beta + k})^m \ . \end{aligned}$$

Therefore,

$$Pr[\bigcup_{i=1}^{k} A_i] = \sum_{i=1}^{k} \binom{k}{i} (\frac{2^n - \epsilon 2^n - \beta + k - i}{2^n - \epsilon 2^n - \beta + k})^m (-1)^{i+1} .$$

Assembling, we get

$$\begin{aligned} Pr[D < \beta] &= \sum_{k=0}^{\beta-1} P(D = k) \\ &= \sum_{k=0}^{\beta-1} \binom{\epsilon 2^n + \beta}{k} Pr[R] \\ &= \sum_{k=0}^{\beta-1} \binom{\epsilon 2^n + \beta}{k} (\frac{2^n - \epsilon 2^n - \beta + k}{2^n})^m \times \\ &\qquad \left\{ 1 - \sum_{i=1}^{k} \binom{k}{i} (\frac{2^n - \epsilon 2^n - \beta + k - i}{2^n - \epsilon 2^n - \beta + k})^m (-1)^{i+1} \right\} , \end{aligned}$$

and the theorem follows. □

Let us now estimate the value of β in this result in closed form. First, let us find how many examples are sufficient to make β at least 1 (that is, the sample size that guarantees learning a ball of radius at least $\epsilon 2^n + 1$). This is easy, since by substituting 1 for β in the inequality of Theorem 3.7 we get the condition

$$\binom{\epsilon 2^n + 1}{0} \left(\frac{2^n - \epsilon 2^n - 1}{2^n}\right)^m \leq \delta .$$

That is

$$\left(\frac{2^n - \epsilon 2^n - 1}{2^n}\right)^m \leq \delta ,$$

which is certainly satisfied if

$$(1 - \epsilon)^m \leq \delta .$$

By Lemma 2.3, $(1 - \epsilon)^m \leq e^{-\epsilon m}$, and thus, the above condition is satisfied if

$$m \geq \frac{1}{\epsilon} \ln \frac{1}{\delta} ,$$

which gives the desired sample size. It is important to note that this bound is *independent* of n. Even for $\epsilon = \delta = 0.01$, this evaluates to only 461 examples, no matter how large n is!

For the general case, the following lemma gives a lower bound on what the value of β in Theorem 3.7 can be.

Lemma 3.8 *If* $\epsilon < \frac{1}{2}$, $\epsilon 2^n > 1$ *and* $m \leq \frac{1}{2\log_2 e} 2^n \approx 0.3472^n$, *then setting*

$$\beta = \left\lfloor \frac{\epsilon m \log_2 e - \log \frac{1}{\delta} - 1}{n - \log \frac{1}{\epsilon}} \right\rfloor$$

satisfies the inequality of Theorem 3.7.

Proof:

The left-hand side of the inequality of Theorem 3.7 is

$$\begin{aligned} & \textstyle\sum_{k=0}^{\beta-1} \binom{\epsilon 2^n+\beta}{k} \left(\frac{2^n-\epsilon 2^n-\beta+k}{2^n}\right)^m \left\{1 - \sum_{i=1}^{k} \binom{k}{i} \left(\frac{2^n-\epsilon 2^n-\beta+k-i}{2^n-\epsilon 2^n-\beta+k}\right)^m (-1)^{i+1}\right\} \\ \leq & \quad \textstyle\sum_{k=0}^{\beta-1} \binom{\epsilon 2^n+\beta}{k} \left(\frac{2^n-\epsilon 2^n-\beta+k}{2^n}\right)^m \end{aligned} \tag{3.10}$$

since $\{1 - \sum_{i=1}^{k} \binom{k}{i} (\frac{2^n-\epsilon 2^n-\beta+k-i}{2^n-\epsilon 2^n-\beta+k})^m (-1)^{i+1}\}$ is just the probability $Pr[R1|R2]$ as defined in the proof of Theorem 3.7, and hence, can be at most 1. Also, since k is at most $\beta - 1$,

$$\begin{aligned} \left(\frac{2^n - \epsilon 2^n - \beta + k}{2^n}\right)^m &\leq \left(\frac{2^n - \epsilon 2^n - \beta + (\beta - 1)}{2^n}\right)^m \\ &= \left(\frac{2^n - \epsilon 2^n - 1}{2^n}\right)^m \\ &< (1 - \epsilon)^m \\ &\leq e^{-\epsilon m} \qquad \text{(by Lemma 2.3)} . \end{aligned}$$

This is independent of k and thus can be moved outside the summation of Equation (3.10).

Next, we need to bound $\sum_{k=0}^{\beta-1} \binom{\epsilon 2^n+\beta}{k}$. Here, we can use Lemma 3.3 in addition to the fact that

$$\binom{\epsilon 2^n + \beta}{\beta - 1} = \frac{\beta(\epsilon 2^n + \beta)}{\epsilon 2^n (\epsilon 2^n + 1)} \binom{\epsilon 2^n + \beta - 1}{\beta} . \tag{3.11}$$

These give

$$\begin{aligned} \sum_{k=0}^{\beta-1} \binom{\epsilon 2^n + \beta}{k} &= \sum_{k=0}^{\frac{\beta-1}{\epsilon 2^n+\beta}(\epsilon 2^n+\beta)} \binom{\epsilon 2^n + \beta}{k} \\ &\leq \frac{1 - \frac{\beta-1}{\epsilon 2^n+\beta}}{1 - 2\frac{\beta-1}{\epsilon 2^n+\beta}} \binom{\epsilon 2^n + \beta}{\beta - 1} \end{aligned}$$

$$\text{(by Lemma 3.3, provided that } \beta < \epsilon 2^n + 2) \quad (3.12)$$

$$= \frac{\epsilon 2^n + 1}{\epsilon 2^n - \beta + 2}\binom{\epsilon 2^n + \beta}{\beta - 1}$$

$$= \frac{\beta(\epsilon 2^n + \beta)}{\epsilon 2^n(\epsilon 2^n - \beta + 2)}\binom{\epsilon 2^n + \beta - 1}{\beta} \quad \text{(by Equation (3.11))}$$

$$\leq \frac{\beta(\epsilon 2^n + \beta)}{\epsilon 2^n(\epsilon 2^n - \beta + 2)}(\epsilon 2^n)^\beta$$

$$\leq \frac{3}{2}(\epsilon 2^n)^\beta \qquad \text{(provided that } \beta \leq \frac{1}{2}\epsilon 2^n). \quad (3.13)$$

Therefore, the left-hand side of the inequality of Theorem 3.7 is at most

$$\frac{3}{2}(\epsilon 2^n)^\beta (1-\epsilon)^m \leq \frac{3}{2}(\epsilon 2^n)^\beta e^{-\epsilon m}$$

Setting this to be at most δ, and taking the logarithm of both sides of the inequality gives

$$\log_2 \frac{3}{2} + \beta(\log_2 \epsilon + n) - \epsilon m \log_2 e \leq \log_2 \delta \, .$$

This is certainly satisfied when

$$\beta = \left\lfloor \frac{\epsilon m \log_2 e - \log_2 \frac{3}{2\delta}}{n - \log_2 \frac{1}{\epsilon}} \right\rfloor \, .$$

Clearly, this is less than $\epsilon m \log_2 e$. Given the assumption that $m < \frac{1}{2\log_2 e}2^n$, the above value of β is at most $\frac{1}{2}\epsilon 2^n$, which satisfies the conditions of Equations (3.12) and (3.13). This completes the proof. □

By now, we have shown that the radius of the ball of concepts learned by the Large-Ball algorithm grows linearly in m, for both cases of sampling with or without replacement. Note that a unit increment in the radius of the ball corresponds to a very large increment in coverage. The coverage of Large-Ball, therefore, grows quite rapidly as the sample size increases.

The coverage lower bound obtained for Large-Ball appears to overlap the bound for Multi-Balls, although Multi-Balls gives higher coverage for small values of ϵ (e.g. 0.01). In any case, the fact that a trivial algorithm like Large-Ball achieves such high coverage is rather disturbing, since this algorithm does not seem to be of any practical value. Before considering this point further, we will measure, in the subsequent section, the coverage of some popular learning algorithms.

3.4.3 Unifying the Balls Algorithms

The balls approach for designing learning algorithms with high coverage involves three parameters: the number of balls of concepts to be learned, the radius of each ball and the separation distance between the centers of these balls. The Multi-Balls algorithm works by fixing the radius at $\epsilon 2^n$ and choosing the separation distance that results in learning as many balls of concepts as possible. The Large-Ball algorithm, on the other hand, attempts to learn only one ball, but with radius as large as possible.

Although both algorithms has been shown to achieve high coverage, neither of them necessarily gives the maximum coverage obtainable by following the balls approach. Indeed, the particular choices hard-wired in these two algorithms (e.g. fixing the radius in Multi-Balls and fixing the number of balls in Large-Ball) are there merely to ease the coverage analysis and to allow deriving lower bounds on what coverage can actually be achieved.

To obtain the best coverage that can be achieved by the the balls approach, one has to compute the coverage for all *legitimate* combinations of the radius, separation distance, and number of the balls. Of course, this should result in coverage that must be at least as large as the coverage of each of Multi-Balls and Large-Ball, since we are taking the maximum over these. Unfortunately, the analysis of this approach gets too complicated, so it is not pursued further in this thesis.

3.5 Measuring the Coverage of Current Learning Algorithms

So far, our objective has been to design learning algorithms with the maximum possible coverage. In this section, we look at the problem of evaluating the coverage of practical learning algorithms that are currently in use.

The most straightforward method to perform such evaluations (as done in [Dietterich 89]) is to exhaustively test an algorithm with respect to each concept by running it on every possible training sample of that concept. However, this requires prohibitively expensive computations. We are faced with two sources of combinatorial explosion:

1. **The number of samples per concept:** To determine with certainty whether a learning algorithm L learns a given concept c, one must measure the percentage of samples on which L returns an ϵ-close hypothesis and check whether it actually reaches $1 - \delta$. This requires running the algorithm on all the possible samples of c. With n features, there are as many as $\binom{2^n}{m}$ different samples of m distinct examples. This says that for $n = 5$ and $m = 16$, checking whether a learning algorithm learns a certain concept requires running the algorithm on $\binom{32}{16}$, i.e. more than 600 million different training samples of that concept!

2. **The number of concepts to be tested:** There are as many as 2^{2^n} different concepts defined over n features. For $n = 5$ this means that we have to test the performance of the algorithm on as many as $2^{2^5} > 4$ billion concepts.

The large number of runs required by the exhaustive approach limited Dietterich's experiments to the space of concepts defined over only 3 features. The goal of this part of the thesis is to extend these experiments to handle a larger number of features. This is achieved by introducing two techniques that help in reducing the required running time of the experiments. Although these techniques do not entirely eliminate the computational cost problem, they do result in a significant cut in these costs. Specifically, applying these techniques enables carrying out coverage evaluation experiments on the space of concepts defined on up to 5 Boolean features. The results of these experiments are reported at the end of this section.

3.5.1 Statistical Approximation of the Learnability of a Concept

One way to avoid the high computational costs involved in determining whether a learning algorithm learns a given concept is to resort to statistical approximation. Given a concept c and a learning algorithm L, we can decide with high level of significance whether L learns c with respect to specific values of ϵ, δ and m by applying the following procedure:

1. For some large integer k, generate k random training samples of c of size m.
2. Run L on each of these k training samples
3. Let $\tilde{\delta}_c$ be the percentage of the k training samples on which L returned an hypothesis that is ϵ-far from c.
4. For some small fraction θ,
   - If $\tilde{\delta}_c < \delta - \theta$, conclude that L learns c.
   - If $\tilde{\delta}_c > \delta + \theta$, conclude that L does not learn c.
   - Otherwise, if $\delta - \theta \leq \tilde{\delta}_c \leq \delta + \theta$, then *hesitate* to give either conclusion.

The coverage of L is then given as a range $a \pm b$, where $a - b$ is the number of concepts which resulted in the first outcome, and $a + b$ is the sum of this number plus the number of those which resulted in the third outcome. Note that we can always reduce the occurrence of the third outcome—and hence, the gap b in the coverage estimate— by choosing a smaller value for θ.

Statistically, the higher the values of k and θ, the more significant is the outcome of the above procedure. The key point here is the fact that a satisfactorily high level of significance (e.g. 99%) can be assured by testing only an affordable number of training samples.

To see this, let δ_c denote the exact proportion of the samples of size m on which L returns an hypothesis that is ϵ-far from c in the above procedure. Also,

let $Y_1, Y_2, \cdots\cdots, Y_k$ be k random variables such that Y_i is 1 if L returns an ϵ-close hypothesis on the i-th training sample and 0 otherwise. Clearly, each of these random variables is 0 with probability δ_c and 1 with probability $1 - \delta_c$. The sum of these variables, $Y = \sum_{i=1}^{k} Y_i$, therefore represents a binomial random variable on k trials with δ_c as the ratio of success. Clearly, Y is equal to $(1 - \tilde{\delta}_c)k$.

In the above procedure, there are two ways of making a wrong decision—considering a learned concept unlearned and vice versa. To measure the significance of our decision, we need to compute the probability of making such wrong decisions.

Now, suppose that c is learned by the algorithm. This, by definition, implies that $\delta_c \leq \delta$. The probability of the (wrong) conclusion that c is not learned is

$$\begin{aligned} Pr[\tilde{\delta}_c > \delta + \theta] &= Pr[Y < (1 - \delta - \theta)k] \\ &= Pr\left[\frac{Y - (1 - \delta_c)k}{\sqrt{k\delta_c(1 - \delta_c)}} < \frac{(1 - \delta - \theta)k - (1 - \delta_c)k}{\sqrt{k\delta_c(1 - \delta_c)}}\right]. \end{aligned}$$

Since $\delta_c \leq \delta$, and assuming that $\delta \leq \frac{1}{2}$, it can be confirmed that

$$\frac{(1 - \delta - \theta)k - (1 - \delta_c)k}{\sqrt{k\delta_c(1 - \delta_c)}} \leq \frac{-\theta k}{\sqrt{k\delta(1 - \delta)}}.$$

Thus, the above probability is bounded above by

$$Pr\left[\frac{Y - (1 - \delta_c)k}{\sqrt{k\delta_c(1 - \delta_c)}} < \frac{-\theta k}{\sqrt{k\delta(1 - \delta)}}\right].$$

Since Y has mean $(1 - \delta_c)k$ and variance $k\delta_c(1 - \delta_c)$, by applying the central limit theorem (e.g. [Rice 88]), the distribution of $\frac{Y-(1-\delta_c)k}{\sqrt{k\delta_c(1-\delta_c)}}$ is approximated by the standard normal distribution. That is,

$$Pr\left[\frac{Y - (1 - \delta_c)k}{\sqrt{k\delta_c(1 - \delta_c)}} < \frac{-\theta k}{\sqrt{k\delta(1 - \delta)}}\right] \approx Pr\left[\Phi < \frac{-\theta k}{\sqrt{k\delta(1 - \delta)}}\right],$$

where Φ denotes the standard normal random variable. $Pr\left[\Phi < \frac{-\theta k}{\sqrt{k\delta(1-\delta)}}\right]$ can be evaluated for any k and θ from the tables of the standard normal distribution. Using this, we can estimate the probability of the wrong decision of concluding a learned concept as unlearned.

Following a symmetric argument, the probability that we consider an unlearned concept as learned can be shown to be the same as above.

As a numerical example, suppose that $\delta = 0.1$, then to achieve 99% level of significance (that is, to make the probability of a wrong decision within 0.01), it suffices that we test $k = 10,000$ samples if we let $\theta = 0.007$.

3.5.2 Exploiting Symmetry Properties

In the previous subsection, we have shown how a feasible test can determine with high level of significance whether a given concept is learned by an algorithm. Now we deal with the other source of high computational costs which is due to the huge number of concepts (2^{2^n}) to be considered. As explained earlier, measuring the coverage by directly testing the algorithm on every possible concept can be done for $n = 3$ (as in [Dietterich 89]) or perhaps $n = 4$ but certainly not for $n \geq 5$.

It turns out, however, that testing a learning algorithm on all the concepts in the space is highly redundant and can be avoided by exploiting the fact that most of the learning algorithms are insensitive to the permutation and/or negation of the features of the domain. Because of these symmetry properties, these algorithms give identical performance in learning concepts that are isomorphic under the permutation and negation operations on the set of features (Section 2.4). That is, for given values of ϵ, δ and m, if we know that an algorithm learns a concept represented by a Boolean function say $f(x_1, x_2, \cdots, x_i, \cdots, x_j, \cdots, x_n)$, then this implies that the same algorithm also learns the concepts represented by $f(x_1, x_2, \cdots, x_j, \cdots, x_i, \cdots, x_n)$, $f(x_1, x_2, \cdots, \neg x_i, \cdots, x_j, \cdots, x_n)$ and so on for all the functions obtained by permuting and/or negating the features in f.

Many algorithms are also symmetric with respect to complementing the concept itself. That is, a concept represented by $f(x_1, x_2, \cdots\cdots, x_n)$ is learned if and only if the concept represented by $\neg f(x_1, x_2, \cdots\cdots, x_n)$ is learned as well.

As explained in Section 2.4, the permuting, negating and complementing operations on the space of Boolean concepts partition the space into equivalence classes

such that every two concepts in the same equivalence class can be generated from each other by applying the appropriate operations. The performance of a symmetric learning algorithm is, therefore, identical for any two concepts in the same equivalence class. Thus, to measure the coverage of the algorithm, it suffices to test it against only one concept (a *representative*) from each equivalence class. The coverage is then obtained by summing up the sizes of the equivalence classes whose representative concept was found to be learned by the algorithm.

3.5.3 Usefulness and Limitations of the Reduction Techniques

How effective are the two techniques given above in reducing the computational costs involved in measuring the coverage of learning algorithms?

The statistical approximation approach proposed to test the learnability of a concept by a given learning algorithm is valid regardless of the number of features. This is not true, however, with reducing the number of concepts to be tested by exploiting the symmetry properties of learning algorithms. For n features, there are $n!$ permutations and 2^n patterns of negation. Therefore, instead of 2^{2^n} concepts, we end up testing roughly $\frac{2^{2^n}}{n!2^n}$ concepts for algorithms that are symmetric with respect to permutation and negation of the features, and roughly $\frac{2^{2^n}}{2n!2^n}$ for those which are also symmetric with respect to complementing the concept. Table 1 gives specific numbers for n up to 6 [Harrison 65]. The third column in the table (PN) shows the number of equivalence classes to be considered for algorithms that are symmetric with respect to features permutation and negation only. The fourth column (PNC) gives the number for algorithms that are symmetric with respect to complementing the concept as well.

These numbers show that the techniques we give enable us to run coverage measurement experiments for n up to 5. This is a considerable improvement compared to Dietterich's experiments in which n is only 3. Unfortunately, however, it is clear that performing experiments for $n \geq 6$ is far from feasible, unless further

Table 1. The number of equivalence classes under permuting and negating the features of the concepts and complementing the concepts.

n	2^{2^n}	# of equivalence classes PN	PNC
3	256	22	14
4	65,536	402	238
5	$> 4.2 \times 10^9$	1,228,158	698,635
6	$> 1.8 \times 10^{19}$	$> 4 \times 10^{14}$	$> 2 \times 10^{14}$

cost reduction techniques are discovered and incorporated.

3.6 Experimental Results on Five Boolean Features

In this section, we describe the coverage evaluation experiments we carried out on the space of concepts defined over five Boolean features by directly applying the reduction techniques of the previous section.

3.6.1 Experiments Description

We considered four learning algorithms:

1. **ID3** of Quinlan [Quinlan 86] which is described in Section 2.5.

2. **FRINGE** [Pagallo and Haussler 90] as described in Section 2.6.

3. **MDT(Minimum Decision Tree):** This algorithm searches exhaustively for a decision tree that is consistent with the sample and has a minimum number of nodes (ties broken randomly).

4. **RSC(Random Selection Criterion):** Same control structure as ID3 but the feature to be tested is chosen arbitrarily in each recursive call.

The last two algorithms are tested to check how the feature selection criterion implemented in ID3 and FRINGE is compared to the optimal (MDT) and arbitrary (RSC) behavior. Particularly, RSC was reported to achieve a level of performance surprisingly comparable to that of ID3 in some real-world applications [Mingers 89].

The learning parameters were as follows: $n = 5$, $m = 8, 10, 12, 14$, and 16, $\epsilon = \frac{3}{32}$ and $\delta = 0.10$. Sampling was done without replacement under the uniform distribution.

FRINGE is symmetric with respect to permuting and/or negating the features. The other three algorithms are, in addition, symmetric with respect to complementing the target concept. Therefore, the number of representative concepts tested was 1,228,158 for FRINGE, and only 698,635 for the other algorithms. (See Table 1.) The representative concepts were found using the procedure Find-Representatives given in Section 2.4.

Determining whether or not a concept is learned by an algorithm in the above setting was done in two passes:

- In the first pass, only 100 randomly-chosen samples per concept were tested. Concepts for which the number of samples that resulted in ϵ-close hypothesis is less than 60 out of 100, were considered not learned and thus excluded.

- In the second pass, the remaining concepts were tested more extensively as explained in Subsection 3.5.1, setting $k = 10,000$ and $\theta = 0.007$.

3.6.2 Coverage of the Balls Approach

For the sake of comparison, let us see what coverage can be attained by following the balls approach explained in Section 3.4. Let us first look at the coverage of the Large-Ball algorithm (Figure 8). Suppose that we let the default concept of the algorithm be the *nil* concept. Then, for any sample size, this algorithm trivially learns all the concepts in $Ball(0, \frac{3}{32})$. When the sample size is $m \geq 14$, the radius

of the ball grows from 3 to 4. This is true since letting $\beta = 1$ and $m = 14$ in Theorem 3.6 gives

$$\frac{\sum_{i=0}^{\beta-1}\binom{2^n-\epsilon 2^n-\beta}{m-i}\binom{\epsilon 2^n+\beta}{i}}{\binom{2^n}{m}} = \frac{\binom{32-3-1}{14}\binom{3+1}{0}}{\binom{32}{14}} = 0.085 < \delta = 0.1 \ .$$

Thus, the coverage of the Large-Ball algorithm is

$$\binom{32}{0} + \binom{32}{1} + \binom{32}{2} + \binom{32}{3} = 5,489$$

for m up to 12, and

$$\binom{32}{0} + \binom{32}{1} + \binom{32}{2} + \binom{32}{3} + \binom{32}{4} = 41,449$$

for $m \geq 14$.

One can, however, do better than that for this particular case. Consider the concepts in the balls $Ball(0, \frac{3}{32})$ and $Ball(1, \frac{3}{32})$. The distance, d, between any two concepts $c_1 \in Ball(0, \frac{3}{32})$ and $c_2 \in Ball(1, \frac{3}{32})$ is at least 26. Therefore, the number of samples of size m that are shared by c_1 and c_2 is at most

$$\binom{2^n - d}{m} = \binom{6}{m} .$$

Therefore, if $m > 6$, then there exist no samples shared by c_1 and c_2.

Similarly, the distance between any two concepts $c_1 \in Ball(0, \frac{4}{32})$ and $c_2 \in Ball(1, \frac{4}{32})$ is at least 24, which means that there are exist no samples of size > 8 shared by c_1 and c_2.

Due to the absence of any interaction between the concepts in $Ball(0, \frac{3}{32})$ and $Ball(1, \frac{3}{32})$ when $m > 6$, and between $Ball(0, \frac{4}{32})$ and $Ball(1, \frac{4}{32})$ when $m > 8$, the Large-Ball algorithm can be easily modified to learn "two large balls". This is achieved by the "Two-Large-Balls" algorithm given in Figure 9. Note that this algorithm is just the consistent version of the Majority-Class algorithm (Figure 5). This algorithm classifies all the examples that are not in the training sample as positive if the majority of the examples in the sample are positive, or negative

Algorithm: Two-Large-Balls(*Sample*)

1. If *disagreement*(*Sample*,0) $<$ *disagreement*(*Sample*,1), then $c_1 = 0$, else, if *disagreement*(*Sample*,0) $>$ *disagreement*(*Sample*,1), then $c_1 = 1$. Otherwise, arbitrarily set c_1 to 0 or 1.
2. Define the function s as follows:

$$s(X) = \begin{cases} 1 & \text{if } X \in \textit{Sample} \\ 0 & \text{otherwise} \end{cases}$$

3. Define the function p as follows:

$$p(X) = \begin{cases} 1 & \text{if } X \in \textit{Sample} \text{ and } X \text{ is a positive example} \\ 0 & \text{otherwise} \end{cases}$$

4. Return $h = (\neg s \wedge c_1) \vee p$.

end.

Figure 9. The Consistent Two-Large-Balls learning algorithm.

Table 2. Coverage figures for the tested algorithms.

	Sample size				
Algorithm	8	10	12	14	16
ID3	12±0	332±0	396±0	1,756±0	4,954±640
FRINGE	12±0	332±0	396±0	1,756±0	5,284±970
MDT	12±0	12±0	116±40	496±0	3,694±0
RSC	66±0	66±0	226±0	226±0	1,698±0
2-Balls	10,978	10,978	10,978	82,898	82,898
Upper Bound	661,333	2,041,173	6,148,551	17,985,991	50,753,991

otherwise (breaking ties arbitrarily). For those examples included in the training sample, the algorithm gives the same class as given in the sample.

The coverage of Two-Large-Balls is twice the coverage of Large-Ball. That is, it learns 10,978 concepts for $m = 8, 10, 12$ and 82,898 concepts for $m = 14$ and 16.

3.6.3 Results

Table 2 summarizes the coverage figures obtained by each of the algorithms we have tested. The last row in the table gives the maximum coverage that can not be exceeded by any algorithm, using the upper bound result given by Theorem 3.1. The "2-Balls" coverage indicates the the coverage of the Two-Large-Balls algorithm as given in the previous subsection.

There are three important points to note in the above results:

- MDT does not give better coverage than the heuristic algorithms ID3 and FRINGE,
- the coverage of ID3 and FRINGE is disappointingly smaller than what is obtained by the balls approach, and

- the coverage of all these algorithms is far below the upper bound of Theorem 3.1.

3.7 Summary

The goal of this chapter was to determine what can be achieved if we take the number of training examples m to be fixed and seek learning algorithms that maximize the number of concepts learned with those m examples (i.e., maximize the "coverage"). Various results on using coverage as the evaluation criterion of inductive learning algorithms were introduced. Specifically, we have shown that when sampling is done with replacement under the uniform distribution, the Multi-Balls algorithm which follows the above strategy has a coverage of at least

$$\lfloor 2^{\text{constant} \times m - \log_2 \frac{2}{\delta}} \rfloor \sum_{i=0}^{\epsilon 2^n} \binom{2^n}{i}$$

for sample size m, accuracy parameter ϵ and confidence parameter δ (assuming practically reasonable ranges for ϵ and the ratio $\frac{m}{2^n}$). It is also shown that

$$2^{(1-1.44\epsilon)m+1+\log_2 \frac{1}{1-\delta}} \sum_{i=0}^{\epsilon 2^n} \binom{2^n}{i}$$

is an upper bound on the coverage of any learning algorithm. For any fixed δ, these results show that to achieve a given coverage, Multi-Balls requires a sample size that is within a constant factor of that required by an algorithm with optimal coverage. It should be noted that this is analogous to other results on the sample complexity of learning algorithms where the gap between upper and lower bounds is at best a constant factor [Ehrenfeucht *et al.* 88].

We have also shown that there exist fairly trivial algorithms, such as the Large-Ball algorithm, that can also achieve very large coverage—larger than Multi-Balls in cases where ϵ is reasonably big. Experimental tests confirm that Large-Ball and Two-Large-Balls, a variation of Large-Ball, have much better coverage than the popular ID3 algorithm and its relatives.

3.8 Discussion

The results introduced in this chapter are very thought-provoking, because, upon careful analysis, it becomes clear that the Large-Ball algorithm is rather trivial and uninteresting. Why does Large-Ball strike us as trivial? Because it merely memorizes the training sample—it does not attempt to find any regularity in the data. For example, if we let the default concept of the algorithm be the *nil* concept, this algorithm will simply guess the negative class for every example unless the example appeared as positive in the training sample. Therefore, the Large-Ball algorithm is unlikely to be appropriate in real-world learning situations. Our results, however, show that the coverage of this algorithm is substantially higher than practical algorithms such as ID3. Hence, coverage maximization alone does not appear to yield practically-useful learning algorithms.

The problem with the Large-Ball algorithm is that the concepts it learns, while they are very numerous, are all located around a single concept. This suggests that, to evaluate the usefulness of learning algorithms, one should not merely count the *number* of the learned concepts but should also look at *which* concepts are being learned. This requires that we adopt some basis to judge the *importance* of each concept in the space and that the computation of coverage be biased accordingly. Of course, with this *biased coverage*, optimality is defined only with respect to the assumed basis for assessing the *importance* of concepts.

Discriminating between different concepts should, of course, be based on our prior knowledge about the characteristics of the learning domain. For example, if it is suspected that there are many features in the domain that are irrelevant to the target concept, then a good criterion is to define the significance of a concept based on the minimum number of features necessary to describe that concept. Likewise, one may have reasons to suspect that the target concept in a given domain has a short description in a certain language for representing the space of concepts (e.g. decision trees). In this case, concepts with short descriptions should carry more

weight.

Now, define $Learns(L, c, m, \epsilon, \delta)$ to be 1 if algorithm L can learn concept c from a sample of size m for accuracy and confidence parameters ϵ and δ, and 0 otherwise. One possible way to define the *biased coverage* of a learning algorithm L is as follows. We define ω as a positive real-valued function defined on the space of all concepts such that $\omega(c)$ indicates the weight or importance of a concepts c. Then, the biased coverage of L can be defined as

$$\sum_{c \in 2^{U_n}} \omega(c) \times Learns(L, c, m, \epsilon, \delta) .$$

Alternatively, we can define a relation ">" such that $c_1 > c_2$ if and only if c_1 is carries more importance than c_2. Then the biased coverage of a learning algorithm L is defined as the size of the set

$$\{c \mid Learns(L, c, m, \epsilon, \delta) = 1 \text{ and } \forall_{c' \in 2^{U_n}} [c' > c \Rightarrow Learns(L, c', m, \epsilon, \delta) = 1]\}$$

That is, a concept is "covered" only if *all* of the more important concepts are also covered.

In conclusion, the results of this chapter suggest that an important problem for future research is to design and analyze learning algorithms that maximize the *biased coverage* for many of the popular biases. In the next chapters, the above ideas will be applied in the evaluation of algorithms intended for learning situations where, among the many available features of the domain, only a few features are assumed to be relevant to the target concept.

Chapter 4

Learning With Many Irrelevant Features: Exact Methods

4.1 Motivation

In this chapter, we investigate the problem of learning in situations where the target concept depends only on a small proportion of the features tested in the training data. Such situations are quite common in practice. For example, in many 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 in the data is when the same body of training examples is being used to learn many different concepts. In such cases, one must ensure that the set of features measured in the data is sufficient to learn all of the target concepts. However, when learning each individual concept, 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 [Littlestone 88]) 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.

To deal with these situations, one should seek learning algorithms that maintain a good level of immunity against the presence of irrelevant features. In other words, for a learning algorithm to be appropriate for application domains of the above type, the performance of the algorithm (e.g., the required number of training examples) should be minimally affected by the number of irrelevant features included in the training data.

The goal of this chapter is to explore the task of learning in situations of the above nature. We start by formalizing the problem by letting the complexity of a concept depends on the number of features involved in the definition of that concept. We then define and study the MIN-FEATURES bias in which consistent hypotheses definable over as few features as possible are preferred. We discuss sample and time complexities of learning algorithms that implement the MIN-FEATURES bias, and introduce the FOCUS-1 and FOCUS-2 algorithms which are actual implementations of this bias.

4.2 Definitions and Terminology

The assumption that many of the features measured in the training data are irrelevant to the target concept translates to the belief that the target concept is more likely to be one of those concepts defined over a fewer number of features than those depending on a larger number of features.

Given that a learning algorithm can learn only a limited number of concepts as discussed in Chapter 3, this says that a good learning algorithm in this context should concentrate its attention on learning the more important concepts—those defined only on a a small subset of the available features. In other words, we desire that the number of features involved in the definition of a concept be what

determines the difficulty of learning that concept, or equivalently, the *complexity* of the concept. Given a limited amount of data, the algorithm should attempt to learn the concepts that involve a small number of features. With a larger amount of data, the algorithm should be able to learn concepts that involve a larger number of features, and so on.

Let's now formalize this. For an object $X \in U_n$, let $X^{(i)}$, for $1 \leq i \leq n$, denote the value of the feature x_i in the object X. A feature x_i is said to be *relevant* to a concept c if and only if there exist two objects X_1 and $X_2 \in U_n$ such that

1. $c(X_1) \neq c(X_2)$, and
2. $X_1^{(i)} \neq X_2^{(i)}$ and $\forall_{j \neq i} X_1^{(j)} = X_2^{(j)}$.

In other words, we require that X_1 and X_2 agree on the values of all the features except x_i and that X_1 and X_2 be classified differently by c.[5]

Now, consider the following method for encoding the space of Boolean concepts. A concept c is represented as the concatenation of two bit strings R_c and T_c where

- R_c is an n-bit string in which the i-th bit is 1 if and only if x_i is relevant to c.
- T_c is the (transposed) 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.

For example, for $n = 5$, let c be the concept represented by $x_1 \vee x_3$. Then, $R_c = 10100$, $T_c = 0111$, and the concept c is represented by the string 101000111.

We define the *complexity* of a concept c as the number of bits needed to represent c under the above encoding scheme. Therefore, if c has p relevant features out of n available features, then the complexity of c is given by $\textit{complexity}(c) = 2^p + n$. For

[5]Note that this condition is satisfied if and only if x_i appears in every Boolean formula that represents c. This, therefore, gives an alternative definition for a feature to be relevant to a concept.

the example given above, we have $complexity(c) = 9$. In the rest of this chapter we will use s exclusively to denote the complexity of the target concept.

An important property of the above complexity measure is that for any two concepts c_1 and c_2, $complexity(c_1) < complexity(c_2)$ if and only if the number of relevant features in c_1 is less than the number of relevant features in c_2.

As explained in Chapter 2, the performance of a learning algorithm is generally based on the number of examples (sample complexity) and the amount of computation (computational complexity) "consumed" by the algorithm to achieve some learning task. For our purposes here, we can state the learning task as

> Learning an arbitrary Boolean concept c of complexity s defined over n available features, for given accuracy and confidence parameters ϵ and δ.

Accordingly, evaluating the performance of a learning algorithm L means answering the following two questions:

- **Sample complexity:** What is $M(n, s, \epsilon, \delta)$ such that L accomplishes the learning task whenever the number of training examples m is greater than or equal to $M(n, s, \epsilon, \delta)$?

- **Computational complexity:** What is $T(n, m, s, \epsilon, \delta)$ such that L terminates in time at most $T(n, m, s, \epsilon, \delta)$ when given a sample of size m?

In other words, we are interested in computing how many training examples and how many computational steps are required for the algorithm L to learn an arbitrary Boolean concept of complexity s.

The goal is, of course, to design learning algorithms whose sample and computational complexities are as efficient as possible. Particularly, we are interested in the question of whether there exists a learning algorithm whose sample and computational complexities are bounded by some polynomial in $n, s, \frac{1}{\epsilon}$ and $\frac{1}{\delta}$.

4.3 The MIN-FEATURES Bias

Given the assumption that many of the features measured in the training examples are irrelevant to the target concept, a reasonable bias is to prefer consistent hypotheses definable over as few features as possible. We call this the *MIN-FEATURES bias*. More specifically, this bias can be stated as follows: Given a training sample S of some unknown target concept c, let V be the set of all Boolean functions consistent with S,[6] and H the subset of V whose elements have the fewest relevant features. Then, the MIN-FEATURES bias chooses its hypothesis from H.

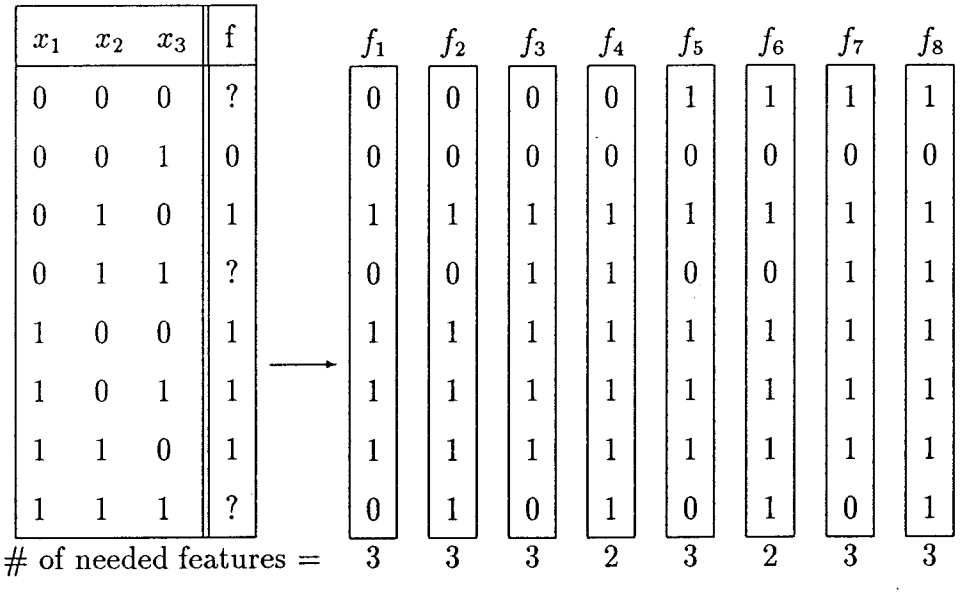
Example: Consider the following training sample:

$$\{\langle 001, 0\rangle, \langle 010, 1\rangle, \langle 100, 1\rangle, \langle 101, 1\rangle, \langle 110, 1\rangle\}$$

As shown in Figure 10, f_1 through f_8 are 8 different hypotheses that are consistent with the above training sample. It can be checked that f_4 and f_6 can be expressed using only two features, whereas the rest of the hypotheses are defined on 3 features. Hence, applying the MIN-FEATURES bias means returning either f_4 or f_6 as the final hypothesis. □

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. In the rest of this chapter, we will investigate the problem of learning in the presence of many irrelevant features mainly by analyzing algorithms that implement the MIN-FEATURES bias.

[6] V is usually called the *version space* [Mitchell 80].



$$f_4 = x_1 \vee x_2$$
$$f_6 = x_1 \vee \neg x_3$$

Figure 10. An example of the MIN-FEATURES problem.

4.4 Sample Complexity Analysis

The goal of this section is to analyze the sample complexity; that is, the number of examples needed for learning. First, we give an upper bound on this quantity for algorithms that implement the MIN-FEATURES bias, and then show that our bound is asymptotically tight by proving an identical lower bound on the sample complexity of *any* learning algorithm, whether or not it implements the MIN-FEATURES bias.

4.4.1 Upper Bound

To derive an upper bound on the sample complexity of learning algorithms that implement the MIN-FEATURES bias, we make use of a simple (yet powerful) lemma given by Blumer et al. [Blumer *et al.* 87b]. This result provides a useful tool to obtain the sample complexity of any consistent learning algorithm in terms of the size of the hypothesis space considered by the algorithm.

Briefly, the idea behind this lemma can be outlined as follows. Suppose that L is an algorithm whose hypothesis never disagrees with the examples in the training sample (i.e., a consistent algorithm) and that L always chooses its hypothesis from a set H of cardinality $|H|$. That is, H is the hypothesis space explored by L. Upon examining any training example, L rules out any element in H that incorrectly classifies this example. Let V_S be the subset of H containing the elements that were not ruled out after the set S of training examples has been examined, i.e. the elements of H for which no inconsistency with the the sample S has been detected. As S grows, the set V_S shrinks progressively. Naturally, the larger the difference between an element in H and the target concept, the more it is likely that the algorithm will encounter an example on which this element and the target concept disagree, and hence, the greater the chance that this element be ruled out. Therefore, if S is large enough, then it would be highly unlikely that V_S contains an hypothesis that differs too much from the target concept. As

a consequence, choosing any element from V_S leads to a probably approximately correct hypothesis.[7]

Of course, the larger the set H explored by the algorithm, the more training examples would be needed to reach this stage. The lemma of Blumer *et al.* turns this relation into a sufficient sample size expressed in terms of the size of the hypothesis space of the learning algorithm.

Lemma 4.1 (Blumer, Ehrenfeucht, Haussler and Warmuth) *Let C be a set of concepts and let L be a consistent learning algorithm whose hypothesis space is C. Then, for any probability distribution D over the space of objects, any ϵ and δ such that $0 < \epsilon, \delta < 1$ and any $c \in C$, a sample of size*

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

is sufficient to guarantee that L learns c with respect to ϵ, δ and D.

Proof:

Suppose $c \in C$ is the target concept, and let g be any concept in C that is ϵ-far from c. Then the probability that a randomly drawn example of c is consistent with g is at most $(1-\epsilon)$. Therefore, a sample of m independently drawn examples of c can be consistent with g with probability at most $(1-\epsilon)^m$. Since g can be any concept in C, the probability that *some* concept in C is consistent with a sample of c of size m is at most $|C|(1-\epsilon)^m$ which is in turn at most $|C|e^{-\epsilon m}$. Setting this to be at most δ and solving for m gives

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

as desired. □

Lemma 4.1 is useful in deriving the sample complexity in cases where the size of the hypothesis space explored by the learning algorithm can be analytically computed. This is the case for algorithms that implement the MIN-FEATURES

[7] In [Haussler 88], the term *exhausting* the hypothesis space is used to describe this process.

bias. The following theorem gives an upper bound on the sample size required by any algorithm that implements this bias.

Theorem 4.1 *Let $C_{n,s}$ be the set of concepts defined on n features with complexity at most s. Then, under any probability distribution D over the space of objects, 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 size*

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

is sufficient to guarantee that <u>any</u> algorithm implementing the MIN-FEATURES bias learns c with respect to ϵ, δ and D.

Proof:

For any target concept of complexity at most s, the hypothesis of any algorithm that implements the MIN-FEATURES bias must be 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}. Also, there are $\binom{n}{p}$ ways to choose p out of n features. Therefore, 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 in the sufficient sample size given by Lemma 4.1 we get

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

which is in turn bounded by

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

as desired. □

Theorem 4.1 gives an upper bound on the sample complexity of any algorithm that implements the MIN-FEATURES bias. It is interesting to note that

by adopting this bias, the number of examples sufficient for learning grows only logarithmically in the number of irrelevant features, and linearly in the complexity of the concept being learned.

4.4.2 Lower Bound

In the previous section, we proved an upper bound on the number of examples needed by algorithms that implement the MIN-FEATURES bias. We now show that this bound is tight by exhibiting an identical lower bound using the methods developed by Blumer et al. [Blumer *et al.* 87a] exploiting the Vapnik-Chervonenkis dimension (VC-dimension).

Definition: Let us say that a set of objects $I \subseteq \{0,1\}^n$ is *shattered* by a set of concepts C if the objects in I can be labeled by the concepts in C in all the possible $2^{|I|}$ ways. That is, for C to shatter I, we require that for each of the $2^{|I|}$ ways of labeling the elements of I, there exists some concept $c \in C$ that is consistent with that labeling. The *VC-dimension* of C, denoted $VCdim(C)$, is then defined as the cardinality of the largest set of objects that is shattered by C. □

In [Blumer *et al.* 87a], it is shown that the number of examples needed for learning any class of concepts strongly depends on the VC-dimension of that class. Specifically, we will make use of the general lower bound on sample complexity given in [Ehrenfeucht *et al.* 88] which we state here without proof.

Theorem 4.2 (Ehrenfeucht, Haussler, Kearns and Valiant) *Let C be a set of concepts and $0 < \epsilon, \delta < 1$. Then, any algorithm that learns all the concepts in C with respect to ϵ, δ and all probability distributions must use a sample of size*

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

To be able to apply this result here, we need to derive the VC-dimension of $C_{n,s}$ as defined in Theorem 4.1. This is given by the following lemma.

Lemma 4.2 *Let $C_{n,s}$ be as in Theorem 4.1. Then,*

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

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

Definition: Let $\alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_r$ be r Boolean variables. 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 d Boolean variables $\alpha_{r-d+1}, \alpha_{r-d+2}, \alpha_{r-d+3}, \cdots, \alpha_r$ (which, of course, can be viewed as functions).

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

Definition: Let V be a vertical bit-vector

$$\begin{pmatrix} v_0 \\ v_1 \\ \cdot \\ \cdot \\ \cdot \\ v_{t-1} \end{pmatrix}$$

where each v_i is either 0 or 1. We say that V is the *bit-vector representation* of the Boolean function $f(\alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_r)$ if t, the length of V, is equal to 2^r and for each $0 \leq i < t$, v_i is equal to the value of f when the variables $\alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_r$ are assigned the values imposed by the binary representation of i in the obvious manner. That is, V is the bit-vector representation of f if V matches the right-most column of the truth table of f.

Further, we define Boolean operations on bit-vectors by applying the operations bit-wise on the components of these vectors. That is, the i-th bit of "V_1 *op* V_2" is obtained by applying *op* on the i-th bits of V_1 and V_2, where *op* is either $\vee$ or $\wedge$. Likewise, the i-th bit of $\neg V_1$ is just the complement of the i-th bit of V_1. □

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 $g_1, g_2, \cdots, g_z$. For example, let

$$\begin{aligned} g(x_1, x_2, x_3) &= x_1 \\ h(x_1, x_2, x_3) &= x_2 \vee x_3 \\ f(x_1, x_2, x_3) &= x_1 \vee (\neg x_2 \wedge \neg x_3) \end{aligned}$$

Clearly, $f = g \vee \neg h$. In the bit-vecor representation, this is given as

$$\overset{f}{\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}} = \overset{g}{\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}} \vee \neg \overset{h}{\begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}}$$

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

$$\begin{aligned} &f(\alpha_1, \alpha_2, \cdots, \alpha_i, \cdots, \alpha_r) = \\ &\qquad (\neg\alpha_i \wedge f(\alpha_1, \alpha_2, \cdots, 0, \cdots, \alpha_r)) \vee (\alpha_i \wedge f(\alpha_1, \alpha_2, \cdots, 1, \cdots, \alpha_r)) \end{aligned}$$

will be called *unfolding* of the function f on the variable α_i. □

Lemma 4.3 *Any 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 of $\alpha_1, \alpha_2, \cdots, \alpha_{r-1}$. Unfolding these two functions again on α_{r-1}, f will be expressed in terms of α_{r-1}, α_r and four functions of $\alpha_1, \alpha_2, \cdots, \alpha_{r-2}$. 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 of $\alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_{r-d}$. All of these are in $\mathcal{F}_{r,d}$ and thus the lemma follows. □

Proof of Lemma 4.2:

Let $p = \log_2(s - n)$. We prove the lemma by constructing a set of objects 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 objects as a matrix M of width n, in which each row represents an object 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 object in the collection of objects being constructed. An arbitrary labeling of these objects is viewed as a bit-vector. The proof proceeds by showing that, for any labeling of the objects, the resulting 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 objects, there exists a Boolean concept defined over only p features of $x_1, x_2, \cdots, x_n$ (that is, a concept in $C_{n,s}$) which labels the objects as in ℓ. Hence, the set of objects 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 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

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

The remaining columns of M can be filled arbitrarily.

Let V be a bit-vector representing an arbitrary labeling of the 2^p objects 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 objects is shattered by $C_{n,s}$ and hence, $VCdim(C_{n,s}) \geq 2^p = s - n$.

Part 2: 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 representations of each function in $\mathcal{F}_{r,d}$ constitute a column of the matrix (that is to 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 $\neg g$.[8] 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

$$\frac{2^{2^{r-d}}}{2} + d \,. \tag{4.14}$$

[8]It does not matter which one is removed.

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:

Claim1: The number of columns of M does not exceed n. That is, the quantity given Equation 4.14 is bounded by n.

Claim2: An arbitrary bit-vector of length 2^r can be expressed using Boolean operations over at most p columns of M, and hence that the rows of M are shattered by $C_{n,s}$.

Claim 1 is shown as follows:

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

Claim 2 is shown using Lemma 4.3 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 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

$$2^r = 2^{\lfloor \log_2 p + \log_2 \log_2 n \rfloor - 2}$$

$$\begin{aligned} &\geq 2^{\log_2 p + \log_2 \log_2 n - 3} \\ &= \frac{1}{8} p \log_2 n \\ &= \frac{1}{8} \log_2(s-n) \log_2 n \end{aligned}$$

as desired. □

We are now ready to state our lower bound on sample complexity.

Theorem 4.3 *Under the same conditions as Theorem 4.1, any algorithm that learns every concept in $C_{n,s}$ 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 Lemma 4.2 and Theorem 4.2. □

It should be noted that the lower bound given by Theorem 4.3 is within only a constant factor of the upper bound given by Theorem 4.1.

4.4.3 Discussion

Theorems 4.1 and 4.3 show that

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

training examples are required to learn an arbitrary Boolean concept of complexity s that is defined on n features, for accuracy parameter ϵ and confidence parameter δ. In terms of the number of relevant features p of the target concept, this corresponds to

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

training examples since $s = 2^p + n$.

An important implication of this result is that the presence of many irrelevant features does not make the learning task substantially harder, at least in terms of the sample complexity. This is because the number of available features n appears only logarithmically in the above bound. 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.

The results of this section also tell us that the mechanism used by a learning algorithm to resolve ties under the MIN-FEATURES bias affects the sample complexity of the algorithm only within a constant factor. This is true because the sample complexity of *all* algorithms that implement the MIN-FEATURES bias is within some constant factor of the minimum possible.

4.5 Computational Aspects of Implementing the MIN-FEATURES Bias

Now that we have analyzed the sample complexity of algorithms that implement the MIN-FEATURES bias, we turn in this section to the computational aspects of implementing this bias.

Implementing the MIN-FEATURES bias can be divided into the following two tasks:

- **Task 1:** Searching for a subset of features that is sufficient to construct an hypothesis consistent with the training sample.
- **Task 2:** Constructing a consistent hypothesis from a sufficient set of features.

In the following subsections, we discuss the computational costs of implementing each of these tasks. We will see that Task 2 can be solved quite efficiently, and that the computational complexity of implementing the MIN-FEATURES bias is in fact dominated by the Task 1. Then, in the succeeding section, we will introduce two algorithms that implement the MIN-FEATURES bias.

4.5.1 Constructing a Consistent Hypothesis Given a Sufficient Set of Features

Consider the following theorem.

Theorem 4.4 *Suppose S is a sample of some concept, and Q is a subset of features. Then, Q is sufficient to form an hypothesis consistent with S if and only if there exist no two examples $\langle X_1, class_1\rangle, \langle X_2, class_2\rangle \in S$ such that*

- *$class_1 \neq class_2$, and*
- *X_1 and X_2 have the same truth assignments to the features in Q. That is, for every i such that $x_i \in Q$, it is true that $X_1^{(i)} = X_2^{(i)}$.*

Proof:

(1) The "If" Part: The sufficiency is established by actually constructing a consistent hypothesis defined only on the features in Q. A simple construction method is as follows:

1. For each positive example in S, construct a conjunction of literals in which each feature in Q appears negated if the value of the feature in the example is 0, or un-negated otherwise. More formally, for each $\langle X_i, 1\rangle \in S$, construct a conjunction $\prod_{j=1}^{n} l(X_i^{(j)})$ where

$$l(X_i^{(j)}) = \begin{cases} \neg x_j & \text{if } x_j \in Q \text{ and } X_i^{(j)} = 0 \\ x_j & \text{if } x_j \in Q \text{ and } X_i^{(j)} = 1 \\ 1 & \text{otherwise} \end{cases}$$

   For example, if $\langle 011001, 1\rangle$ is a training example in S and $Q = \{x_2, x_4, x_5\}$, then the resulting conjunction is $x_2\neg x_4\neg x_5$.

2. Let the hypothesis h be the disjunction of all the conjunctions formed in the first step.

It is clear that h uses only the features in Q. It is also clear that h is consistent with every positive example in the sample. This is true since the conjunction formed from a positive example will be satisfied by that example and, hence, will cause h to be 1 for the example. Hence, the only potential problem is the possibility that h classifies some of the negative examples as positive. But for a negative example

$\langle X, 0\rangle$ to be classified as positive by h, it must satisfy some conjunction in h. This can happen only if the training sample contains some positive example $\langle Y, 1\rangle$ such that the values of all the features of Q in both X and Y are matching, which is ruled out by assumption.

(2) The "Only If" Part: Suppose there exists a pair of examples $\langle X_1, \text{class}_1\rangle$ and $\langle X_2, \text{class}_2\rangle \in S$ with the given properties. Then, for any function z defined only on the features in Q, it must be true that $z(X_1) = z(X_2)$ since X_1 and X_2 share the same values for the features in Q. Therefore, z disagrees with either $\langle X_1, class_1\rangle$ or $\langle X_2, class_2\rangle$, and hence, z is inconsistent with S. □

It should be clear that the construction method given in the "If" part of the above proof runs in time linear in the total number of features and the size of the training sample, i.e. $O(nm)$. This means that once a sufficient set of features is identified, the second step of implementing the MIN-FEATURES bias (constructing an hypothesis consistent with the training sample) can be done quite efficiently. The major computational cost in implementing the MIN-FEATURES bias lies, therefore, in the first step—finding a minimal set of features sufficient to form a consistent hypothesis.

We should mention, however, that the above construction method is given only as an example to show that constructing a consistent hypothesis from a sufficient set of features is easy. In practice, one may seek an hypothesis of better quality (rather than "any consistent hypothesis") by employing other methods that can be computationally more expensive (e.g., running ID3 but restricting it to use only the features in the chosen subset).

4.5.2 Testing the Sufficiency of a Subset of Features

Given a training sample S and a subset of features Q, how can we judge whether Q is sufficient to construct an hypothesis consistent with S? As done in the function "Sufficient" of Figure 11, all we need to do is simply to search in the

Function Sufficient(*Features, Sample*)

1. For every pair E_- and $E_+ \in$ *Sample* of negative and positive examples:
   If the values of the features in *Features* are the same in E_- and E_+, return *false*.
2. Return *true*.

end.

Figure 11. Simple but inefficient implementation of Sufficient

sample for two examples that have the same values for all the features in Q but have conflicting classes. According to Theorem 4.4, the subset Q is sufficient to construct a consistent hypothesis if and only if such a pair cannot be found in the training sample. However, testing the sufficiency of a given subset of features can be implemented more efficiently. To continue the discussion, it is convenient to make the following definitions.

Definitions: A *conflict* generated by a pair of examples $\langle X_1, 1\rangle$ and $\langle X_2, 0\rangle$ is an n-bit vector $\langle a_1 a_2 \cdots a_n\rangle$ 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 $i = 0, 1, 2, \cdots, n$. We say that the feature x_i *explains* the conflict if and only if a_i is 1.

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

For instance, the examples $\langle 00110010, 1\rangle$ and $\langle 10100011, 0\rangle$ define the conflict $\langle 10010001\rangle$. This conflict is explained by the features x_1, x_4 and x_8 but not explained by the features x_2, x_3, x_5, x_6 and x_7.

With the above definition, Theorem 4.4 can be restated as follows:

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 12. Efficient implementation of Sufficient

Corollary 4.1 *A set of features Q is sufficient to form an hypothesis consistent with a training sample S if and only if every conflict generated from the examples in S is explained by some feature in Q.*

The function Sufficient given in Figure 11 effectively works by generating all the possible conflicts from the training sample and checking that every generated conflict is explained by some feature in the given subset of features. Clearly, this takes time quadratic in the size of the training sample in the worst case. In Figure 12, we give an alternative method that avoids this by following a divide-and-concur strategy. The following theorem shows that this implementation of a sufficiency test is correct.

Theorem 4.5 *The algorithm given in Figure 12 returns "true" if and only if the*

set "Features" is sufficient to construct an hypothesis consistent with all the training examples in "Sample".

Proof:

We prove this theorem by induction on the size of the set *Features*.

The basic step is when $Features = \phi$. The only concepts that we can construct with 0 features are the *true* and *nil* concepts. Therefore, the returned value must be *true* if all the examples have a common class, or *false* otherwise. This is what the algorithm does in Steps 1 and 2.

Now, suppose that the algorithm is correct when $|Features| \leq k$ for some $k \geq 0$. We need to show that it is also correct when $|Features| = k+1$. Upon partitioning *Sample* into *Sample0* and *Sample1* based on the feature x in Steps 4 and 5, let us partition the set of all conflicts generated from *Sample* into the following three disjoint sets:

- *Conflicts–0–0* in which each conflict is generated from two examples that are both in *Sample0*.
- *Conflicts–1–1* in which each conflict is generated from two examples that are both in *Sample1*.
- *Conflicts–0–1* in which each conflict is generated from two examples, one from *Sample0* and the other from *Sample1*.

According to Corollary 4.1, we should return *true* if and only if the set *Features* explains all the conflicts in *Conflicts–0–1* $\bigcup$ *Conflicts–0–0* $\bigcup$ *Conflicts–1–1*. Obviously, all the conflicts in *Conflicts–0–1* are explained by x. On the other hand, none of the conflicts in *Conflicts–0–0* $\bigcup$ *Conflicts–1–1* are explained by x. Therefore, we should return *true* if and only if the set $Features - \{x\}$ explains all the conflicts in *Conflicts–0–0* $\bigcup$ *Conflicts–1–1*. By assumption, the recursive calls in Step 6 return the correct conclusions since $|Features - \{x\}| = k$. Since Step 6 returns *true* if and

only if both calls return *true*, the algorithm is also correct for $|Features| = k + 1$. This completes the proof. □.

Let us now find out the time complexity of the algorithm of Figure 12. The algorithm successively splits the training sample using a different feature at each recursive level. Suppose that the set of features being tested is $\{x_{j_1}, x_{j_2}, x_{j_3}, \cdots, x_{j_k}\}$, and that the algorithm chooses x in Step 3 in such order. At the initial call of the procedure, the sample is partitioned into two disjoint sets based on the feature x_{j_1}. The algorithm will then be called on each of these sets. At the first recursion, each of the two sets will be partitioned further into two disjoint sets based on x_{j_2}, and thus there will be four disjoint sets at this level. Partitioning continues recursively in this manner for the rest of the features, except that no recursion will take place on sets in which all the examples share the same class in which case the algorithm will just return *true* in Step 1. Therefore, at the i-th level of recursion the sample will be partitioned into at most 2^i disjoint subsets. Let us denote by S_t^i the t-th subset in the i-th level of recursion (See Figure 13). It should be clear that for any t_1 and t_2, $S_{t_1}^i \bigcap S_{t_2}^i = \phi$ and that $\bigcup_t S_t^i = Sample$, since these constitute a partitioning of *Sample*. It is easy to see that the cost of running Step 1 through Step 5 on any S_t^i is just $O(|S_t^i|)$. Therefore, the total cost of running these steps on all the subsets in a particular level is just $O(|Sample|)$.

Step 2 ensures that there will be no more than $|Features|$ levels of recursion, and therefore, the overall worst-case time complexity of this procedure is $O(|Sample||Features|)$. This is only linear in the number of features and the size of the training sample, and is certainly an improvement on the procedure of Figure 11.

4.5.3 Searching For a Sufficient Subset of Features

Let us formalize the problem of searching for a subset of features that is sufficient to construct a consistent hypothesis as follows:

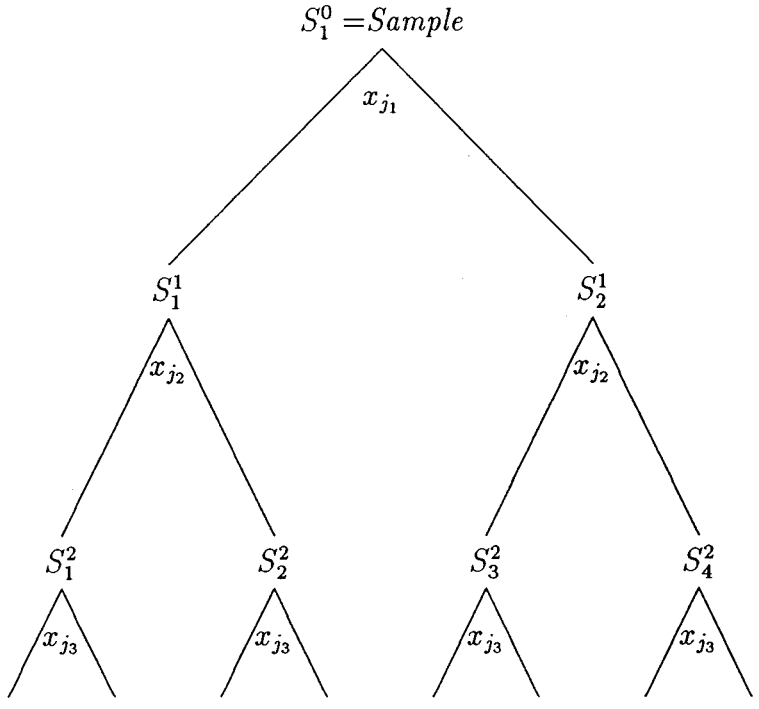


Figure 13. The upper three recursion levels of the function Sufficient.

The MIN-FEATURES Problem:

Given:
- A set $V_n = \{x_1, x_2, \cdots\cdots, x_n\}$ of Boolean variables.
- A set $S = \{\langle X_i, f(X_i)\rangle | i = 1, \cdots, m\}$ where each X_i is an assignment to the variables in V_n and f is some (unknown) Boolean function.

Find: A set $Q \subseteq V_n$ of minimum cardinality such that there exists a Boolean function f' defined only on those variables in Q satisfying the condition

$$\forall_i \quad \langle X_i, f(X_i)\rangle \in S \Longrightarrow f'(X_i') = f(X_i)$$

where X_i' is the assignment to the variables in Q as specified by X_i.

The set S can be viewed as an incomplete (or, partial) truth table T_S of the function f defined on all the n variables in V_n. Arbitrarily filling each of the empty entries in T_S by either 0 or 1 results in a complete truth table, which, of course, represents some Boolean function that is consistent with S. There are certainly many ways to complete T_S. Suppose that each way of completing T_S is assessed by the number of variables required to represent the resulting truth table as a Boolean formula. The objective here is to minimize this number. That is, the goal of the MIN-FEATURES problem is to come up with some set of variables Q of minimum cardinality such that there exists some way to complete T_S leading to a truth table representable as a Boolean function that uses only those variables in the chosen Q (see the example in Figure 10).

To investigate the computational hardness of the MIN-FEATURES problem, we draw an interesting analogy between this problem and a well-studied problem known as the "MINIMUM-SET-COVER problem" [Garey and Johnson 79]. Specifically, we show that these two problems are strongly related in the sense that an instance of one problem can easily be transformed to an instance of the

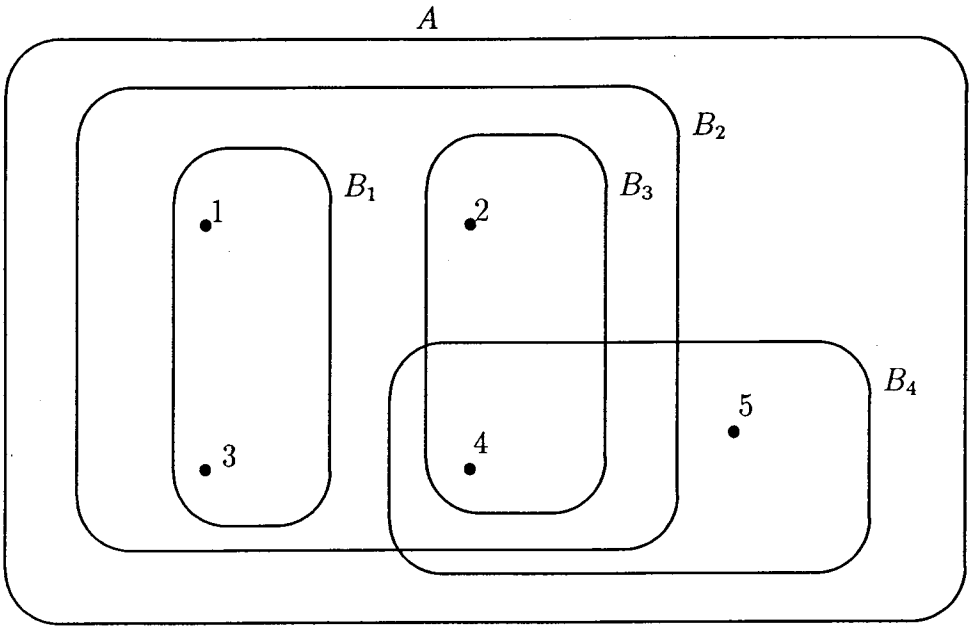


Figure 14. An example of the MINIMUM-SET-COVER problem.

other problem, and therefore, any algorithm that solves one of these problems can immediately be turned into an algorithm that solves the other problem.

First, let us state the MINIMUM-SET-COVER problem formally.

The MINIMUM-SET-COVER Problem:

Given: - A finite set A of cardinality l.

- A set $B = \{B_1, B_2, \cdots, B_v\}$ where $B_i \subseteq A$ for $i = 1, \cdots, v$ and such that $\bigcup_{i=1}^{v} B_i = A$.

Find: A set $B' \subseteq B$ of minimum cardinality such that $\bigcup_{B_i \in B'} B_i = A$.

Usually, an element $a \in B_i$ is said to be *covered* by B_i. The goal is, therefore, to come up with the fewest number of subsets from B that together cover every element in A. Figure 14 illustrates an example of the MINIMUM-SET-COVER problem. The reader can verify that the solution for this example is $\{B_2, B_4\}$.

A convenient way to represent an instance of the MINIMUM-SET-COVER problem is to build a matrix with a row for each element in A and a column for each set in B, such that an entry $\langle i, j \rangle$ is 1 if the i-th element in A is covered by the j-th set in B, or 0 otherwise. For the example in Figure 14, this gives

	B_1	B_2	B_3	B_4
e_1	1	1	0	0
e_2	0	1	1	0
e_3	1	1	0	0
e_4	0	1	1	1
e_5	0	0	0	1

In this matrix representation, finding a solution means removing as many columns from the matrix as possible, without letting any of the rows become all 0's.

Now, let us see how one can transform an instance of the MIN-FEATURES problem to an instance of the MINIMUM-SET-COVER and vise versa, such that a solution of the resulting instance yields a solution to the original instance.

From MINIMUM-SET-COVER to MIN-FEATURES:

Let $n = |B|$ and create n Boolean variables $\{x_1, x_2, \cdots x_n\}$ where each x_i corresponds to $B_i \in B$. For each element $e \in A$, we create a negative example $\langle a_1 a_2 \cdots a_n, 0 \rangle$, where $a_i = 1$ if $e \in B_i$ and 0 otherwise, for $i = 1, \cdots, n$. That is, for each row in the matrix representation of the MINIMUM-SET-COVER instance, we create a negative example $\langle X, 0 \rangle$ where X is just the row itself. In addition, we create one positive example $\langle \overbrace{000\ldots00}^{n}, 1 \rangle$.

In the resulting MIN-FEATURES instance, exactly one conflict can be generated for each negative example, and thus, there is one-to-one correspondence between the conflicts to be explained in the resulting MIN-FEATURES instance and the elements to be covered in the original MINIMUM-SET-COVER instance. Hence, a solution to the resulting MIN-FEATURES instance immediately gives a solution to the original MINIMUM-SET-COVER instance. □

From MIN-FEATURES to MINIMUM-SET-COVER:
The transformation in the opposite direction is quite obvious. We only need to generate the set of all the conflicts out of the training sample, and then construct a matrix by letting each conflict form a row of the matrix. This matrix clearly represents a MINIMUM-SET-COVER instance whose solution immediately yields a solution to the original MIN-FEATURES instance. □

The above two-way transformation shows that the MINIMUM-SET-COVER and MIN-FEATURES problems are computationally equivalent in that any algorithm that solves one of these problems can immediately be turned into an algorithm that solves the other problem.

The MINIMUM-SET-COVER problem is known to be NP-hard [Garey and Johnson 79]. At a first glance, this may appear to mean that we have to give up the possibility of implementing the MIN-FEATURES bias in polynomial time (unless P=NP). This conclusion, however, does not provide an adequate view for the reason given below.

Generally, the computational complexity of an algorithm is measured in terms of the size of its input. The NP-hardness of the MINIMUM-SET-COVER problem implies that (unless P=NP) there exists no algorithm that solves this problem in time polynomial in v, the number of elements to be covered and l, the number of subsets to be used to form a minimum cover. This in turn implies that there exists no algorithm that solves the MIN-FEATURES problem in time polynomial in n, the number of features and m, the number of training examples.

However, the usual practice in computational learning theory is to include s, the complexity of the target concept, in the parameters used to assess the time complexity of a learning algorithm. Thus, by polynomially implementing the MIN-FEATURES bias we mean constructing learning algorithms that implement this bias and run in time polynomial in s among other parameters. In our setting, if the number of features relevant to the target concept is p, then the complexity of the

concept is $s = 2^p + n$. Therefore, being polynomial in n, m and s is equivalent to being polynomial in n, m and 2^p, or any constant raised to the power p (i.e., something like n^p, for example, is not acceptable). For the MINIMUM-SET-COVER problem, this corresponds to the following: Given any instance of MINIMUM-SET-COVER that has v elements and l subsets such that there exists a cover of cardinality k, it is desired to construct an algorithm that finds a solution in time polynomial in v, l and 2^k.

The common practice of including the target concept complexity among the parameters used to measure the computational complexity of learning algorithms is based on the assumption that an appropriate target concept complexity measure is being used. A given target concept complexity measure is appropriate to a given application domain if one can assume that the target concept in that domain is one of those concepts that are relatively less complex (i.e. *simple*) according to that measure.

In our setting, a simple concept is a concept that has a relatively small number of relevant features. Thus, in the MIN-FEATURES problem, it is assumed that there exists a *small* subset of features that is sufficient to construct a hypothesis consistent with the training sample. This in turn corresponds to restricting the MINIMUM-SET-COVER problem to instances that have small covers, and this is why we allow running in time exponential in p and k in these problems.

To summarize, the relation between the MIN-FEATURES and the MINIMUM-SET-COVER problems can be stated as follows:

Theorem 4.6 *Let s denote the complexity of f in the MIN-FEATURES problem and k be the size of the smallest cover for the MINIMUM-SET-COVER problem. Then, the MIN-FEATURES problem is solvable in time polynomial in n, m and s if and only if the MINIMUM-SET-COVER problem is solvable in time polynomial in v, l and 2^k.*

Proof:

Follows immediately from the two polynomial-time transformation methods given in page 113. □

The question now becomes: Can we solve the MINIMUM-SET-COVER problem in time polynomial in v, l and 2^k? Of course, this is more lenient compared to requiring the running time to be polynomial in v and l only. Nonetheless, we do not know the answer to this question. To the best of our knowledge, such a question has not been addressed anywhere, which, unfortunately, leaves our main question regarding the time complexity of the MIN-FEATURES problem open.

In any case, linking the MIN-FEATURES problem to the MINIMUM-SET-COVER problem as we did here is useful in that it provides an alternative avenue in assessing the computational hardness of the MIN-FEATURES problem. Moreover, since the MINIMUM-SET-COVER problem has been addressed by many researchers, the transformation method given in the proof of Theorem 4.6 is useful because it allows the application of algorithms and ideas developed for this problem to the problem of learning in the presence of many irrelevant features.

4.6 Algorithms That Implement the MIN-FEATURES Bias

So far, we have been mainly concerned with assessing the computational hardness of implementing the MIN-FEATURES bias. In this section, we look at actual algorithms that implement this bias as efficiently as possible. Earlier in Subsection 4.5.1, we argued that constructing a consistent hypothesis from a sufficient subset of features can be done efficiently, and that the most difficult task in implementing the MIN-FEATURES bias is the search for such a sufficient subset of features. Hence, in this section, finding a *solution* will be taken to mean identifying a subset of features sufficient to form a consistent hypothesis—that is, solving the MIN-FEATURES problem as defined in Subsection 4.5.3.

Algorithm FOCUS-1 (*Sample*)

1. If all the examples in *Sample* have the same class, return ϕ.
2. *Queue*= $\{\phi\}$. /* This is a first-in-first-out data structure. */
3. Repeat
   3.1. Pop the first element in *Queue*. Call it *Parent*.
   3.2. If *Parent* = ϕ then *Last* = 0,
   else *Last* = the largest index of the features in *Parent*.
   3.3. For i = *Last* +1 to n
      3.3.1 *Current* = *Parent* $\bigcup\{x_i\}$.
      3.3.2 If Sufficient(*Current*, *Sample*), then return *Current*.
      3.3.3 Insert *Current* at the tail of *Queue*.

end.

Figure 15. The FOCUS-1 learning algorithm.

4.6.1 The FOCUS-1 Algorithm

In Subsection 4.5.2, 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 only a 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. An algorithm that follows this method is FOCUS-1 given in Figure 15. This algorithm uses a first-in-first-out queue to systematically generate the subsets of features from the smaller to the larger size. The algorithm terminates whenever a subset for which the test Sufficient (see Figure 12) returns *true* is encountered.

Now, let us compute the computational complexity of FOCUS-1. Suppose that

the target concept 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 1, 2, 3 $\cdots$ up to at most p. Therefore, the number of sufficiency tests performed will be at most

$$\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots\cdots + \binom{n}{p} = O(n^p)\ .^9$$

As shown in Subsection 4.5.2, 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 is just $O(m \log(s-n) n^{\log(s-n)})$.

4.6.2 The FOCUS-2 Algorithm

Although FOCUS-1 runs in time quasi-polynomial in n and s, it is clearly impractical in applications where n and s are large. The major computational cost of this simple algorithm stems from the rather *blind* search it performs in the space of feature subsets. Let us see this through the following example.

Example: Let the training sample be

$\langle 010100, + \rangle$ $\quad$ $\langle 011000, - \rangle$
$\langle 110010, + \rangle$ $\quad$ $\langle 101001, - \rangle$
$\langle 101111, + \rangle$ $\quad$ $\langle 100101, - \rangle$

Then, the conflicts generated from this sample are

$a_1 = \langle 001100 \rangle$ $\quad$ $a_4 = \langle 101010 \rangle$ $\quad$ $a_7 = \langle 110111 \rangle$
$a_2 = \langle 111101 \rangle$ $\quad$ $a_5 = \langle 011011 \rangle$ $\quad$ $a_8 = \langle 000110 \rangle$
$a_3 = \langle 110001 \rangle$ $\quad$ $a_6 = \langle 010111 \rangle$ $\quad$ $a_9 = \langle 001010 \rangle$

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. □

[9] Note that we are mainly interested in cases where p is much smaller than n.

In the above example, FOCUS-1 tests all the $\binom{6}{0} + \binom{6}{1} + \binom{6}{2} = 22$ subsets of features of size 0, 1 and 2, and some of the $\binom{6}{3} = 20$ 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 explain 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 16 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,

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

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 $M_{\{x_4\},\{x_3\}}$

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 explained by any of the features in A, such that $|Z_a - B|$ is minimized, where Z_a is the set of features explaining 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 16. The FOCUS-2 learning algorithm.

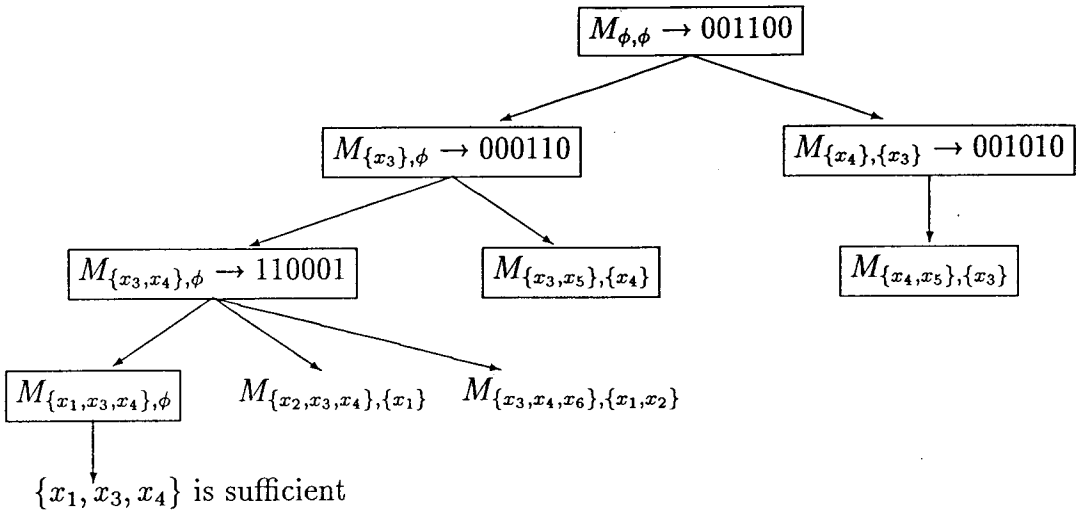


Figure 17. An example of FOCUS-2. Rectangles indicate where the sufficiency tests occurred.

(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 explained 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 17, 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 explain 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.

4.6.3 Time Complexity of FOCUS-2

Clearly, the improvement of FOCUS-2 over FOCUS-1 in computational costs depends on the availability of conflicts that lead to significant pruning. Let's use the term *weight* of a conflict to indicate the number of features that explain the conflict. Then, loosely speaking, the superiority of FOCUS-2 is contingent upon the availability of conflicts of small weight. An *unfortunate* sample in which all the conflicts are of weight close to n causes the number of sufficiency tests done by FOCUS-2 to be close to that of FOCUS-1. Therefore, the worst case computational complexity of FOCUS-2 should be similar to that of FOCUS-1.

However, we can argue that the worst case situation is very unlikely to occur in a sample of reasonably large size. This is because the probability that *all* the conflicts have a large weight rapidly decreases as the size of the training sample grows. To see this, let us assume for simplicity that the probability distribution over the space of instances is uniform and that the target concept has p relevant features out of n available features. Also, let $q = n - p$ be the number of irrelevant features. For convenience, we will let x_1 through x_q be the irrelevant features.

Suppose we draw a conflict a at random. That is, we successively draw training examples until we get a pair of positive and negative examples from which we form the conflict a. We will use W_a to denote the weight of a, and W'_a to denote the

number of irrelevant features that explain a (i.e. the contribution of the irrelevant features to the weight of a). Clearly, $W_a \leq W'_a + p$.

Let $Y_1, Y_2, \cdots, Y_q$ be q random variables such that for each $1 \leq i \leq q$, Y_i is 1 if the feature x_i explains the conflict a, and 0 otherwise. W'_a is, therefore, just the sum $Y_1 + Y_2 + \cdots + Y_q$.

The value of each of the irrelevant features is totally independent from the class of the example. That is, the value of an irrelevant feature in a positive example can be 0 or 1 with equal probability. The same thing also applies for the negative examples. Therefore, the probability that $Y_i = 1$ for each $1 \leq i \leq q$ is just $\frac{1}{2}$.

W'_a can, therefore, be viewed as a binomial random variable with q trials and $\frac{1}{2}$ as the ratio of success. Thus, for any integer $w, 0 \leq w \leq q$, the probability that there are w or more irrelevant features that explain a is given by

$$Pr[W'_a \geq w] = \sum_{i=w}^{q} \binom{q}{i} (\frac{1}{2})^i (\frac{1}{2})^{q-i} = 2^{-q} \sum_{i=w}^{q} \binom{q}{i} .$$

Since $W_a \leq W'_a + p$, the probability that a is explained by $w + p$ or more features is, therefore, bounded as follows:

$$Pr[W_a \geq w + p] \leq 2^{-q} \sum_{i=w}^{q} \binom{q}{i} .$$

Thus, if we have l independently drawn conflicts, then the probability that *all* of these conflicts have weight no less than $w + p$ is at most

$$\{2^{-q} \sum_{i=w}^{q} \binom{q}{i}\}^l$$

which is rapidly diminishing as l increases.

To clarify this, let us look at a numerical example. Suppose $n = 20$ and $p = 5$ (i.e. the target concept has 5 relevant features out of 20 available ones). Letting $w = 6$, the probability that a randomly drawn conflict has weight larger than 10 is at most

$$2^{-15} \sum_{i=6}^{15} \binom{15}{i} \simeq 0.85$$

Table 3. The number of the relevant features in the target concepts.

Concept	1	2	3	4	5	6	7	8	9
Relevant Features	5	5	5	6	6	6	7	7	7
Total Features	15	20	25	15	20	25	15	20	25

Therefore, if we have 20 independently drawn conflicts, then the probability that all of these conflicts are of weight greater than 10 is at most $(0.85)^{20} \approx 0.039$. If the number of conflicts is increased to 40, this probability reduces to $(0.85)^{40} \approx 0.0015$. Further, with 60 independent conflicts, this probability becomes at most $(0.85)^{60} \approx 5.82 \times 10^{-5}$.

These numbers show that it is almost certain that a training sample of reasonable size contains conflicts that enable FOCUS-2 to attain significant cuts in the search space, and thus to outperform FOCUS-1 in terms of the amount of computation required.

4.6.4 Empirical Comparison Between FOCUS-1 and FOCUS-2

To compare the computational costs of FOCUS-1 and FOCUS-2, we have implemented and tested the two algorithms on 9 different target concepts. The concepts, numbered as Concept 1 through Concept 9, were chosen randomly but such that the number of the relevant features in each concept is as given in Table 3.

For each of the concepts, the experiment was conducted by randomly drawing (under the uniform distribution, sampling with replacement) a sample of size m, where m was varied from 10 to 220 examples. For each training sample, the two algorithms were run and the sufficiency tests performed by each algorithm were counted. For each sample size, the number of sufficiency tests performed by each algorithm was averaged over 10 runs. The tables in Figure 18 show the results for each of the 9 concepts. These tables list the number of sufficiency tests performed

by FOCUS-1 and FOCUS-2 and the ratio of these for each sample size.

The general observation is that FOCUS-2 outperforms FOCUS-1 by roughly two orders of magnitude for the nine concepts we have tested. It is interesting to note that the number of sufficiency tests done by FOCUS-1 remains steady regardless of the training sample size, since this algorithm always follows the same steps regardless of the contents of the sample. FOCUS-2, on the other hand, makes a fewer sufficiency tests as the number of training examples increases. This is because with a larger sample there is a greater chance of getting conflicts of small weight, and consequently, a better chance for significant reduction in the number of sufficiency tests needed.

Before ending this section, we should mention that, although FOCUS-2 performs a significantly smaller number of sufficiency tests than FOCUS-1, finding the best conflict each time in Step 4.3 of FOCUS-2 does not come for free. To include this *overhead* in the comparison, one needs to measure the actual execution time of each algorithm. Experiments of this type will be reported later in Section 5.5. In addition to FOCUS-1 and FOCUS-2, the experiments include a heuristic we call the "Weighted-Greedy" Algorithm to be introduced in the next chapter.

4.7 Summary

The goal of this chapter was to study the problem of learning Boolean concepts in domains where many irrelevant features are assumed to be present in the training examples. We introduced and studied the MIN-FEATURES bias which prefers consistent hypotheses definable over as few features as possible.

We analyzed the sample complexity of any probably-approximately correct (PAC) learning algorithm that implements the MIN-FEATURES bias. It was shown that

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

training examples are required to PAC-learn a binary concept involving p input

	Concept1: 5 out of 15			Concept2: 5 out of 20			Concept3: 5 out of 25		
Ex's	FOCUS-1	FOCUS-2	Ratio	FOCUS-1	FOCUS-2	Ratio	FOCUS-1	FOCUS-2	Ratio
20	1292.2	88.2	14	2331.5	208.5	11	3939.3	507.2	7
60	3419.7	59.0	57	13868.3	380.9	36	43477.2	1708.5	25
100	4069.2	40.8	99	15555.7	209.9	74	47455.1	960.7	49
140	3792.9	19.7	192	15891.9	159.7	99	41603.3	669.1	62
180	3578.4	17.1	209	14564.4	123.2	118	46626.3	505.8	92
220	2970.2	12.8	232	9723.7	70.3	138	51285.2	477.7	107

	Concept4: 6 out of 15			Concept5: 6 out of 20			Concept6: 6 out of 25		
Ex's	FOCUS-1	FOCUS-2	Ratio	FOCUS-1	FOCUS-2	Ratio	FOCUS-1	FOCUS-2	Ratio
20	979.7	77.1	12	1628.1	167.7	9	2770.2	366.7	7
60	7069.8	122.7	57	42580.1	986.0	43	155010.7	6253.7	24
100	8511.6	66.9	127	47001.1	602.3	78	186207.8	2978.3	62
140	7914.6	27.1	292	46531.1	401.0	116	162895.7	2559.4	63
180	7790.7	16.1	483	43706.8	245.1	178	175410.5	1550.6	113
220	6639.2	13.1	506	32274.1	155.6	207	192860.4	1412.2	136

	Concept7: 7 out of 15			Concept8: 7 out of 20			Concept9: 7 out of 25		
Ex's	FOCUS-1	FOCUS-2	Ratio	FOCUS-1	FOCUS-2	Ratio	FOCUS-1	FOCUS-2	Ratio
20	1024.3	95.6	10	1942.4	218.0	8	2913.4	372.5	7
60	7358.1	128.2	57	42675.1	1017.0	41	161003.9	6315.0	25
100	13736.7	129.0	106	113473.4	1631.2	69	508762.7	11087.8	45
140	13176.3	52.2	252	106263.5	823.2	129	494819.7	6828.1	72
180	13394.4	32.2	415	103811.1	546.3	190	545982.1	4121.1	132
220	12514.5	23.1	541	81521.9	345.2	236	539462.1	3839.6	140

Figure 18. Comparison between FOCUS-1 and FOCUS-2. For each target concept, the figures indicate the number of sufficiency tests done by each algorithm, and the ratio of that number for FOCUS-1 to FOCUS-2. For each sample size, the results are averaged over 10 runs.

features (out of a space of n input features) with accuracy parameter ϵ and confidence parameter δ. In this bound, 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.

The computational complexity of implementing the MIN-FEATURES bias was discussed by showing an equivalence relationship between implementing this bias and solving the MINIMUM-SET-COVER problem. Following this analysis, we have introduced FOCUS-1 and FOCUS-2, which are two algorithms that implement the MIN-FEATURES bias. FOCUS-1 was analytically shown to be a quasi-polynomial time algorithm while FOCUS-2 was empirically shown to be substantially more efficient than FOCUS-1.

Chapter 5

Learning With Many Irrelevant Features: Approximation Methods

5.1 Motivation

Exact implementation of the MIN-FEATURES bias (as done by the FOCUS-1 and FOCUS-2 algorithms given in the previous chapter) in domains with large numbers of features can be computationally infeasible. In such cases, one may be willing to employ efficient heuristics that provide good but not necessarily optimal solutions. The goal of this chapter is to study learning algorithms that approximate the MIN-FEATURES bias—that is, algorithms which produce hypotheses that use a number of features that is reasonably close to the minimum possible while being, at the same time, computationally efficient.

Previously in Theorem 4.1, we have derived the sample complexity, or the number of training examples sufficient to learn, when the learning algorithm is implementing the exact MIN-FEATURES bias. The first question one should ask here is what happens to this sample complexity if the learning algorithm only approximates the MIN-FEATURES bias?

Let us say that a learning algorithm L ω-approximates the MIN-FEATURES bias, if for any training sample S, L returns an hypothesis that is consistent with S and defined using at most ωp features, where p is the number of features used in defining the hypothesis of an algorithm that implements the exact

MIN-FEATURES bias when given S. Then, we can easily generalize Theorem 4.1 as follows:

Theorem 5.1 *Let $C_{n,s}$ be the set of concepts defined on n features with complexity at most s. Then, under any probability distribution D over the space of objects, 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 size*

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

is sufficient to guarantee that any algorithm ω-approximating the MIN-FEATURES bias learns c with respect to ϵ, δ and D.

Proof:

We follow the same steps as in the proof of Theorem 4.1. For any target concept defined on p features (that is, any concept of complexity $s = 2^p + n$), the hypothesis of any algorithm that ω-approximates the MIN-FEATURES bias must be defined on at most ωp features. The number of concepts one can define on at most ωp specific features is $2^{2^{\omega p}}$. Also, there are $\binom{n}{\omega p}$ ways to choose ωp out of n features. Therefore, the size of the hypothesis space of the algorithm is at most

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

Substituting in the sufficient sample size given by Lemma 4.1 we get

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

as desired. □

Clearly, Theorem 5.1 is a generalization of Theorem 4.1 since in this latter theorem ω is just set to 1. It is interesting to note that the sample complexity of any algorithm that ω-approximates the MIN-FEATURES bias for some ω independent of n, remains logarithmic in n.

Another important implication of Theorem 5.1 is the trade-off between the sample complexity of a learning algorithm and the *degree* to which it approximates the MIN-FEATURES bias. Since better approximation naturally means higher computational costs, this in turn implies a trade-off between the sample and computational complexities of algorithms that approximate the MIN-FEATURES bias. That is, we can choose to have a learning algorithm that is computationally efficient but requires a large amount of training data, or vice versa. Hence, Theorem 5.1 tells us that the topic of this chapter—seeking efficient approximation algorithms for the MIN-FEATURES bias—is well-motivated.

At first glance, it may appear that there are already many algorithms that approximate the MIN-FEATURES bias. For example, ID3 [Quinlan 86] has a bias in favor of "small" decision trees, and small trees would seem to test only a subset of the input features. In this chapter, we will present experiments comparing the FOCUS algorithms introduced in the previous chapter to ID3 and FRINGE [Pagallo and Haussler 90]. These experiments demonstrate that—despite common belief—ID3 and FRINGE are not good implementations of 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 FOCUS algorithms.

We will then introduce a collection of candidate heuristics that approximate the MIN-FEATURES bias with various degrees of success. The validity of these heuristics is assessed by empirically comparing them to the ID3, FRINGE, FOCUS-1 and FOCUS-2 algorithms. We will show that one of our heuristics (the "Weighted-Greedy" algorithm) provides excellent approximation to the FOCUS algorithms while being substantially more efficient than these algorithms.

5.2 Feature Selection in Pattern Recognition

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 of the work in that area seeks to enhance the computational efficiency of particular classifiers while leaving their accuracy unaffected, whereas the goal of this chapter is to improve the accuracy of the classifier (which can be any Boolean function) by selecting the minimal number of features.

The typical case studied in pattern recognition involves a classifier that is capable of performing quite well without feature selection (i.e., using all of the available features). However, for ease of hardware implementation and speed of processing, it is necessary to reduce the number of features considered by the classifier. Generally, the classifiers studied in pattern recognition have the so-called monotonicity property ([Narendra and Fukunaga 77]) that as the number of features is reduced, the accuracy decreases. The goal of feature selection there is to eliminate as many features as possible without significantly degrading performance.

Most feature selection criteria in pattern recognition are defined with respect to a specific classifier or group of classifiers. For example, [Kittler 80] 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 [Ichino and Sklansky 84a], the linear classifier [Ichino and Sklansky 84b] and the Bayes classifier [Queiros and Gelsma 84]. 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 [Morgera 86, Mucciardi and Gose 71].

To our knowledge, the problem of finding the smallest (or nearly-smallest) subset of Boolean features that is sufficient to construct a consistent hypothesis

(regardless of the *form* of the hypothesis)—which is the topic of this chapter—has not been addressed.

5.3 How Good Are ID3 and FRINGE in Handling Irrelevant Features?

Several learning algorithms appear to have biases similar to the MIN-FEATURES bias. In particular, algorithms related to ID3 [Quinlan 86] (see Section 2.5) attempt to construct "small" decision trees. These algorithms construct the decision tree top-down (i.e., starting at the root) by recursively splitting the sample on one of the features, and they terminate as soon as they find a tree consistent with the training examples. The feature tested at each node is chosen by measuring the mutual information between the value of this feature as observed in the training examples and the classification of these examples. In a sense, this criterion measures the *estimated relevance* of a feature to the target concept, and thus, a feature that is totally irrelevant to the target concept is not expected to be selected. Hence, the ability of these algorithms to handle the irrelevant features has never been questioned.

In this section, we test these algorithms to see how well they learn in situations where many irrelevant features are present. In particular, we compare three algorithms:

1. **ID3:** As described in Section 2.5, resolving ties randomly when two or more features look equally good.

2. **FRINGE:** As described in Section 2.6, with the maximum number of iterations set to 10.

3. **FOCUS+ID3:** First, a minimum set of features sufficient to produce a consistent hypothesis is obtained as in the FOCUS-1 or FOCUS-2 algorithms

described in Section 4.6.[10] After finding a minimal-size subset of relevant features, the training examples are filtered to remove all irrelevant features. The filtered examples are then given to ID3 to construct a decision tree.

We consider three evaluation criteria: coverage, sample complexity, and error rate. The *coverage* of a learning algorithm was defined in Chapter 3 as the number of distinct concepts that can be learned from a training sample of size m for given accuracy and confidence parameters ϵ and δ. The *sample complexity* of an algorithm L for a set of concepts C is the smallest sample size sufficient to enable L to learn every concept in C for given values of ϵ and δ. Finally, the *error rate* for an algorithm on a given concept is measured as the probability that a randomly chosen example would be misclassified by the hypothesis output by the algorithm.

Since our objective is to evaluate the learning performance in domains where many of the available features are irrelevant to the target concept, we have specialized the above criteria in the following manner. First, concepts with $i+1$ relevant features are not counted in the coverage of an algorithm unless all concepts of i or fewer features are learned as well. If this condition is not included, there exist trivial algorithms that can attain high coverage while learning only very uninteresting concepts (see Section 3.8 for details). Second, for the sample complexity measurement, we choose C to be a set of concepts with only p or fewer relevant features where p is small compared to the number of the available features. Finally, in measuring the error rates of the three algorithms, the target concept is chosen randomly from those concepts defined only on a small subset of the available features.

One technical problem in performing our experiments is the immense amount of computation involved in the exact measurement of coverage and sample complexity when the number of features is large. Therefore, we employed the two techniques described in Section 3.5 to reduce the computational costs of these measurements:

[10] Note that the FOCUS-1 and FOCUS-2 algorithms are equivalent in terms of their input-output characteristics. They differ only in time complexity.

- First, we exploited the fact that each of the three algorithms is symmetric with respect to permutations and/or negations of the features.

- Second, we resorted to statistical approximation in deciding whether an algorithm successfully learns a given concept by running the algorithm on a large number of randomly-chosen samples rather than the set of all possible samples. However, unlike Subsection 3.5.1, we considered only two outcomes: A concept was considered learned by the algorithm if the algorithm returns an ϵ-close hypothesis for a fraction at least $(1 - \delta)$ of the training samples. Otherwise, the concept was considered unlearned. The number of training samples tested per concept was 10,000 or 100,000 depending on the experiment. These numbers were observed to be large enough to yield reliable decisions.

5.3.1 Experiment 1: Coverage

In this experiment, we measured the MIN-FEATURES-based coverage of each of the three algorithms. For each algorithm, we counted the number of concepts learned *in order* as a function of the size m of the training sample. The learning parameters were $n = 8$, $\epsilon = 0.1$, $\delta = 0.1$ and m varying. The number of samples tested per concept was 10,000. Figure 19 shows the result of this experiment.

5.3.2 Experiment 2: Sample Complexity

In this experiment, we estimated the sample size needed to learn all concepts having 3 or fewer relevant features out of a total of 8, 10, or 12 available features. As before, ϵ and δ were 0.1.

The experiment was conducted by testing each algorithm against every concept in the collection for some sample size m. The value of m is then increased or decreased depending on whether the algorithm fails or succeeds in learning all the concepts in the collection. We started this experiment by running the algorithms

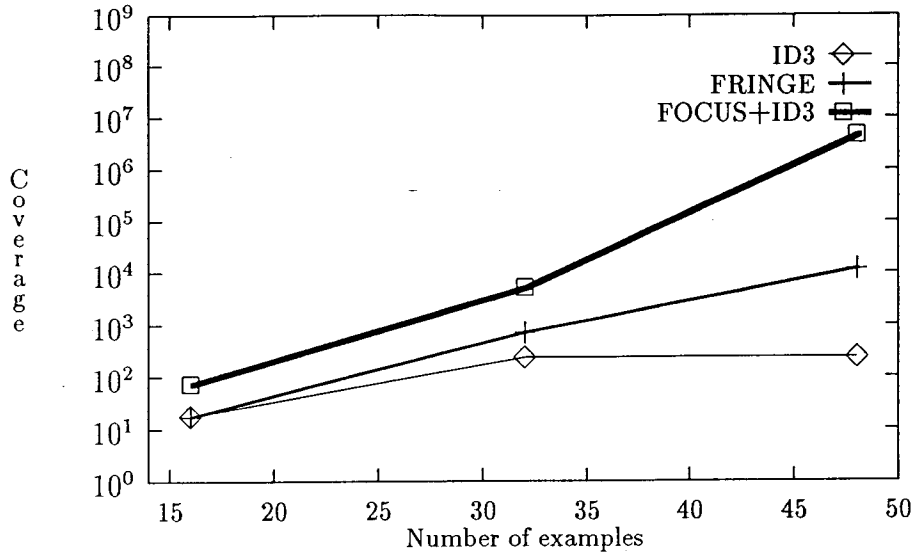


Figure 19. Coverage of the three algorithms for $n = 8$ and $\epsilon = \delta = 0.1$.

Table 4. Sample complexity results.

No. features	ID3	FRINGE	FOCUS+ID3
8	194	52	34
10	648	72	40
12	2236	94	42

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 testing 100,000 training samples per concept.

The results of this experiment are shown in Table 4.

5.3.3 Experiment 3: Error Rates

In the previous experiments we were looking at the "worst case" performance of the learning algorithms. That is, given a reasonable sample size, an algorithm may learn all the concepts under consideration with the exception of a few that require

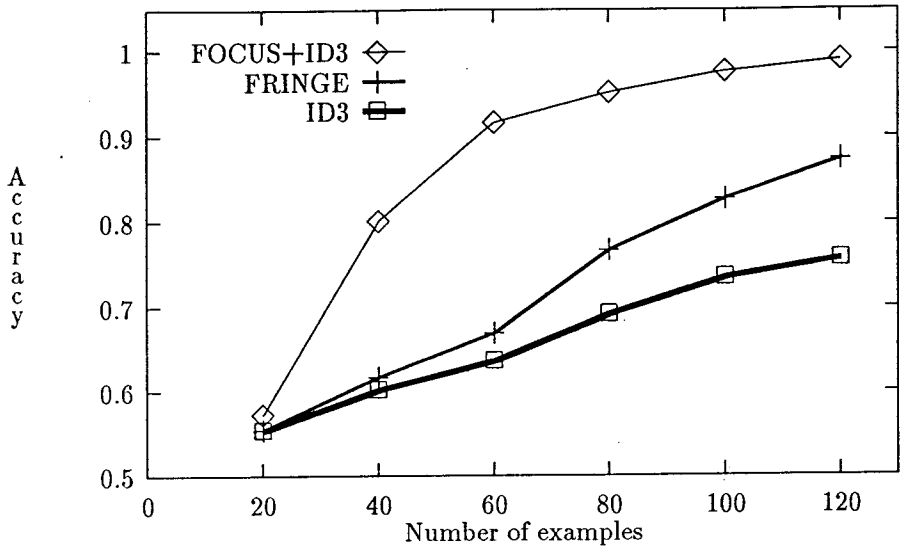


Figure 20. Learning curve for the randomly chosen concept $f(x_1, \ldots, x_{16}) = x_1x_2x_3\bar{x_4} \vee x_1x_2x_3x_4\bar{x_5} \vee x_1x_2\bar{x_3}x_4x_5 \vee x_1\bar{x_2}x_3 \vee x_1\bar{x_2}\bar{x_3}\bar{x_4} \vee \bar{x_1}x_2x_3x_4x_5 \vee \bar{x_1}\bar{x_2}x_3x_4x_5 \vee \bar{x_1}\bar{x_2}x_3\bar{x_4} \vee \bar{x_1}\bar{x_3}x_4x_5 \vee \bar{x_1}\bar{x_3}\bar{x_4}\bar{x_5}$ which has 5 relevant features out of 16.

a substantial increment in the sample size. Such an algorithm could exhibit poor performance in the previous two experiments. The purpose of this experiment is to perform a kind of "average case" comparison between the three algorithms. The procedure is to plot the learning curve (sample size vs. accuracy) for randomly chosen concepts with few relevant features.

We randomly selected 100 concepts each having at most 5 relevant features out of 16 available features. For each of these concepts, we measured the accuracy of the hypotheses returned by the three algorithms while successively increasing the sample size. For each value of m, the accuracy rate was averaged over 100 randomly chosen samples.

Figure 20 shows a pattern typical of all learning curves that we observed.

5.3.4 Experiment 4: Random Irrelevant Features

The goal of this experiment was to see how the three algorithms are influenced by the introduction of additional (irrelevant) features whose values are assigned

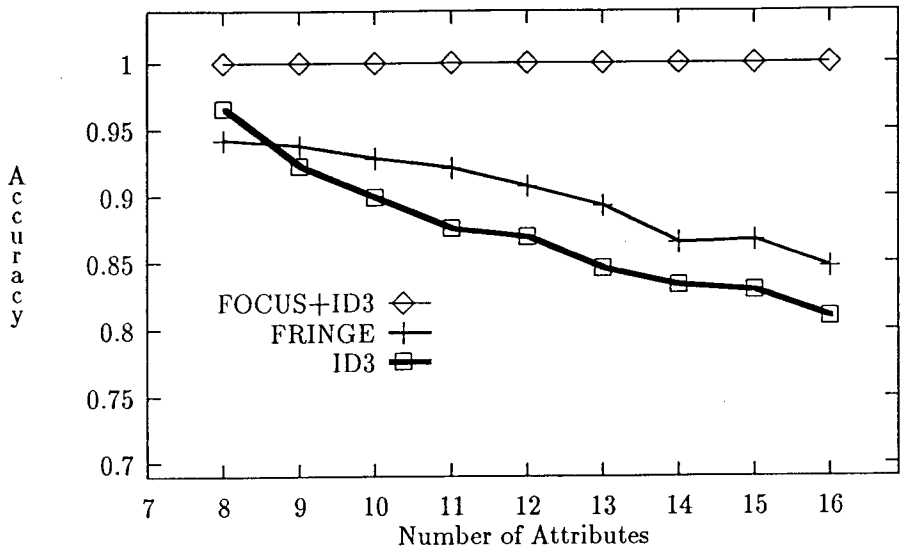


Figure 21. Accuracy of the three algorithms on a randomly chosen concept-sample pair as more irrelevant random features are introduced. The sample size was 100.

at random. For this purpose, we choose a concept-sample pair at random, and measure the accuracy of the hypothesis returned by each algorithm while adding more and more irrelevant features to the sample. The concepts chosen have 5 relevant features out of 8. The sample size was chosen such that all three algorithms are reasonably accurate when tested using only the 8 starting features. A sample of this size is chosen randomly and then augmented by successively adding random features to bring the total number of features up to $n = 16$. For each value of n, the accuracy is averaged over 100 runs.

This experiment was repeated on 100 concept-sample pairs. A typical result of these runs is shown in Figure 21. A complete list of the results for all the 100 tested concept-sample pairs can be found in the appendix.

5.3.5 Discussion

These experiments show conclusively that the biases implemented by ID3 and FRINGE, though they may be interesting and appropriate in many domains, are

not good approximations of the MIN-FEATURES bias. The final experiment shows this most directly. Using the MIN-FEATURES bias, FOCUS+ID3 maintains a constant, high level of performance as the number of irrelevant features is increased. In contrast, the performance of ID3 and FRINGE steadily degrades. This occurs because ID3 and FRINGE are proposing hypotheses that involve many extra features (or perhaps different features) than those used by FOCUS+ID3.

This also explains the results of Experiments 1, 2, and 3. In Experiment 2, we see that many more training examples are required for ID3 and FRINGE to find good hypotheses. These extra training examples are needed to force the algorithms to discard the irrelevant features. This also means that, for a fixed sample size, ID3 and FRINGE can learn many fewer concepts (with respect to the MIN-FEATURES learning behavior), as shown in Experiment 1. Experiment 3 shows that if the MIN-FEATURES bias is appropriate, then FOCUS+ID3 will give much better generalization performance than either ID3 or FRINGE.

As a final remark, it should be noted that both the average and worst-case performance of FRINGE were much better than that of ID3.

5.4 Heuristics for the MIN-FEATURES Problem

The experiments of the previous section demonstrate the necessity of designing efficient learning algorithms that perform well in domains where many irrelevant features are present since, as these experiments show, ID3 and FRINGE perform quite poorly in such domains.

In this section, we describe and evaluate three algorithms that approximate the MIN-FEATURES bias. 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 resulting solution is, therefore, correct but not necessarily minimal. The only difference between the three algorithms is the criterion used in selecting the best

feature in each iteration.

5.4.1 The Mutual-Information-Greedy Algorithm

Suppose we are looking at a training sample S and a set Q of features. There are $2^{|Q|}$ different assignments to the features in 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 us denote by p_i the number of positive examples in the i-th group, and by n_i the number of negative examples in that group.

Now, if the examples in each group are either all positive or all negative (i.e., if for every i, $p_i \cdot n_i = 0$), then this means that, for any example in the training sample, the values of the features in Q determine the class of the example. In other words, Q leaves no *uncertainty* about the classification of the examples of the training sample, and therefore, Q must be sufficient to form a hypothesis consistent with the sample.

However, if any of the $2^{|Q|}$ groups contains a mixture of positive and negative examples, then Q is insufficient, since it does not remove all the uncertainty. A well-known method to measure such uncertainty is the use of the *entropy* function [Abramson 63]. In this case, the amount of uncertainty left by a set Q is given by

$$Entropy(Q) = -\sum_{i=0}^{2^{|Q|}-1} \frac{p_i + n_i}{|Sample|} \left[\frac{p_i}{p_i + n_i} \log_2 \frac{p_i}{p_i + n_i} + \frac{n_i}{p_i + n_i} \log_2 \frac{n_i}{p_i + n_i} \right] \quad (5.15)$$

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

This measure of uncertainty can be used to determine the most promising feature as follows. The relevance of a feature x_i to the target concept is assessed by the amount of reduction in uncertainty obtained by adding x_i to the current partial solution. Thus, if the current set of features is Q, then the score of x_i is computed as $Entropy(Q \cup \{x_i\})$. As the best feature, we choose x_i as the feature

Algorithm: Mutual-Information-Greedy(*Sample*)

1. $Q = \phi$.
2. Repeat until $Entropy(Q) = 0$:
   2.1. For each feature x_i, let $score_{x_i} = Entropy(Q \cup \{x_i\})$.
   2.2. Let *best* be the feature with the lowest score.
   2.3. $Q = Q \cup \{$ *best* $\}$.

end.

Figure 22. The Mutual-Information-Greedy algorithm—An abstract version.

that minimizes this quantity.

The above criterion for selecting the most promising feature is used in our first heuristic which we call the "Mutual-Information-Greedy" algorithm. An abstract description of this algorithm is given in Figure 22. In this implementation, the entropy function for a set Q can be computed directly as given by equation (5.15), by splitting the training sample based on Q into the $2^{|Q|}$ groups and then computing the summation. However, 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 $Entropy(Q \cup \{x_i\})$ 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 (5.15) 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.

A more detailed implementation of the Mutual-Information-Greedy algorithm that is based on these efficiency considerations is given in Figure 23. 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 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*$(Q \cup \{x_i\})$ when the loop of Step 2.2.2 terminates.

5.4.2 The Simple-Greedy Algorithm

The relation between the MIN-FEATURES problem and the MINIMUM-SET-COVER problem has been discussed in detail in Subsection 4.5.3. In Subsection 4.5.2, we have defined a *conflict* generated from a pair of negative and positive examples as an n-bit vector in which the i-th bit is 1 if and only if the feature x_i has non-matching values in the two examples. If the i-th bit of a conflict is 1, then the conflict is said to be *explained* by the feature x_i. According to Corollary 4.1, a set of features is sufficient to construct an hypothesis that is consistent with the training sample if and only every conflict generated from the sample must be explained by

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-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* $=$ *Score* $- \frac{p_0+n_0}{|\text{Sample}|}[e_0 + e_1]$.

2.2.3 If *Score* $<$ *Best-Score* then

2.2.2.1 *Best-Feature* $= x_i$.

2.2.2.2 *Best-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.

Figure 23. The Mutual-Information-Greedy algorithm—A detailed version.

Algorithm: Simple-Greedy(*Sample*)

1. $Q = \phi$.
2. Let A be the set of all conflicts generated from *Sample*.
3. Repeat until A is empty:
   - 3.1. For each feature x_i, let $score_{x_i}$ = the number of conflicts explained by x_i.
   - 3.2. Let *best* be the feature with the highest score.
   - 3.3. $Q = Q \cup \{ best \}$.
   - 3.4. Remove from A all the conflicts explained by *best*.

end Simple-Greedy.

Figure 24. The Simple-Greedy algorithm—An abstract version.

some feature in the set. The equivalence between the MIN-FEATURES and the MINIMUM-SET-COVER problems was drawn by letting "explaining a conflict by a feature" in the first problem correspond to "covering an element by one of the given subsets" in the second problem.

Our second heuristic for approximating the MIN-FEATURES bias originates from a popular greedy heuristic that approximates the MINIMUM-SET-COVER. This heuristic chooses the subset that covers the largest number of elements that are not yet covered by the subsets chosen so far. For the MIN-FEATURES problem, this corresponds to choosing the feature that explains the maximum number of conflicts that are not explained by the features in the current partial solution. Figure 24 describes an abstract implementation of this heuristic which we call the "Simple-Greedy" algorithm.

For the sake of computational efficiency, it should be noted that computing the score of a feature in this algorithm does not require the knowledge of the set of conflicts itself 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 explained by some feature in Q, and thus, the conflicts that are *yet to be explained* 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 unexplained after adding x_j to Q. This can be computed as follows:

$$score_{x_j} = \sum_{i=0}^{2^{|Q|}-1} p_i^{x_j=0} \cdot n_i^{x_j=0} + p_i^{x_j=1} \cdot n_i^{x_j=1} \tag{5.16}$$

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

A detailed implementation of the Simple-Greedy algorithm based on the above ideas is given in Figure 25. Note that this algorithm is identical to the Mutual-Information-Greedy algorithm given in the previous subsection except for the score calculation.

5.4.3 The Weighted-Greedy Algorithm

In the Simple-Greedy algorithm, every conflict contributes a unit increment to the score of each feature that explains the conflict. Our third approximation algorithm,

Algorithm: Simple-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-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. *Score* $=$ *Score* $+p_0 n_0 + p_1 n_1$.

     - 2.2.3 If *Score* $<$ *Best-Score* then
       - 2.2.2.1 *Best-Feature* $= x_i$.
       - 2.2.2.2 *Best-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.

end.

Figure 25. The Simple-Greedy algorithm—A detailed version.

which we call the "Weighted-Greedy" algorithm, works similarly except that the increment per conflict is not fixed, but depends on the number of features that commonly explain the conflict. The idea is that a feature that explains a conflict that is not explained by many other features should get more credit, and vice versa. For example, if $a_1 = \langle 00110000 \rangle$ and $a_2 = \langle 11110011 \rangle$ are two conflicts, then the contribution of a_1 to the scores of the features x_3 and x_4 should be higher than the contribution of a_2 to the scores of the features $\{x_1, x_2, x_3, x_4, x_7, x_8\}$. This is because a_1 is explained by only 2 features which makes it *harder* to explain than a_2 which is explained by as many as 6 features.

In the "Weighted-Greedy" algorithm, we let the increment caused by a conflict y to the scores of the features that explain y be

$$\frac{1}{W_y - 1},$$

where W_y is the number of features that explain y. Under this heuristic, when a feature x_i explains a conflict a, the contribution of a to the score of x_i is inversely proportional to the number of *other* features that explain a. If only a few other features explain a then x_i receives high credit for explaining a. In the extreme case where a is exclusively explained by x_i, the score of x_i becomes ∞. This causes the feature to be included in the solution with certainty (which is, of course, the right action).

The complete details of the Weighted-Greedy algorithm are given in Figure 26. Unfortunately, unlike the Simple-Greedy algorithm, there seems to be no obvious way to implement this algorithm without actually constructing the set of all conflicts out of the given training sample. This causes the algorithm to run in time quadratic in the size of the training sample, which makes it computationally more expensive than the Mutual-Information-Greedy and the Simple-Greedy algorithms.

Algorithm: Weighted-Greedy(*Sample*)

1. $Q = \phi$.
2. $V = \{x_1, x_2, x_3, \cdots, x_n\}$.
3. Let A be the set of all conflicts generated from *Sample*
4. For each conflict $y \in A$
   - 4.1 Let W_y be the number of features that explain y.
   - 4.2 Let $weight_y = \frac{1}{W_y - 1}$
5. Repeat
   - 5.1 For each $x_i \in V$, let $score_{x_i} = 0$.
   - 5.2 For each $y \in A$
     - 5.2.1 For each $x_i \in V$ that explains y, let
       $$score_{x_i} = score_{x_i} + weight_y$$
   - 5.3 Let *Best-Feature* be the feature with the highest score.
   - 5.4 $Q = Q \bigcup \{Best\text{-}Feature\}$.
   - 5.5 Remove *Best-Feature* from V.
   - 5.6 Remove from A all the conflicts explained by *Best-Feature*.

   until A is empty.
6. Return Q.

end.

Figure 26. The Weighted-Greedy algorithm.

5.4.4 Experimental Evaluation of the Three Heuristics

To see how well each of our three heuristics approximates the MIN-FEATURES bias, we have repeated the experiments of Subsections 5.3.2 through 5.3.4 on these algorithms. Just as with the FOCUS algorithms, these heuristics only return a subset of features that is sufficient to construct an hypothesis consistent with the training sample. In the experiments reported here, the hypothesis is constructed by first filtering the training sample to remove all the features that are not in the selected set of features. The filtered examples are then given to ID3 to construct a decision tree.

The results of the sample complexity estimation experiment (Subsection 5.3.2) are shown in Figure 27. These results show that all the three heuristics improved the performance of ID3 and that the performance of Weighted-Greedy algorithm was very close to that of the FOCUS algorithms. This was followed by FRINGE, Simple-Greedy, and then Mutual-Information-Greedy.

A typical pattern of the results of the error rate estimation experiment (Subsection 5.3.3) is shown in Figure 28. As a way to combine the results for all 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 29.

The results of this experiment show again that the learning curve of the Weighted-Greedy algorithm is the closest to that of FOCUS+ID3. In contrast, both ID3 and FRINGE performed quite badly, although FRINGE performed better than ID3 in general. The performance of the Mutual-Information-Greedy and the Simple-Greedy algorithms were inferior to that of Weighted-Greedy, but considerably better than ID3 and FRINGE.

The relative performance of each of the three heuristics in the experiment of Subsection 5.3.4 (measuring the influence of random irrelevant features) varied depending on the concept-sample pair being tested. Figures 30 and 31 show the

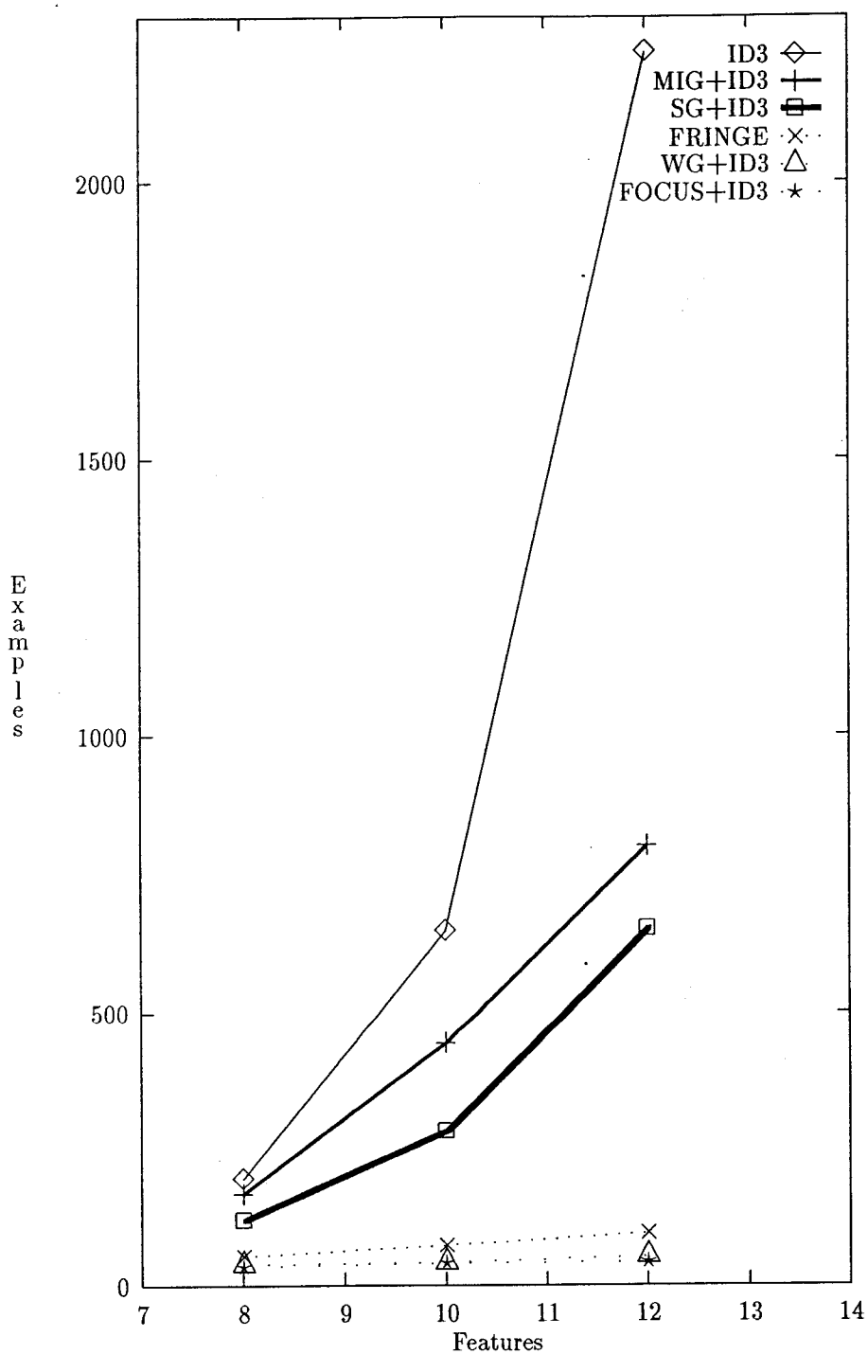


Figure 27. The number of examples needed to learn all the concepts with 3 relevant features out of 8, 10 and 12 available features. The names of the Mutual-Information-Greedy, Simple-Greedy, and Weighted-Greedy algorithms are abbreviated as MIG, SG, and WG, respectively.

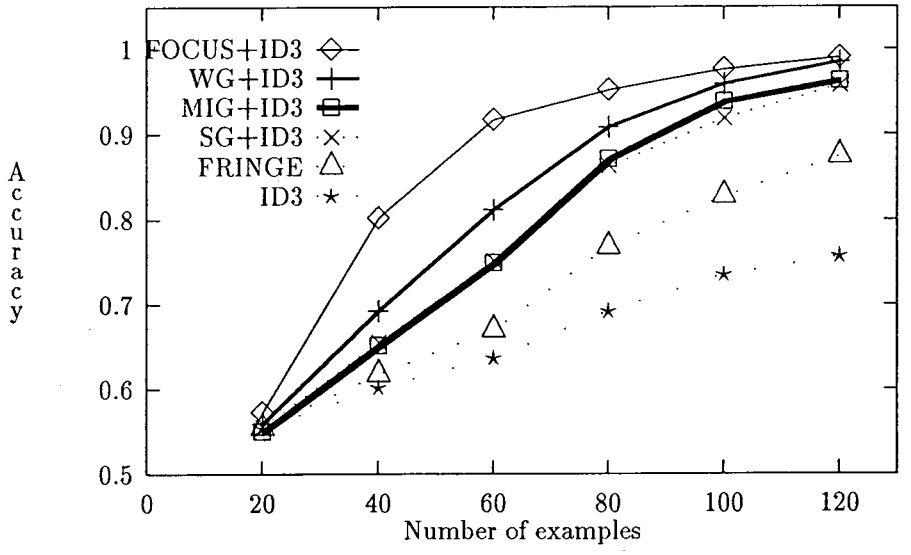


Figure 28. Learning curve for the randomly chosen concept $f(x_1, \ldots, x_{16}) = x_1x_2x_3\bar{x_4} \vee x_1x_2x_3x_4\bar{x_5} \vee x_1x_2\bar{x_3}x_4x_5 \vee x_1\bar{x_2}x_3 \vee x_1\bar{x_2}\bar{x_3}\bar{x_4} \vee \bar{x_1}x_2x_3x_4x_5 \vee \bar{x_1}\bar{x_2}x_3x_4x_5 \vee \bar{x_1}\bar{x_2}x_3\bar{x_4} \vee \bar{x_1}\bar{x_3}x_4x_5 \vee \bar{x_1}\bar{x_3}\bar{x_4}\bar{x_5}$ which has 5 relevant features out of 16. The names of the Mutual-Information-Greedy, Simple-Greedy, and Weighted-Greedy algorithms are abbreviated as MIG, SG, and WG, respectively.

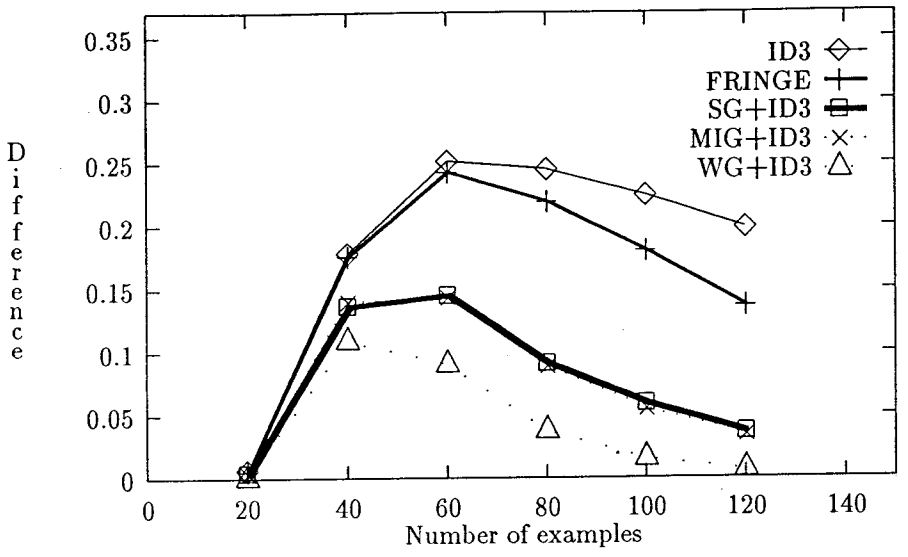


Figure 29. The difference between the accuracy of FOCUS and the accuracy of the other algorithms averaged over all the randomly chosen 100 target concepts. The names of the Mutual-Information-Greedy, Simple-Greedy, and Weighted-Greedy algorithms are abbreviated as MIG, SG, and WG, respectively.

results for two typical cases. A complete list of the results for all the 100 tested concept-sample pairs can be found in the appendix.

Figure 30 illustrates the results for roughly one third of the concept-sample pairs we have tested. For these cases, ID3 and FRINGE have shown high sensitivity to the introduction of irrelevant features. The accuracy performance of these two algorithms continued to degrade as the total number of features increased. In contrast, the performance of all three approximation algorithms remained reasonably stable.

With a few exceptions, Figure 31 was more-or-less the pattern for the rest of the cases. As this figure shows, all three approximation algorithms were affected by the introduction of the irrelevant features. However, the performance of these algorithms was almost always better than ID3 and FRINGE. Moreover, the Weighted-Greedy algorithm was overall the least affected among the three heuristics.

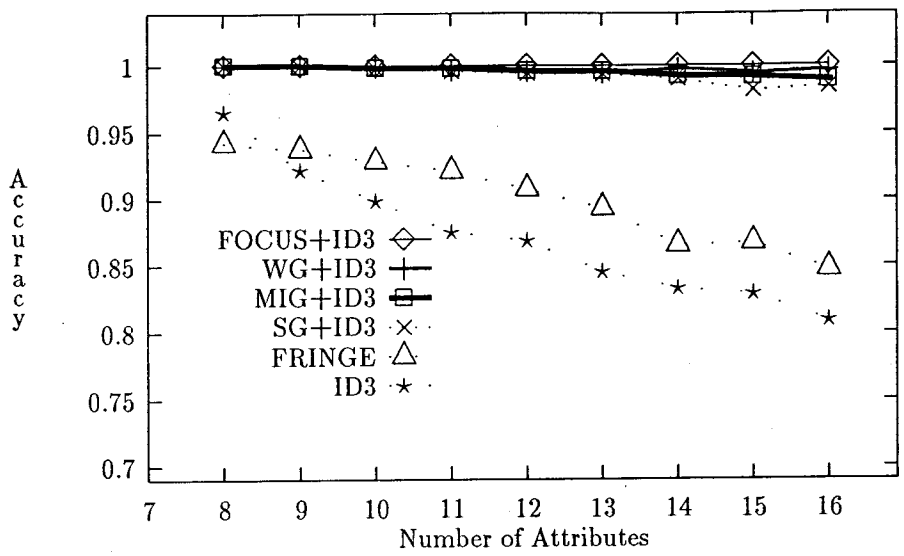


Figure 30. Accuracy of the six learning algorithms on a randomly chosen concept-sample pair as more irrelevant random features are introduced. The sample size was 100. The names of the Mutual-Information-Greedy, Simple-Greedy, and Weighted-Greedy algorithms are abbreviated as MIG, SG, and WG, respectively.

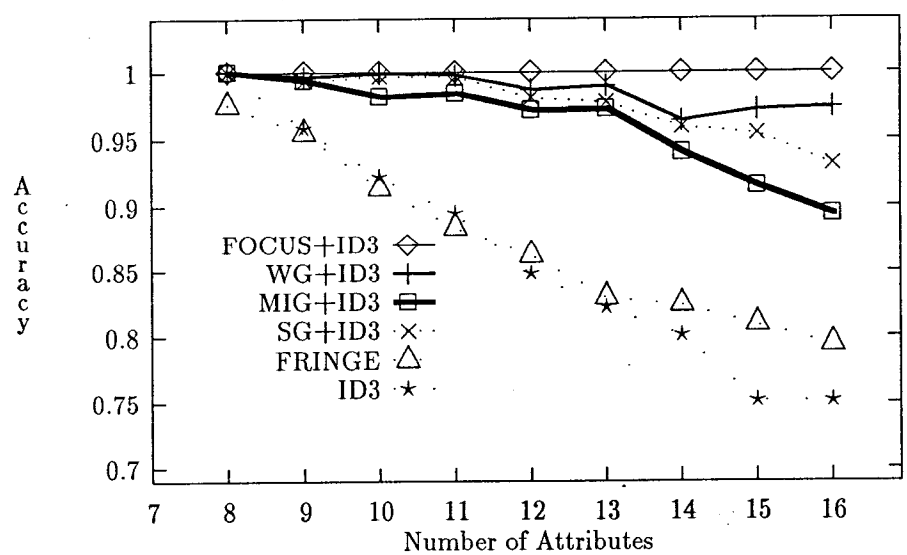


Figure 31. Accuracy of the six learning algorithms on a randomly chosen concept-sample pair as more irrelevant random features are introduced. The sample size was 100. The names of the Mutual-Information-Greedy, Simple-Greedy, and Weighted-Greedy algorithms are abbreviated as MIG, SG, and WG, respectively.

5.4.5 Discussion

The experiments conducted here were designed to demonstrate the performance of the algorithms in two ways:

- The worst-case performance: This is exhibited by the sample complexity estimation experiment in which we look at the worst-case behavior of each algorithm over a collection of concepts. For an algorithm to have good worst-case performance, it should perform well on *every* concept in the collection. The presence of just one concept on which the algorithm behaves poorly is sufficient to conclude that the algorithm has bad worst-case performance.

- The average-case performance: This is exhibited by the error rate estimation experiments. Here we look at the performance of each algorithm on concepts that are selected randomly from the collection of concepts being considered. A good algorithm in this case is the one that behaves well on the majority of the concepts, even if it may perform badly on a few concepts.

The performance of the algorithms we have tested can be summarized as follows:

1. Weighted-Greedy was the most successful heuristic for approximating the MIN-FEATURES bias both in the worst and average cases.

2. Mutual-Information-Greedy and Simple-Greedy behaved quite similarly to each other. These algorithms maintained a reasonable average-case performance, but exhibited a rather bad worst-case performance.

3. FRINGE showed a poor average-case performance, but its worst-case performance was next to that of Weighted-Greedy and substantially better than that of Mutual-Information-Greedy and Simple-Greedy.

4. Finally, ID3 (without preprocessing of the training examples) exhibited very poor performance for both the average and worst cases.

5.5 Execution Time Comparisons

The declared objective of this chapter was the design of *efficient* learning algorithms whose performance is as close to the FOCUS algorithms as possible in learning domains with many irrelevant features. Now that we have shown (through the experimental work of the previous section) that the Weighted-Greedy algorithm provides an excellent approximation of the MIN-FEATURES bias, the next step is to confirm that Weighted-Greedy is computationally more efficient than FOCUS-2. We report here the results of an experiment in which we compare the execution time of the Weighted-Greedy algorithm to that FOCUS-2. For the sake of comparison between FOCUS-2 and FOCUS-1, we also include the latter algorithm in our experiment.

We have implemented the three algorithms in the C programming language. In particular, we paid more attention to optimizing our implementation of FOCUS-1 in order to avoid any possibility for biased results.

The experiments were conducted by running the three algorithms on training samples of target concepts that have only a few relevant features out of many available. The training samples were drawn with replacement under the uniform distribution.

We found that the relative running time 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. Nevertheless, once the problem size was made reasonably large, the relative performance started to follow a consistent and clear trend. This trend is illustrated by Table 5 where we give the result for a target concept with 9 relevant features out of 25 available features. In this table, the number of training examples was varied from 100 to 500. The reported numbers of the table were 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.

Table 5. 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:					
Suff.Tests	442189.4	1329330.5	2908068.9	2665664.0	2695393.0
Time (secs)	9.74	55.1	211.1	191.7	186.1
FOCUS-2:					
Suff.Tests	10593.1	9785.3	11339.5	8768.9	5582.3
Time (secs)	2.0	7.1	20.5	31.9	32.5
WG:					
Suff.Tests	8.5	10.1	10.7	10.5	11.0
Time (secs)	0.16	0.86	2.31	4.3	7.0

5.6 Summary

The goal of this chapter was to come up with computationally efficient algorithms that learn well in the presence of irrelevant features. We proposed three algorithms: Weighted-Greedy, Mutual-Information-Greedy, and Simple-Greedy. The performance of these algorithms in addition to ID3 and FRINGE, was empirically evaluated through carefully designed experiments in which each of the new algorithms was used to preprocess the training data to remove irrelevant features.

The most important lesson of this chapter is that—contrary to expectations—ID3 and FRINGE do not implement a good approximation of the MIN-FEATURES bias. As a consequence, ID3 and FRINGE do not perform nearly as well as the FOCUS algorithms in problems where many irrelevant features are present. These results suggest that one should not rely on ID3 or FRINGE to filter out irrelevant features. Instead, some technique should be employed to eliminate irrelevant

features and focus ID3 and FRINGE on the relevant ones.

All of the three heuristics introduced in this chapter were found to be helpful in this filtering task. In particular, the Weighted-Greedy algorithm exhibited excellent performance that closely matches what is obtained by the exact MIN-FEATURES bias. Empirical comparison confirmed that this algorithm is substantially more efficient in terms of computational costs than the FOCUS-1 and FOCUS-2 algorithms which provide exact implementations of the MIN-FEATURES bias. We recommend that, in applications where many irrelevant features are present, the Weighted-Greedy algorithm should be applied to preprocess the training sample before invoking a decision-tree learning algorithm, such as ID3.

Chapter 6

Conclusion and Future Work

In this chapter, we summarize the contribution of our research and state some of the related problems that we have left open.

We began this thesis by investigating what can be achieved if we take the number of training examples to be fixed and seek learning algorithms that maximize the number of concepts that can be learned with those examples (i.e., maximize the "coverage"). In answering this question, we proved upper bounds on the coverage that can be achieved by any learning algorithm. We also introduced corresponding lower bounds by analyzing the coverage of a family of learning algorithms which we called the "Balls" algorithms. Furthermore, we empirically evaluated the coverage of some of the existing practical learning algorithms. Our findings can be summarized as follows:

- The Balls approach for designing learning algorithms leads to a very high coverage. Namely, we have proven that one of these algorithms (Multi-Balls) requires a number of examples that is within only a constant factor from that required by an optimal algorithm to achieve a given coverage.

- We have shown that the coverage of popular learning algorithms such as ID3 is disappointingly poor compared to the coverage of algorithms designed under the Balls approach.

- In particular, we have shown that there exist learning algorithms emerging from the Balls approach that learn an immense number of concepts (compared to practical learning algorithms such as ID3) but are rather trivial and practically uninteresting.

The above results show that stating the desired learning behavior as merely to maximize the *number* of learned concepts is not sufficient to yield practically useful learning algorithms. In addition to this goal, one should specify *a priori* (as a component of the desired learning behavior) some basis to differentiate the relative *importance* of each concept in the space of concepts to be learned. Coverage should then be redefined according to this basis.

In other words, the design of learning algorithms should be based on a set of "background assumptions" that state what concepts are more "valuable" to learn than others. This defines the desired learning behavior which serves as the "specifications" of the design task and provides the measure for evaluating learning algorithms.

Optimality in this setting is defined only with respect to the declared background assumptions. This means that a learning algorithm with good performance under specific assumptions is expected to perform well in a practical domain only if these background assumptions are satisfied in that domain. Naturally, this suggests that studies should concentrate on background assumptions that are practically well-motivated—i.e, assumptions that are satisfied in a wide range of real applications.

As an example of such practically justified background assumptions, we investigated, in the second part of this thesis, the problem of learning from data that contain many irrelevant features. In this case, we (justifiably) assume that the target concept is one of those defined on only a small subset of the features present in the training data. For such situations, we define the concept complexity measure such that concepts defined on a larger number of features are considered

more complex. Under this measure, *simple* concepts carry more importance than complex ones, and thus, the desired learning behavior here is to learn, from a sample of a given size, as many simple concepts as possible.

In addressing the problem of learning in the presence of many irrelevant features, we have introduced and studied the MIN-FEATURES bias which states that if two functions are consistent with the training examples, then the function that involves fewer features is to be preferred.

Our study of the MIN-FEATURES bias included the following:

- An upper bound on the sample complexity of algorithms that implement the MIN-FEATURES bias.
- A lower bound on the sample complexity of *any* algorithm that attempts to follow the desired learning behavior as defined above. This lower bound is within a constant factor of our upper bound on the sample complexity of algorithms that implement the MIN-FEATURES bias.
- The FOCUS-1 and FOCUS-2 algorithms for exact implementation of the MIN-FEATURES bias.
- The Mutual-Information-Greedy, Simple-Greedy and Weighted-Greedy algorithms for approximating the MIN-FEATURES bias.
- Experimental results that compare the performance of the above algorithms, as well as the two popular algorithms ID3 and FRINGE, in learning problems where many irrelevant features are present.

Our results show that the presence of many irrelevant features does not make the learning problem substantially harder in terms of the number of training examples required to learn. Of course, this is true only if an appropriate mechanism is followed to detect and eliminate the irrelevant features from consideration. Particularly, our experimental results showed that—contrary to expectations—ID3 and

FRINGE handle the presence of irrelevant features quite poorly. These results suggest that, before running these algorithms, one should preprocess the training examples to eliminate irrelevant features and focus these algorithms on the relevant ones. For this filtering task, one may use the FOCUS-2 algorithm or the Weighted-Greedy algorithm depending on the availability of computational resources.

Many problems were left open throughout the course of our research. We conclude this thesis with a list of such problems:

1. In light of Theorem 3.3, the general upper bound on coverage given by Theorem 3.1 does not appear to be tight. An interesting open problem is to tighten this bound.

2. Even with the computational cost reduction techniques we introduced in Section 3.5, our experiments on evaluating the coverage of existing learning algorithms were limited to the space of concepts defined over only 5 features. Going beyond that was not possible because of the immense amount of computation involved in the experiments. An important future task is to come up with reasonable methods to find the set of concepts which are learned by a certain algorithm with respect to given learning parameters. These methods can be analytical, empirical, or a combination of these. Notice that the reduction techniques we presented in Section 3.5 are algorithm-independent—they only require that the learning algorithm be symmetric, which is usually the case for most learning algorithms. Although this is a desired feature, one may also consider techniques that are tailored to a particular algorithm or a family of algorithms that share certain properties in common. As discussed in Chapter 3, the desired methods should help in finding out *what* concepts are learned and not only the *number* of these concepts.

3. An interesting question that we have left open in this thesis is the possibility of implementing the MIN-FEATURES bias in time polynomial in the learning parameters. One way to approach this question is to find out whether

it is possible to solve the MINIMUM-SET-COVER in time polynomial in l, v and 2^k where l is the number of elements to be covered, v the number available subsets, and k the size of the minimum cover of the given problem instance. The connection between this problem and the MIN-FEATURES bias is discussed in detail in Subsection 4.5.3.

4. The answer to the above question may be negative. That is, it may turn out that the exact MIN-FEATURES bias cannot be implemented in polynomial time. A more general approach is to attempt to show whether there exists *any* algorithm (not necessarily implementing the MIN-FEATURES bias) that learns concepts of complexity s with time and sample complexities bounded by some polynomials in $n, s, \frac{1}{\epsilon}$ and $\frac{1}{\delta}$. One way to attack this question is to analyze the sample complexity of heuristics that approximate the MIN-FEATURES bias such as those given in Chapter 5. The running time of these heuristics can be easily shown to be polynomial. Thus, it suffices to show that the sample complexity is also polynomial. Again, it may be useful here to view the problem as MINIMUM-SET-COVER and attempt to find good approximation heuristics for this latter problem. For example, if we can construct an algorithm that finds a solution to the MINIMUM-SET-COVER problem that is within a constant factor form the minimum cover, then we can immediately turn this into an algorithm that ω-approximates the MIN-FEATURES bias (as defined in page 128) for a constant ω. According to Theorem 5.1, this suffices to show that the sample complexity of this algorithm is polynomial. It should be noted, however, that there is no known algorithm that provably approximates the MINIMUM-SET-COVER problem within a constant factor. The best known approximation is within a factor $O(\log l)$, where l is the number of elements to be covered [Johnson 74], [Chavtal 79]. This is achieved by the simple heuristic which chooses each time the subset that covers the maximum number of elements that are not

yet covered. This is the same strategy implemented by the Simple-Greedy algorithm we introduced in Subsection 5.4.2 as an approximation algorithm for the MIN-FEATURES bias. Thus, ω for this algorithm is $O(\log m)$, but this does not lead to polynomial sample complexity as desired.

5. The work reported in this thesis assumes noise-free training data. That is, we assume that examples in the training sample are never labeled with the wrong class. Since this is rarely the case in practice, this work can be extended by considering learning from training samples that may include misclassified examples. A direct way to deal with classification noise is to modify the algorithms given in Chapters 4 and 5 by relaxing the requirement of explaining *all* the conflicts generated from the training sample. That is, we search for a small set of features that may leave a certain percentage of the conflicts unexplained, 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 work.

6. The results reported in Chapter 5 of this thesis recommend the pre-processing of the training data to remove irrelevant features before running decision-tree learning algorithms. These results are already confirmed by the ongoing research on Text Categorization by Lewis and Ringuette [Lewis and Ringuette 92]. One more extension to our work is to apply our algorithms to various real-world learning problems in which many irrelevant features are present.

7. Finally, an important theoretical problem for future research is to further investigate *biased coverage* as defined in Subsection 3.8 for various kinds of background assumptions. This includes proving upper bounds on biased coverage and designing learning algorithms that have biased coverage as large as possible.

BIBLIOGRAPHY

[Abramson 63] Abramson, N. *Information Theory and Coding.* McGraw-Hill. 1963.

[Almuallim and Dietterich 90] Almuallim, H. and Dietterich, T. G. Coverage of Inductive Learning Algorithms. *Workshop on Computational Learning Theory and Natural Learning Systems*, Princeton, NJ. Sep 5-6, 1990.

[Almuallim and Dietterich 91] Almuallim, H. and Dietterich, T. G. Learning With Many Irrelevant Features. *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI-91)*, 547–552, 1991.

[Almuallim 91] Almuallim, H. 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, OR 97331-3202.

[Almuallim and Dietterich 92a] Almuallim, H. and Dietterich, T. G. Efficient Algorithms for Identifying Relevant Features. *Proceedings of the Canadian Conference on Artificial Intelligence*, 1992.

[Almuallim and Dietterich 92b] Almuallim, H. and Dietterich, T. G. On Learning More Concepts. *Proceedings of the Ninth International Conference on Machine Learning*, 1992.

[Almuallim and Dietterich 92c] Almuallim, H. and Dietterich, T. G. How Useful is Maximizing the Number of Learned Concepts?, To appear as a journal paper.

[Almuallim and Dietterich 92d] Almuallim, H. and Dietterich, T. G. Efficient Learning of Boolean Concepts in the Presence of Many Irrelevant Features, To appear as a journal paper.

[Angluin 88] Angluin, D. Queries and Concept Learning. *Machine Learning*, 2(2):312-342, 1988.

[Bakiri 91] Bakiri, G. Converting English Text to Speech: A Machine Learning Approach. Ph.D. Thesis. Oregon State University, 1991.

[Blumer *et al.* 87a] Blumer, A.; Ehrenfeucht, A.; Haussler, D.; and Warmuth, M. Learnability and the Vapnik-Chervonenkis Dimension, Technical Report UCSC-CRL-87-20, Department of Computer and Information Sciences, University of California, Santa Cruz, Nov. 1987. Also in *Journal of ACM*, 36(4):929–965, 1989.

[Blumer *et al.* 87b] Blumer, A.; Ehrenfeucht, A.; Haussler, D.; and Warmuth, M. Occam's Razor. *Information Processing Letters*, 24:377-380, 1987.

[Buntine 90] Buntine, W.L. A Theory of Learning Classification Rules. Ph.D. Thesis. The University of Technology, Sydney. 1990.

[Carter and Catlett 87] Carter, C. and Catlett, J. (1987) Assessing Credit Card Applications Using Machine Learning. *IEEE Expert 2(3)*, pp. 71–79, Fall 1987. IEEE Computer Society.

[Chavtal 79] Chavtal, V. A Greedy Heuristic for the Set Covering Problem. *Math. Oper. Res.* 4(3) pp. 233–235. 1979.

[Chernoff 52] Chernoff, H. A Measure of Asymptotic Efficiency for Tests of Hypothesis Based on the Sums of Observations. *Annals of Mathematical Statistics*, 23, pp. 493–509, 1952.

[COLT88] *Proceedings of the 1988 Workshop on Computational Learning Theory.* Haussler, D. and Pitt, L., Editors. Morgan Kaufmann Publishers, 1988.

[COLT89] *Proceedings of the Second Annual Workshop on Computational Learning Theory.* Rivest, R., Haussler, D. and Warmuth, M.K., Editors. Morgan Kaufmann Publishers, 1989.

[COLT90] *Proceedings of the Third Annual Workshop on Computational Learning Theory.* Fulk, M.A. and Case, J., Editors. Morgan Kaufmann Publishers, 1990.

[COLT91] *Proceedings of the Fourth Annual Workshop on Computational Learning Theory.*

Valiant, L.G. and Warmuth, M., Editors. Morgan Kaufmann Publishers, 1990.

[Dietterich 89] Dietterich, T. G. Limitations on inductive learning. *Proceedings of the Sixth International Conference on Machine Learning, 124–128. Ithaca,* Morgan Kaufmann, 1989.

[Ehrenfeucht *et al.* 88] Ehrenfeucht, A.; Haussler, D.; Kearns, M.; and Valiant, L.G. A General Lower Bound on the Number of Examples Needed for Learning. *Proceedings of the First Workshop on Computational Learning Theory*, 139–154, Boston, Morgan Kaufmann. 1988.

[Garey and Johnson 79] Garey, M. R. and Johnson D. S. *Computers and Intractability.* W.H. Freeman and Company, 1979.

[Haussler 88] Haussler, D. Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework. *Artificial Intelligence*, 36:177-221, 1988.

[Harrison 65] Harrison, M. *Introduction to Switching and Automata Theory.* McGraw Hill, 1965.

[Hoeffding 63] Hoeffding, W. Probability Inequalities for Sums of Bounded Random Variables. *Journal of The American Statistical Association,* 58, 13-3-, 1963.

[Ichino and Sklansky 84a] Ichino, M. and Sklansky, J. Optimum Feature Selection by Zero-One Integer Programming. *IEEE Trans. Sys. Man & Cyb.*, Vol. SMC-14, No. 5, 737–746, Sep 1984.

[Ichino and Sklansky 84b] Ichino, M. and Sklansky, J. Feature Selection for Linear Classifiers, *The Seventh International Conference on Pattern Recognition*, 124–127, 1984.

[Johnson 74] Johnson, D.S. Approximation Algorithms for Combinatorial Problems. *J. Comput. Syst. Sci.* (9) pp. 256–278. 1974.

[Kearns *et al.* 87] Kearns, M., Li. M, Pitt, L. and Valiant, L. On the Learnability of Boolean Formulae. *Proceedings of the 19th ACM Symposium on Theory of Computing*, 285-295, 1987.

[Kittler 80] Kittler, J. Computational Problems of Feature Selection Pertaining to Large Data Sets. *Pattern Recognition in Practice.* Gelsma, E.S. and Kanal, L.N. (eds.) North-Holland

Publishing Company, 405–414, 1980.

[Lewis and Ringuette 92] Lewis, D.D. and Ringuette, M. Text Categorization by Inductive Learning. To appear.

[Littlestone 88] Littlestone, N. Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm. *Machine Learning,* 2:285–318, 1988.

[MacWilliams and Sloane 77] MacWilliams, F.J. and Sloane N.J.A. *The Theory of Error-Correcting Codes.* North-Holland Publishing Company. 1977.

[Mingers 89] Mingers, J. An Empirical Comparision of Selection Measures for Desicion-Tree Induction. *Machine Learning,* 3 (4), 319-342, 1989.

[Mitchell 80] Mitchell, T. The Need for Bias in Learning Generaliztions. Technical Report. CBM-TR 5-110, Rutgers University, New Jersey, 1980.

[Morgera 86] Morgera, S.D. Computational Complexity and VLSI Implementation of an Optimal Feature Selection Strategy. *Pattern Recognition In Practice II,* Gelsma, E.S. and Kanal, L.N. (eds.) Elsevier Science Publishers B.V. (North-Holland), 389–400, 1986.

[Mucciardi and Gose 71] Mucciardi, A.N. and Gose, E.E. A Comparison of Seven Techniques for Choosing Subsets of Pattern Recognition Properties, *IEEE Trans. Computers, Vol. C-20,* No. 9, 1023–1031, Sep 1971.

[Narendra and Fukunaga 77] Narendra, P.M. and Fukunaga, K. A Branch and Bound Algorithm for Feature Subset Selection. *IEEE Trans. Computers,* Vol. C-26, No. 9, 917–922, 1977.

[Natarajan 87] Natarajan, B. On Learning Boolean Functions. *Proceedings of the 19th ACM Symposium on Theory of Computing,* 296-304, 1987.

[Natarajan91] *Machine Learning: A Theoretical Approach.* Morgan Kaufmann, 1991.

[Pagallo and Haussler 90] Pagallo, G.; and Haussler, D. Boolean Feature Discovery in Empirical Learning. *Machine Learning,* 5(1):71–100, 1990.

[Peterson and Weldon 72] Peterson, W. W. and Weldon, E. J. *Error Correcting Codes*, The MIT Press. p.86. 1972.

[Queiros and Gelsma 84] Queiros, C.E. and Gelsma, E.S. On Feature Selection. *The Seventh International Conference on Pattern Recognition*, 128–130, 1984.

[Quinlan 86] Quinlan, J. R. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986.

[Rice 88] Rice, J.A. *Mathematical Statistics and Data Analysis.* Wadthworth and Brooks/Cole Advanced Books and Software. Pacific Grove, California. 1988.

[Rivest 87] Rivest, R. Learning Decision-Lists. it Machine Learning, 2(3):229-246, 1987.

[Ross 88] Ross, S. *A First Course in Probability.* Macmillan Publishing Company, New York. 3rd edition. 1988.

[Rumelhart *et al.* 86] Rumelhart, D.E. , Hinton, G.E. and Williams, R.J. Learning internal representations by error propagation. In Rumelhart, D. E., and McClelland, J. L., (eds.) *Parallel Distributed Processing*, Vol 1. 318-362. 1986.

[Slepian53] Slepian, D. On the Number of Symmetry Types of Boolean Functions of n Variables. *Can. J. Math.* , 5(2):185–193, 1953.

[Utgoff 86] Utgoff, P.E. Machine Learning of Inductive Bias. *Machine Learning of Inductive Bias.* Kluwer Academic Publishers, 1986.

[Valiant 84] Valiant, L.G. A Theory of the Learnable. *Communications of ACM*, 27(11):1134-1142, 1984.

[Verbeurgt 90] Verbeurgt, K. Learning DNF Under the Uniform Distribution in Quasi-polynomial Time. *Proceedings of the Third Annual Workshop on Computational Learning Theory. Morgan Kaufmann Publishers*, 1990.

[Wolpert 90] Wolpert, D. A mathematical theory of generaliation: Parts I and II. *Complex Systems, 4* (2):151–249, 1990.

APPENDIX

We list in the following pages the results for all 100 sample-concept pairs tested in the experiments of Subsection 5.4.4.

$\diamond$ FOCUS+ID3

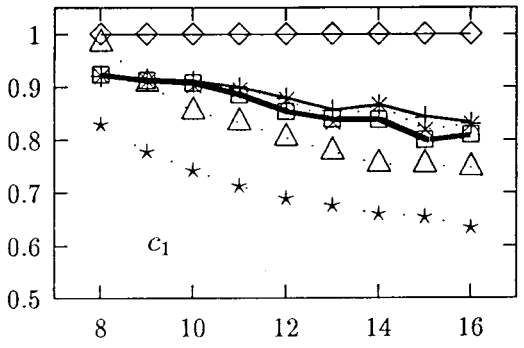
$+$ WG+ID3

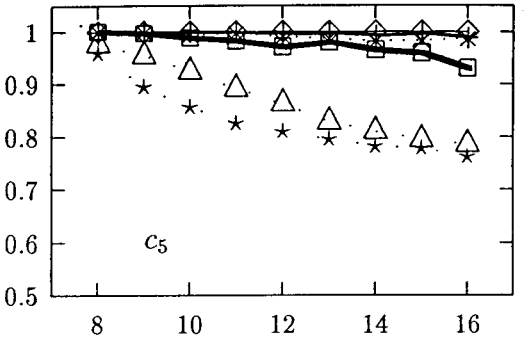
$\times$ MIG+ID3

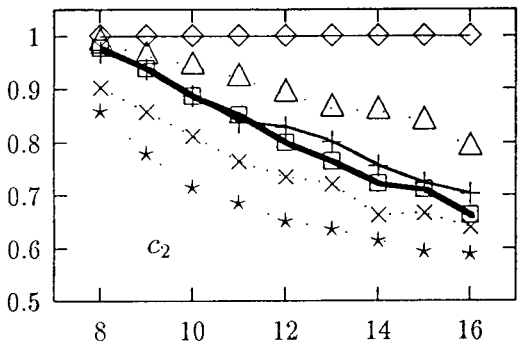
$\square$ SG+ID3

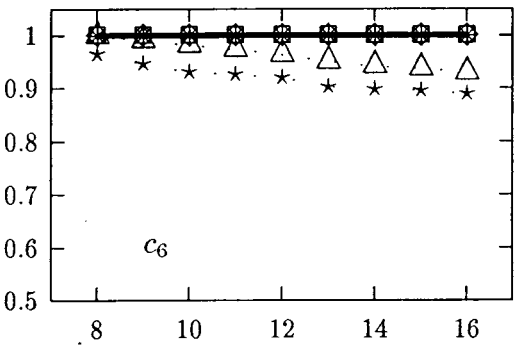
$\triangle$ FRINGE

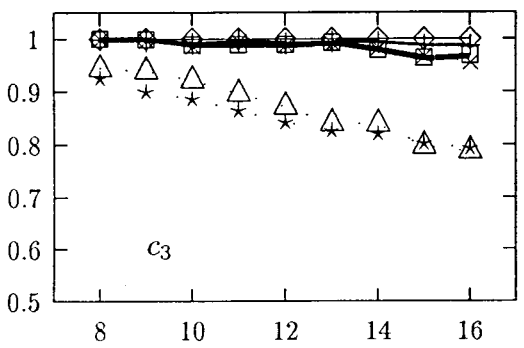
$\star$ ID3

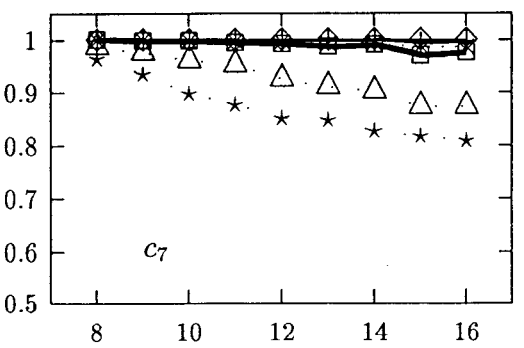

1
0.9
0.8
0.7
0.6
0.5
c_1
8
10
12
14
16

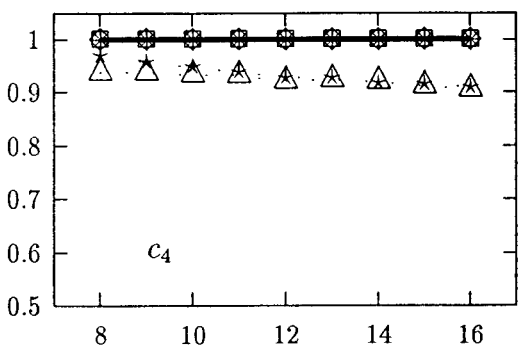

1
0.9
0.8
0.7
0.6
0.5
c_5
8
10
12
14
16

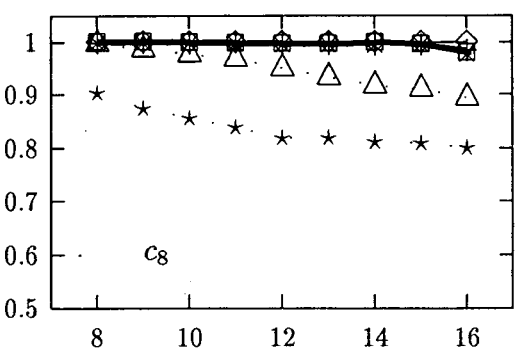

1
0.9
0.8
0.7
0.6
0.5
c_2
8
10
12
14
16

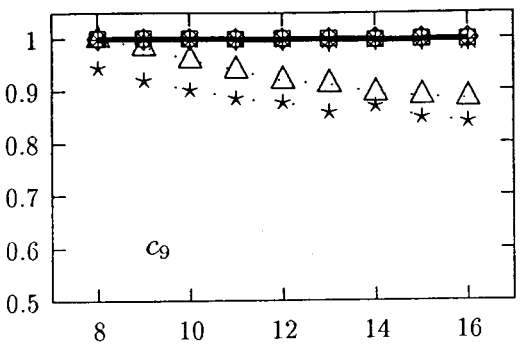

1
0.9
0.8
0.7
0.6
0.5
c_6
8
10
12
14
16

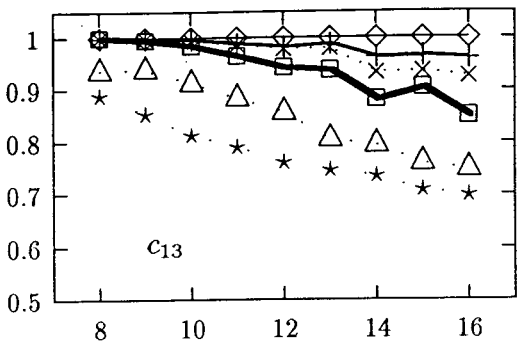

1
0.9
0.8
0.7
0.6
0.5
c_3
8
10
12
14
16

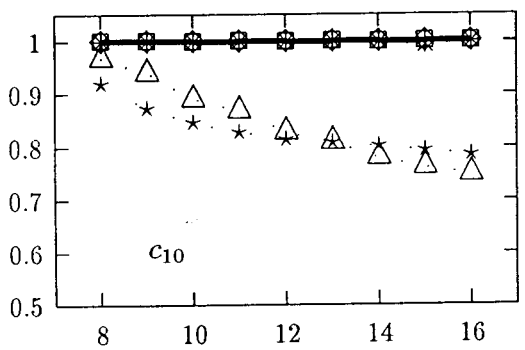

1
0.9
0.8
0.7
0.6
0.5
c_7
8
10
12
14
16

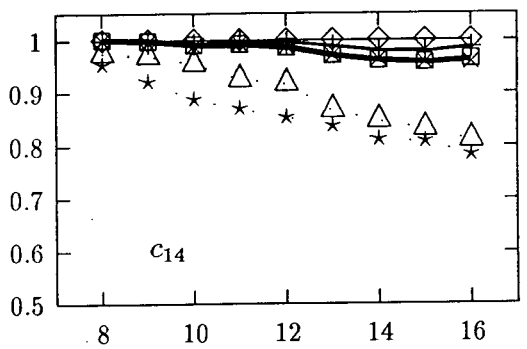

1
0.9
0.8
0.7
0.6
0.5
c_4
8
10
12
14
16

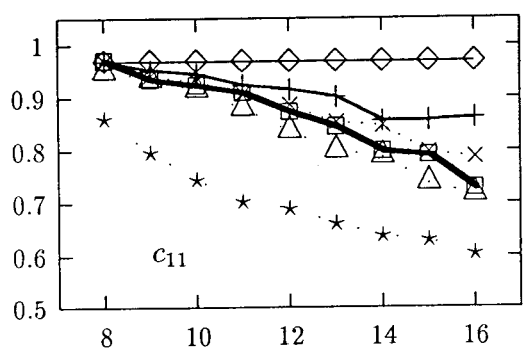

1
0.9
0.8
0.7
0.6
0.5
c_8
8
10
12
14
16

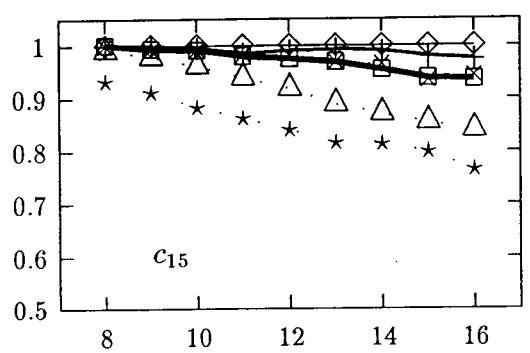

1
0.9
0.8
0.7
0.6
0.5
c_9
8
10
12
14
16

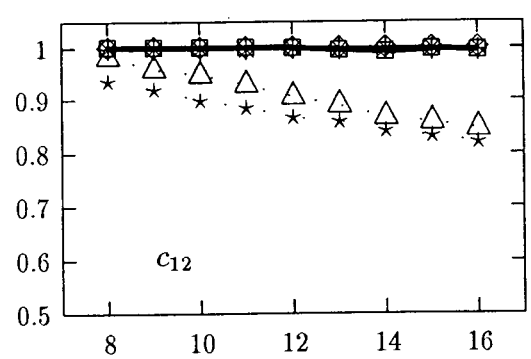

1
0.9
0.8
0.7
0.6
0.5
c_{13}
8
10
12
14
16

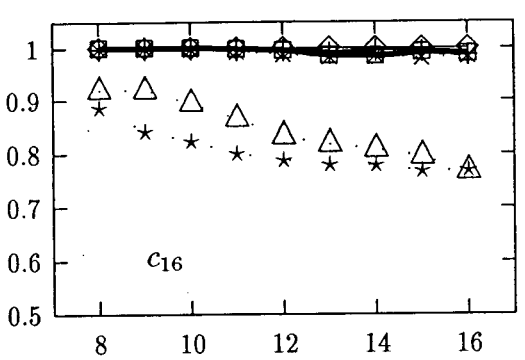

1
0.9
0.8
0.7
0.6
0.5
c_{10}
8
10
12
14
16

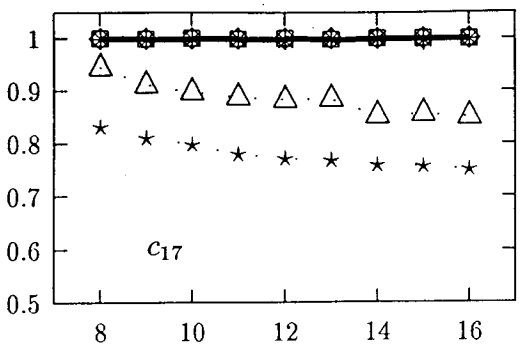

1
0.9
0.8
0.7
0.6
0.5
c_{14}
8
10
12
14
16

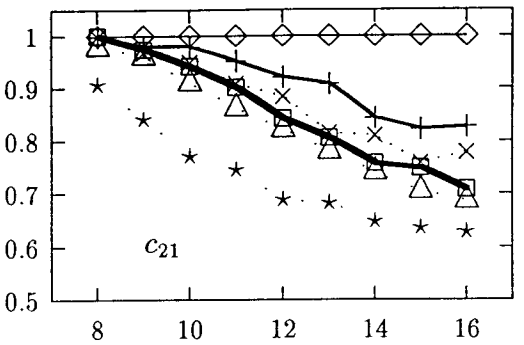

1
0.9
0.8
0.7
0.6
0.5
c_{11}
8
10
12
14
16

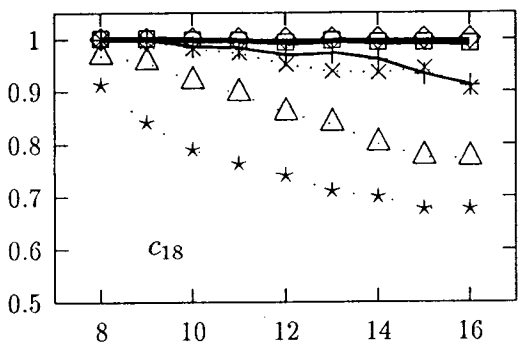

1
0.9
0.8
0.7
0.6
0.5
c_{15}
8
10
12
14
16

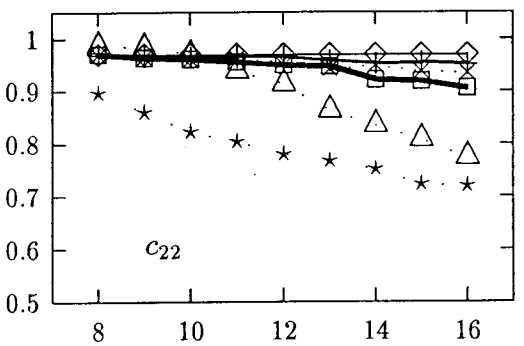

1
0.9
0.8
0.7
0.6
0.5
c_{12}
8
10
12
14
16

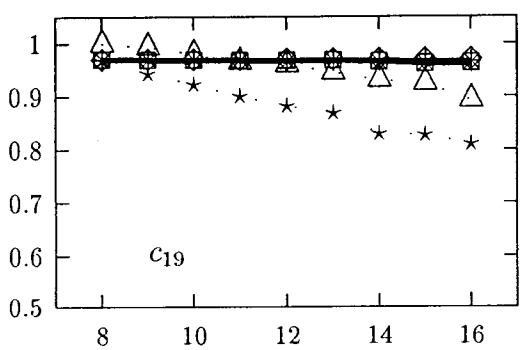

1
0.9
0.8
0.7
0.6
0.5
c_{16}
8
10
12
14
16

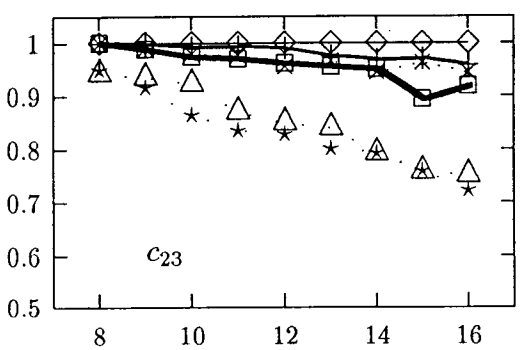

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{17}

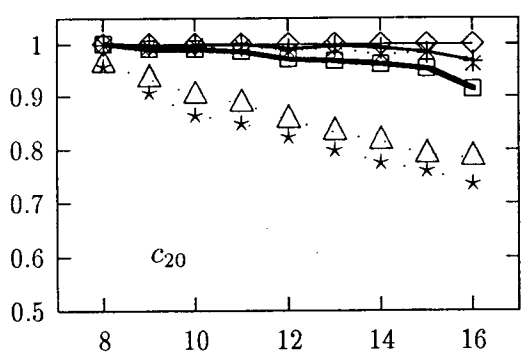

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{21}

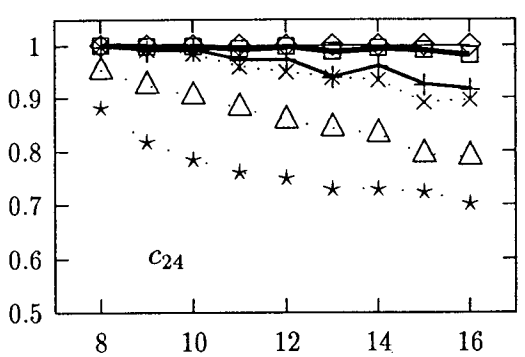

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{18}

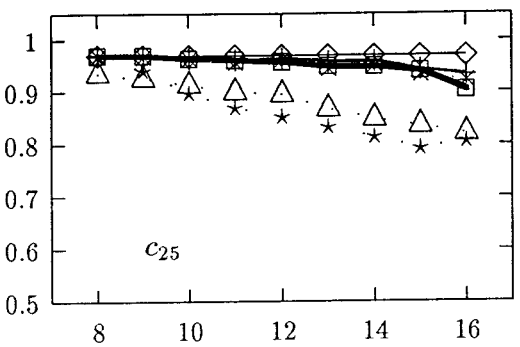

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{22}

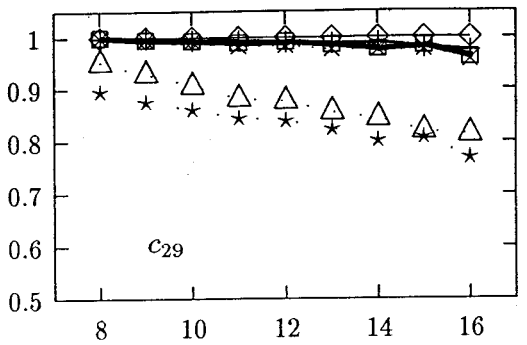

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{19}

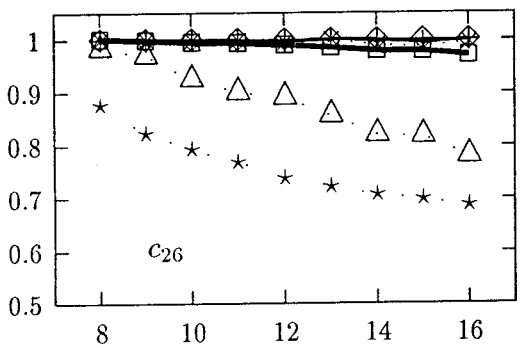

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{23}

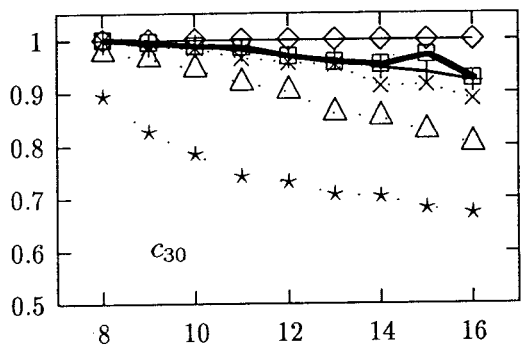

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{20}

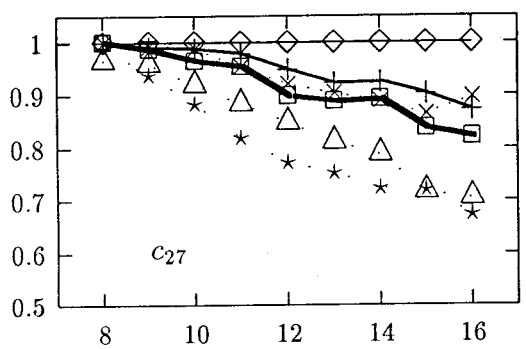

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{24}

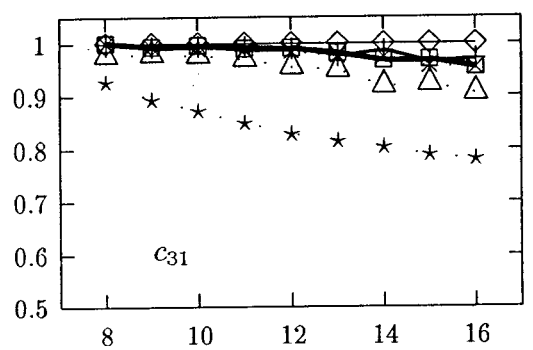

1
0.9
0.8
0.7
0.6
0.5
c_{25}
8
10
12
14
16

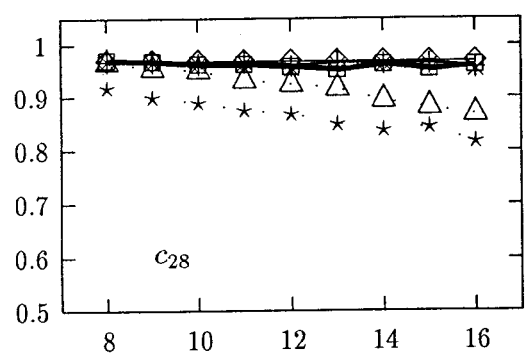

1
0.9
0.8
0.7
0.6
0.5
c_{29}
8
10
12
14
16

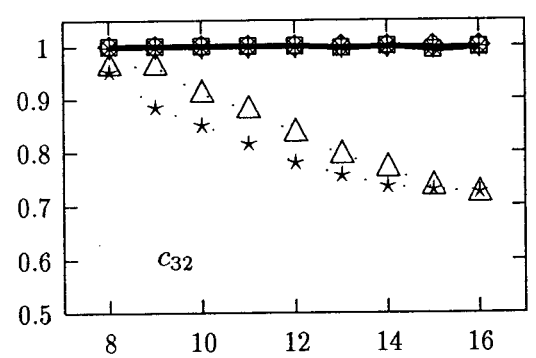

1
0.9
0.8
0.7
0.6
0.5
c_{26}
8
10
12
14
16

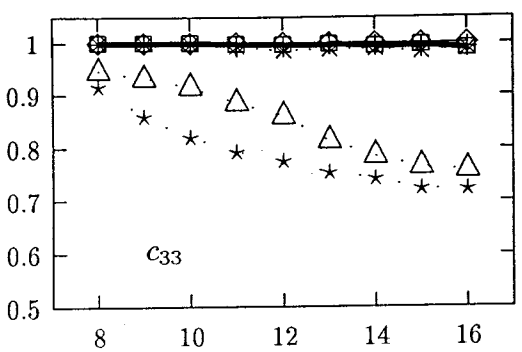

1
0.9
0.8
0.7
0.6
0.5
c_{30}
8
10
12
14
16

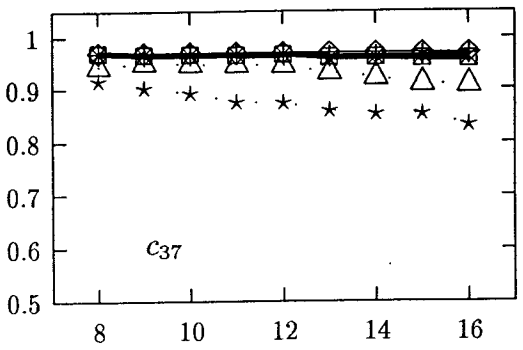

1
0.9
0.8
0.7
0.6
0.5
c_{27}
8
10
12
14
16

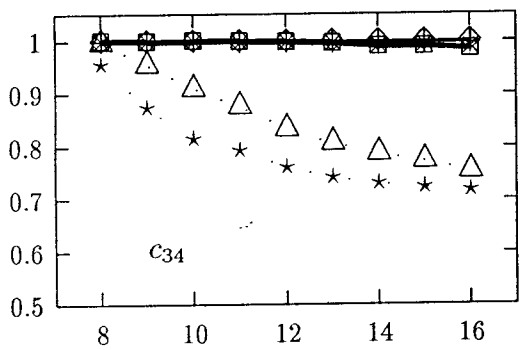

1
0.9
0.8
0.7
0.6
0.5
c_{31}
8
10
12
14
16

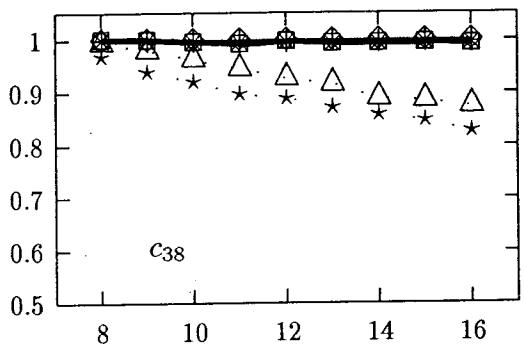

1
0.9
0.8
0.7
0.6
0.5
c_{28}
8
10
12
14
16

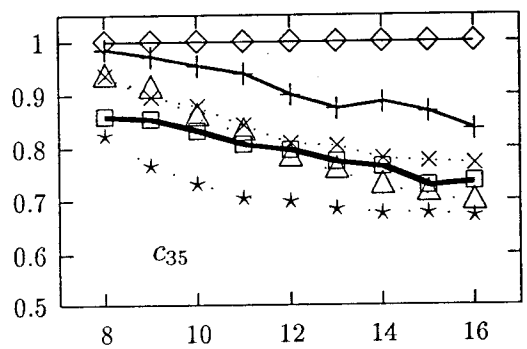

1
0.9
0.8
0.7
0.6
0.5
c_{32}
8
10
12
14
16

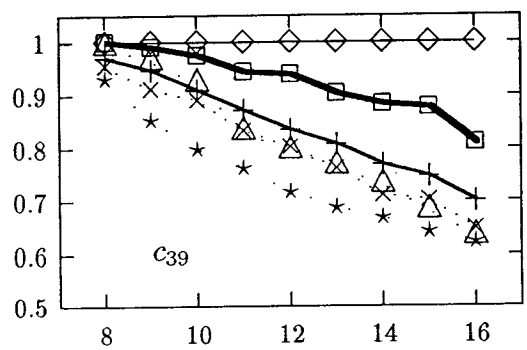

1
0.9
0.8
0.7
0.6
0.5
c_{33}
8
10
12
14
16

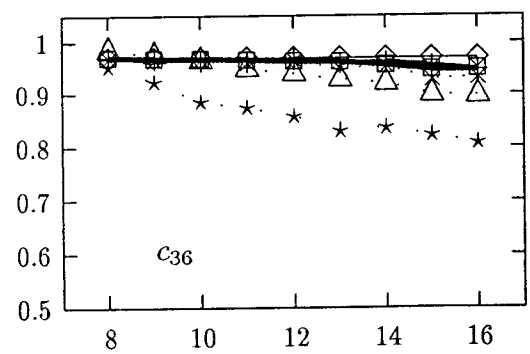

1
0.9
0.8
0.7
0.6
0.5
c_{37}
8
10
12
14
16

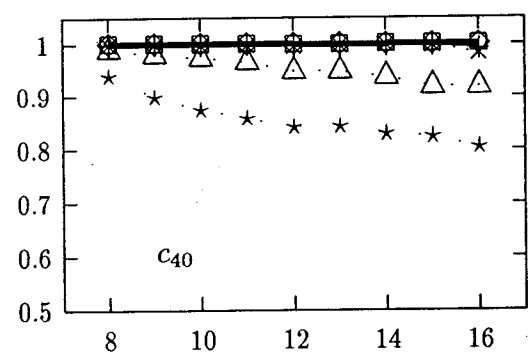

1
0.9
0.8
0.7
0.6
0.5
c_{34}
8
10
12
14
16

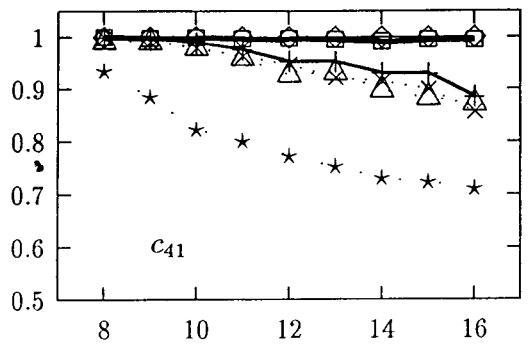

1
0.9
0.8
0.7
0.6
0.5
c_{38}
8
10
12
14
16

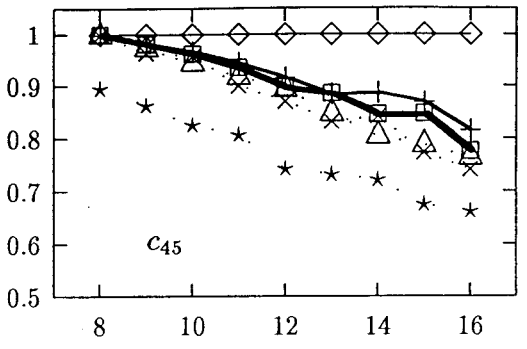

1
0.9
0.8
0.7
0.6
0.5
c_{35}
8
10
12
14
16

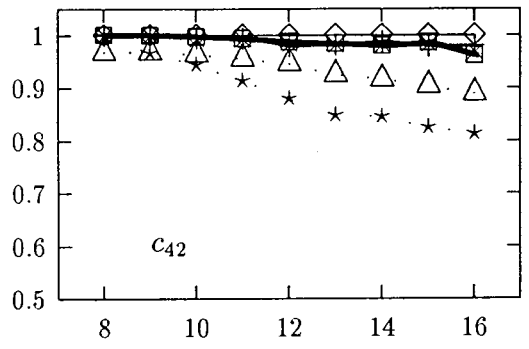

1
0.9
0.8
0.7
0.6
0.5
c_{39}
8
10
12
14
16

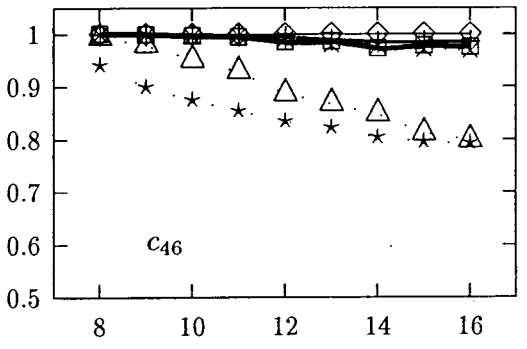

1
0.9
0.8
0.7
0.6
0.5
c_{36}
8
10
12
14
16

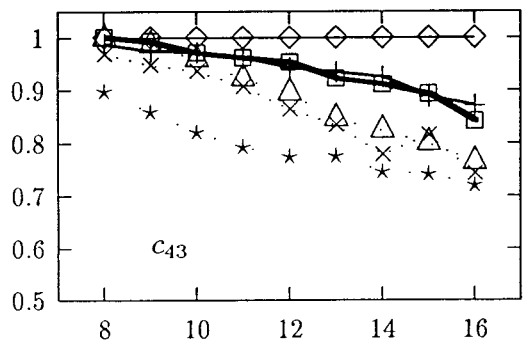

1
0.9
0.8
0.7
0.6
0.5
c_{40}
8
10
12
14
16

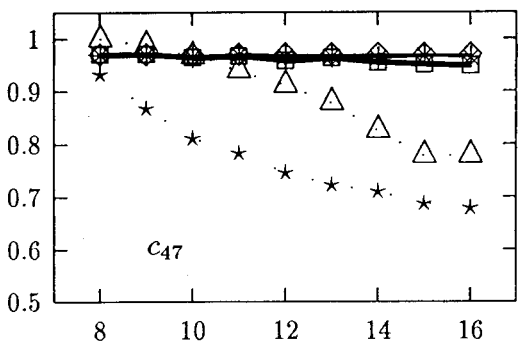

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{41}

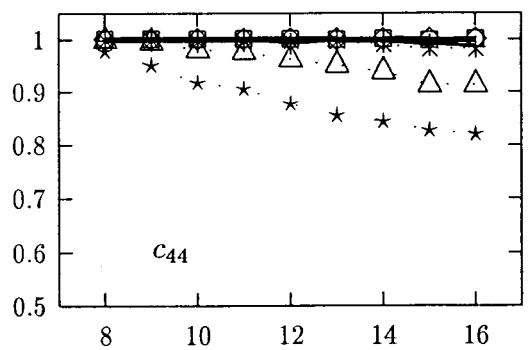

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{45}

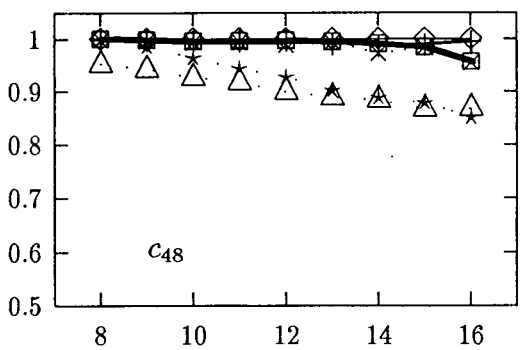

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{42}

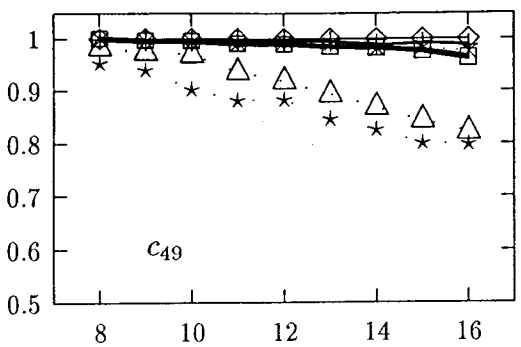

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{46}

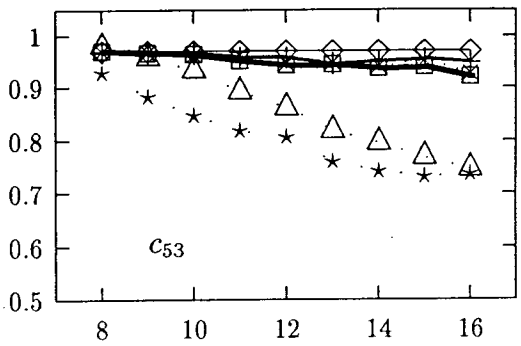

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{43}

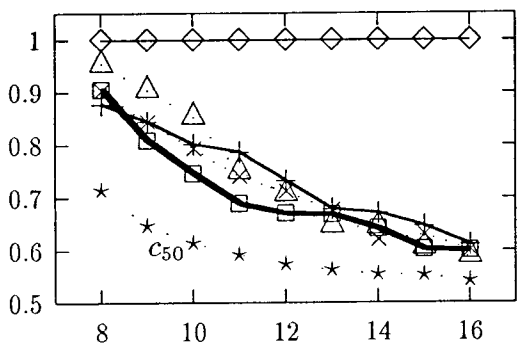

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{47}

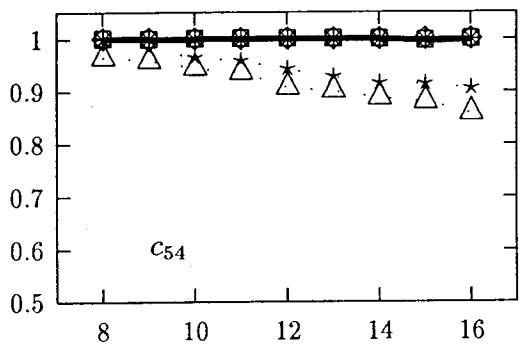

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{44}

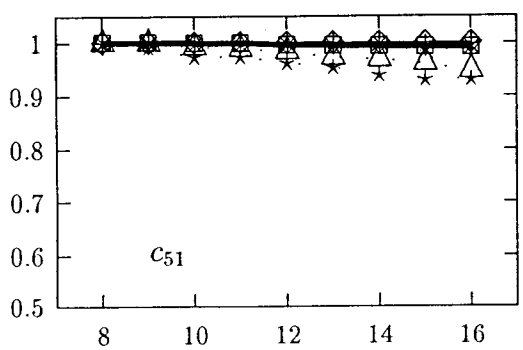

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{48}

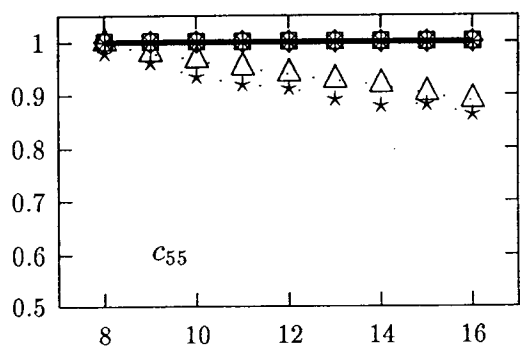

1
0.9
0.8
0.7
0.6
0.5
c_{49}
8
10
12
14
16

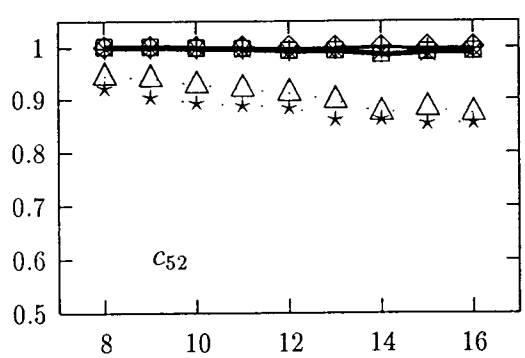

1
0.9
0.8
0.7
0.6
0.5
c_{53}
8
10
12
14
16

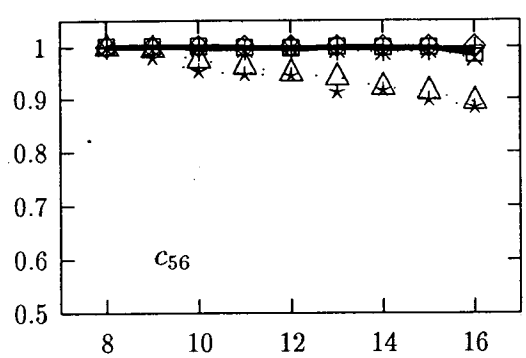

1
0.9
0.8
0.7
0.6
0.5
c_{50}
8
10
12
14
16

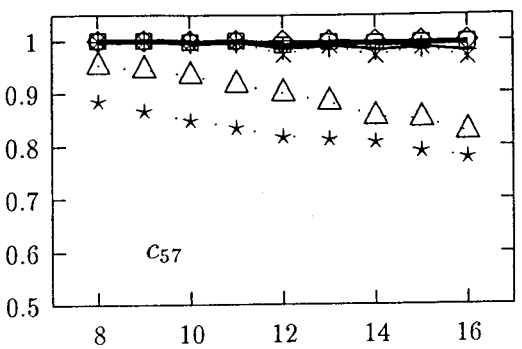

1
0.9
0.8
0.7
0.6
0.5
c_{54}
8
10
12
14
16

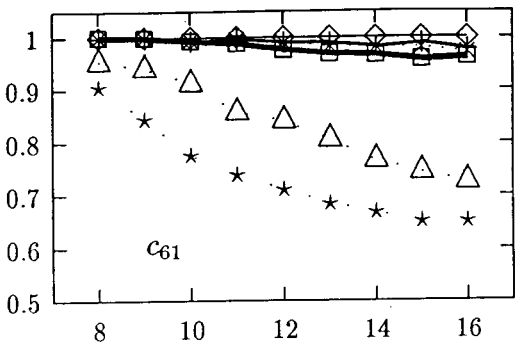

1
0.9
0.8
0.7
0.6
0.5
c_{51}
8
10
12
14
16

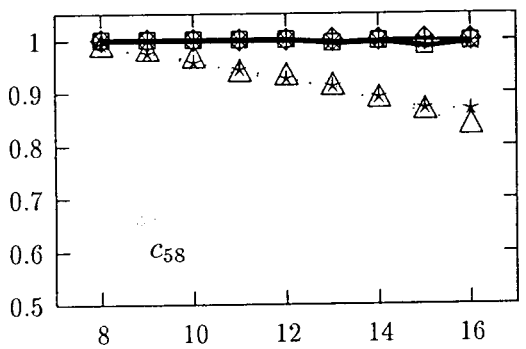

1
0.9
0.8
0.7
0.6
0.5
c_{55}
8
10
12
14
16

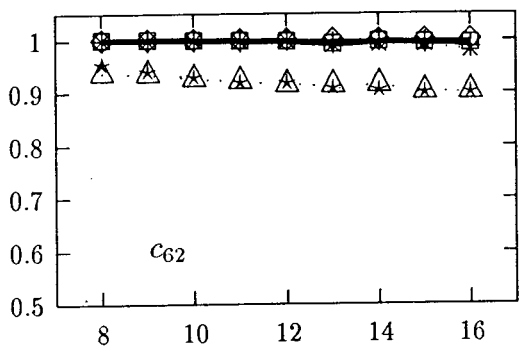

1
0.9
0.8
0.7
0.6
0.5
c_{52}
8
10
12
14
16

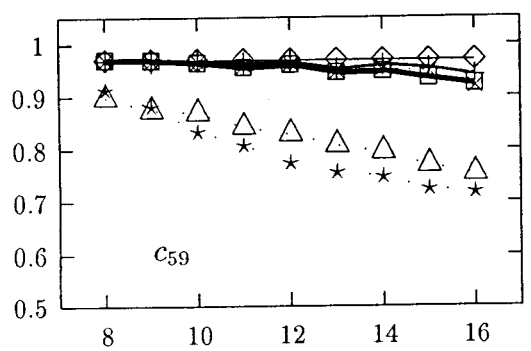

1
0.9
0.8
0.7
0.6
0.5
c_{56}
8
10
12
14
16

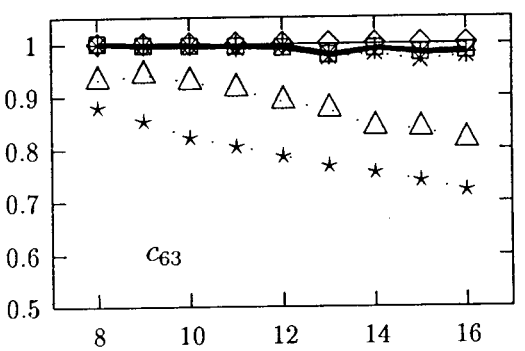

1
0.9
0.8
0.7
0.6
0.5
c_{57}
8
10
12
14
16

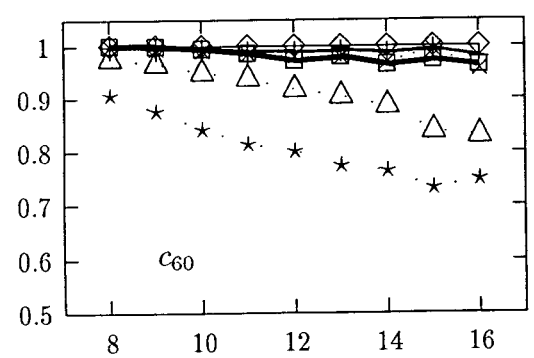

1
0.9
0.8
0.7
0.6
0.5
c_{61}
8
10
12
14
16

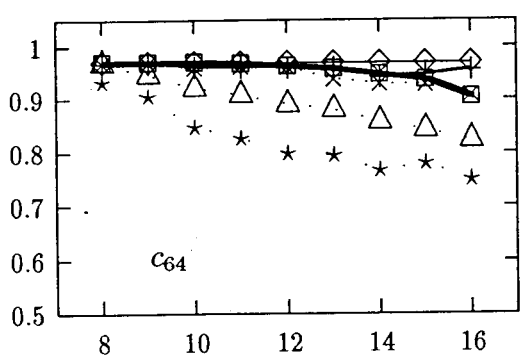

1
0.9
0.8
0.7
0.6
0.5
c_{58}
8
10
12
14
16

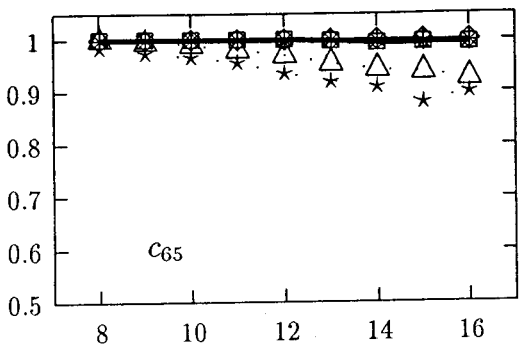

1
0.9
0.8
0.7
0.6
0.5
c_{62}
8
10
12
14
16

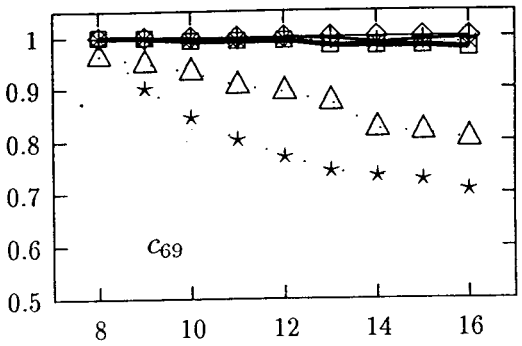

1
0.9
0.8
0.7
0.6
0.5
c_{59}
8
10
12
14
16

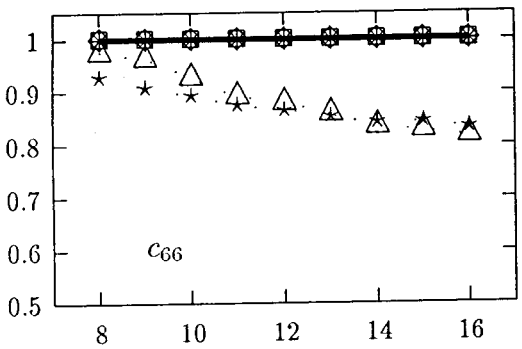

1
0.9
0.8
0.7
0.6
0.5
c_{63}
8
10
12
14
16

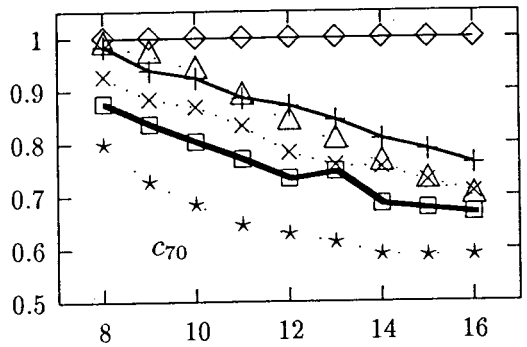

1
0.9
0.8
0.7
0.6
0.5
c_{60}
8
10
12
14
16

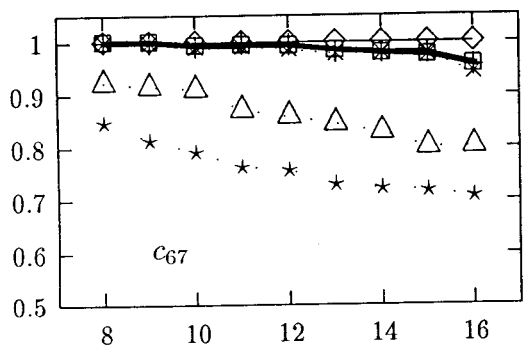

1
0.9
0.8
0.7
0.6
0.5
c_{64}
8
10
12
14
16

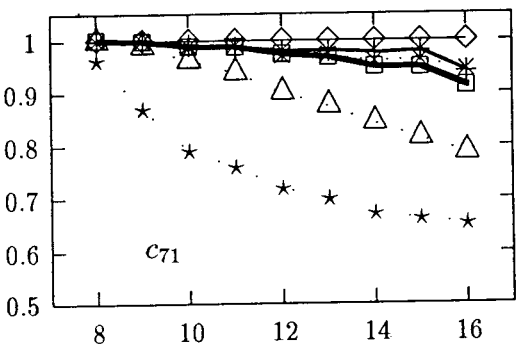

1
0.9
0.8
0.7
0.6
0.5
c_{65}
8
10
12
14
16

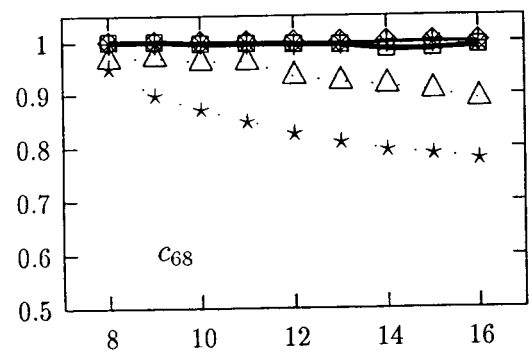

1
0.9
0.8
0.7
0.6
0.5
c_{69}
8
10
12
14
16

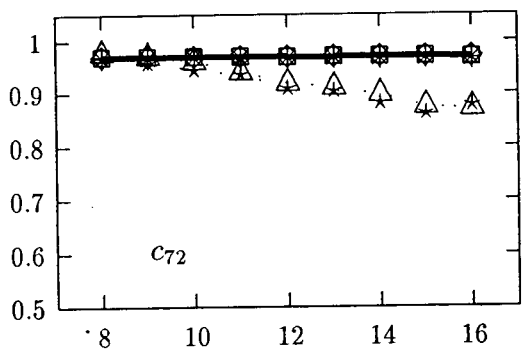

1
0.9
0.8
0.7
0.6
0.5
c_{66}
8
10
12
14
16

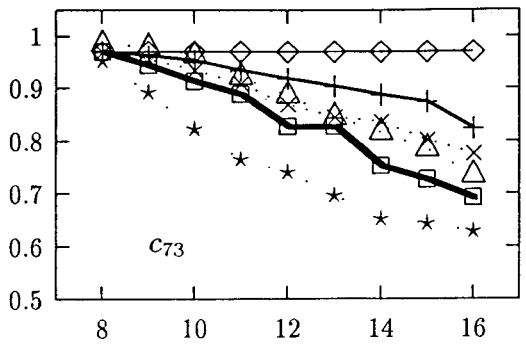

1
0.9
0.8
0.7
0.6
0.5
c_{70}
8
10
12
14
16

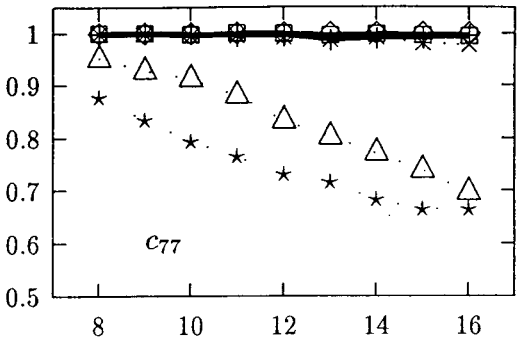

1
0.9
0.8
0.7
0.6
0.5
c_{67}
8
10
12
14
16

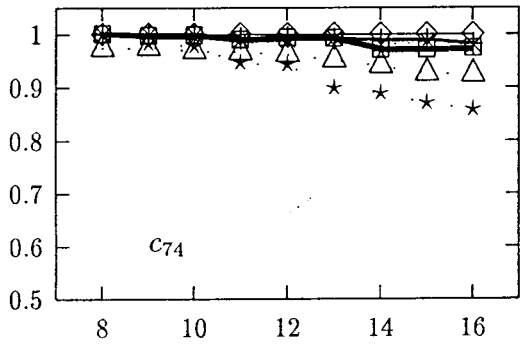

1
0.9
0.8
0.7
0.6
0.5
c_{71}
8
10
12
14
16

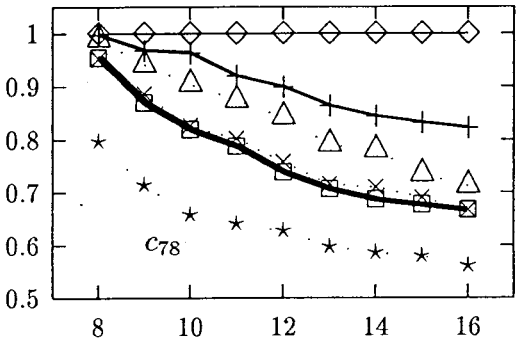

1
0.9
0.8
0.7
0.6
0.5
c_{68}
8
10
12
14
16

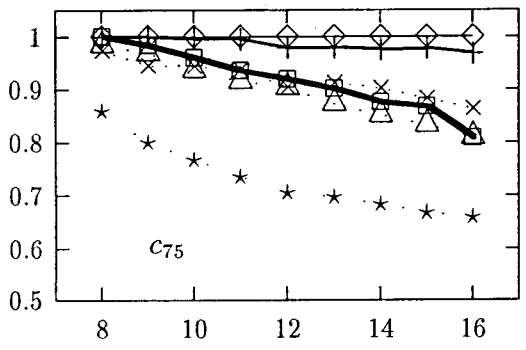

1
0.9
0.8
0.7
0.6
0.5
c_{72}
8
10
12
14
16

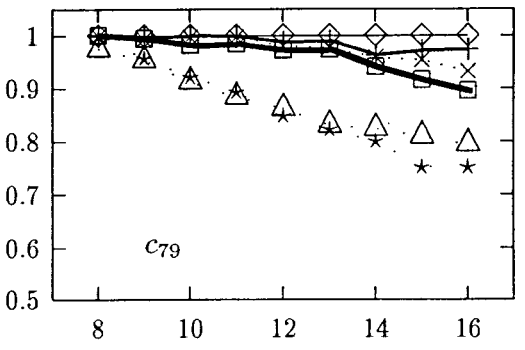

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{73}

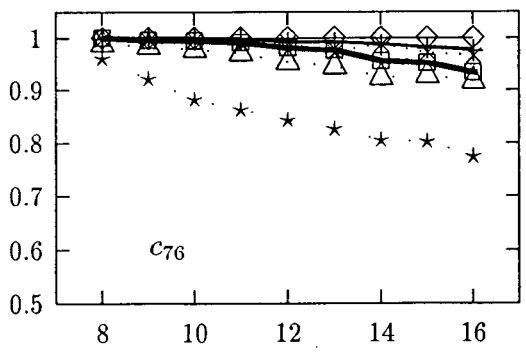

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{77}

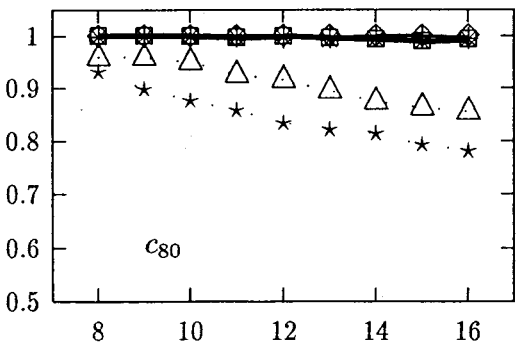

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{74}

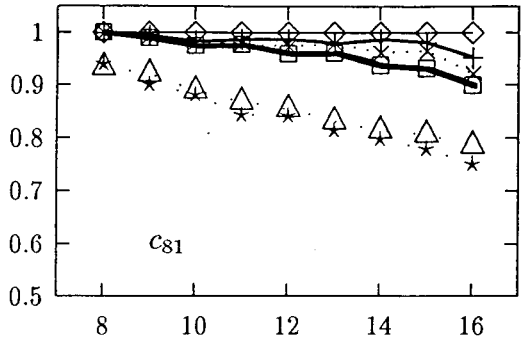

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{78}

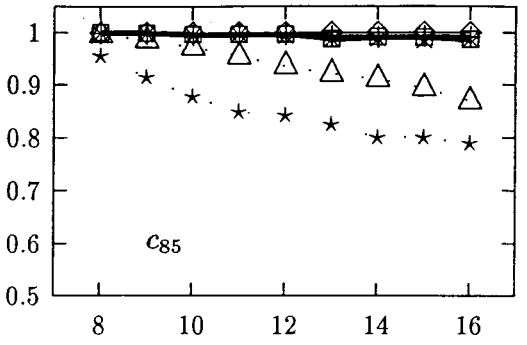

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{75}

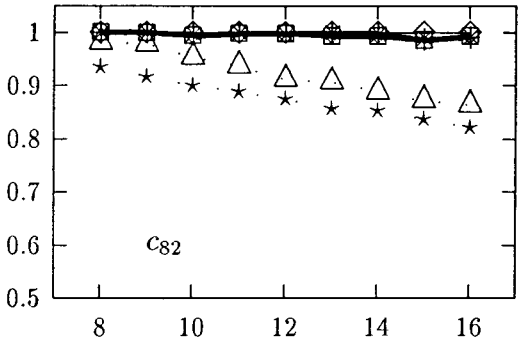

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{79}

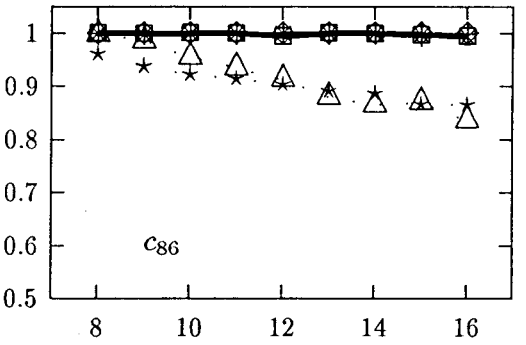

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{76}

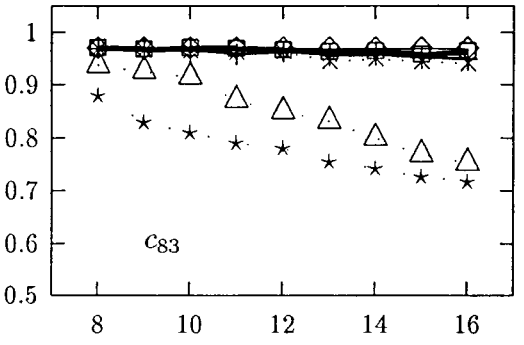

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{80}

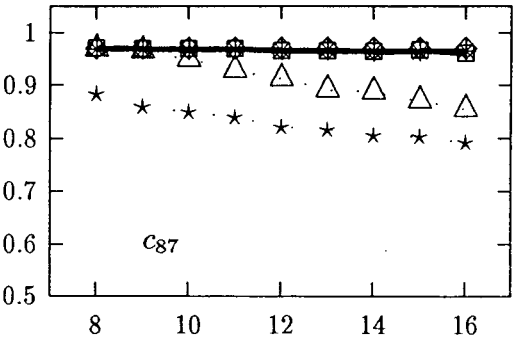

1
0.9
0.8
0.7
0.6
0.5
c_{81}
8
10
12
14
16

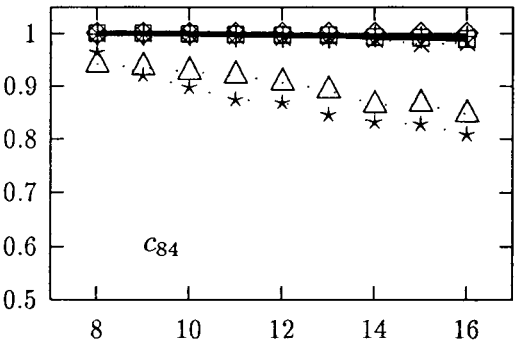

1
0.9
0.8
0.7
0.6
0.5
c_{85}
8
10
12
14
16

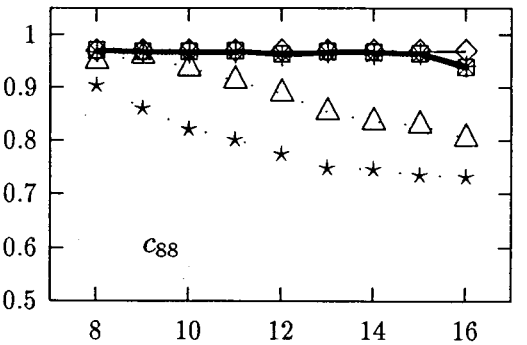

1
0.9
0.8
0.7
0.6
0.5
c_{82}
8
10
12
14
16

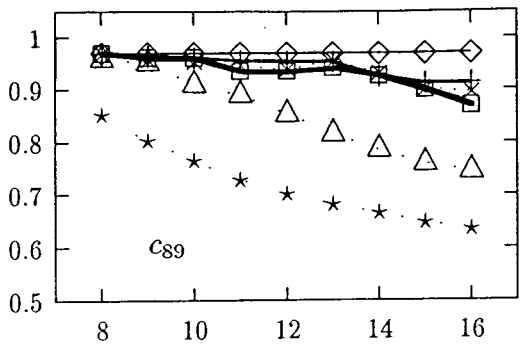

1
0.9
0.8
0.7
0.6
0.5
c_{86}
8
10
12
14
16

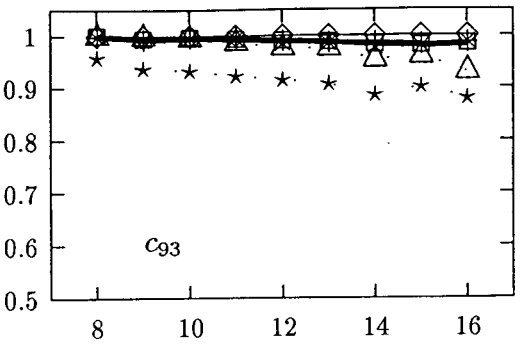

1
0.9
0.8
0.7
0.6
0.5
c_{83}
8
10
12
14
16

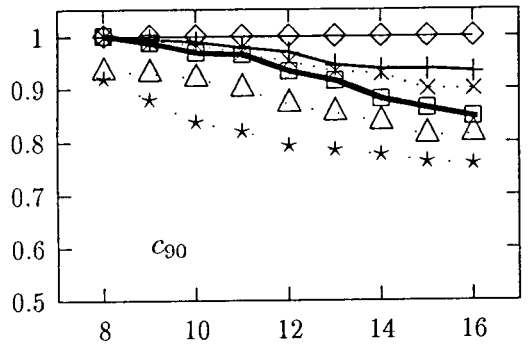

1
0.9
0.8
0.7
0.6
0.5
c_{87}
8
10
12
14
16

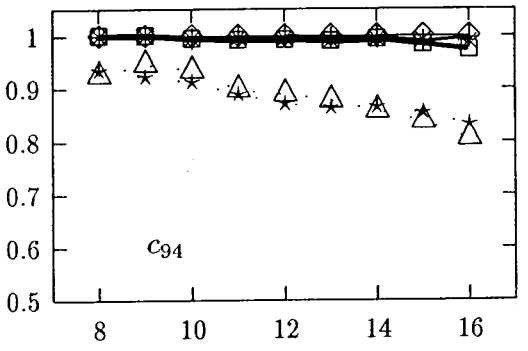

1
0.9
0.8
0.7
0.6
0.5
c_{84}
8
10
12
14
16

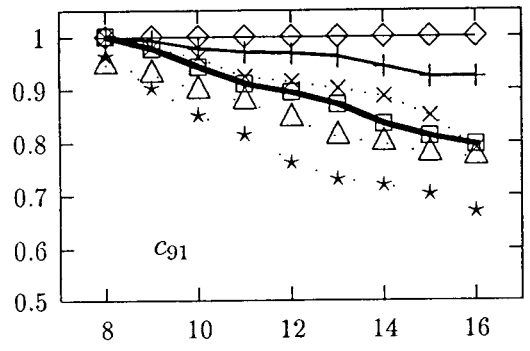

1
0.9
0.8
0.7
0.6
0.5
c_{88}
8
10
12
14
16

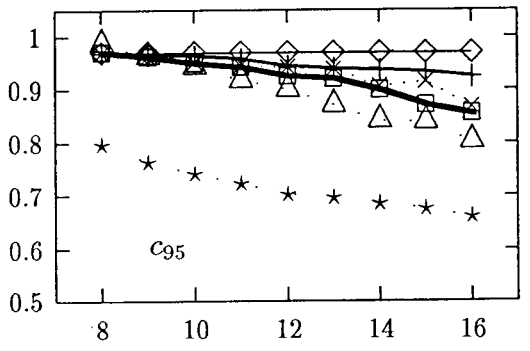

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{89}

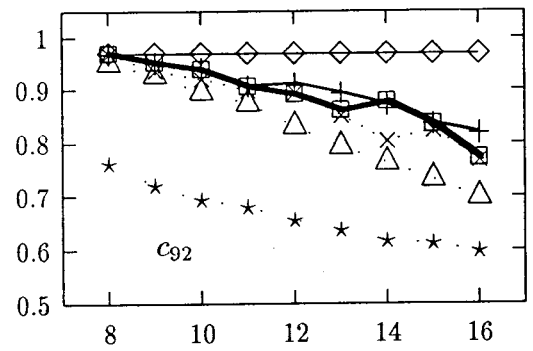

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{93}

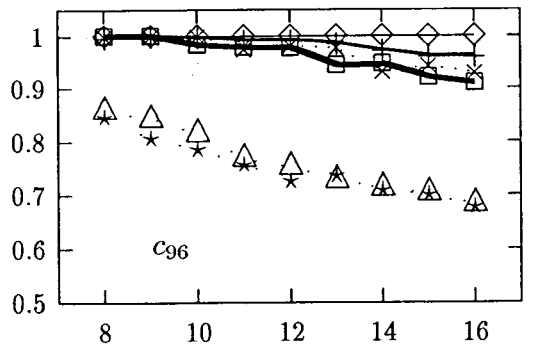

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{90}

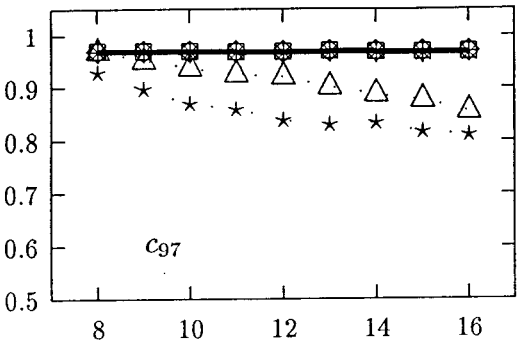

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{94}

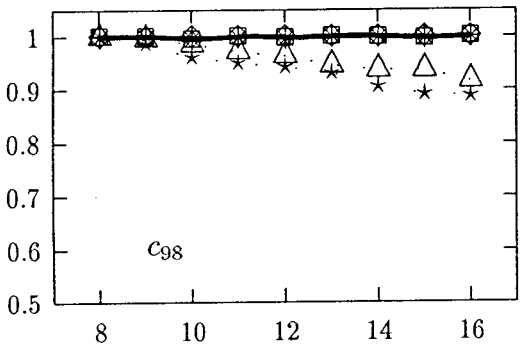

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{91}

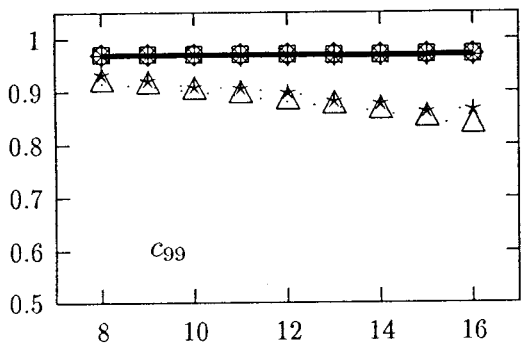

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{95}

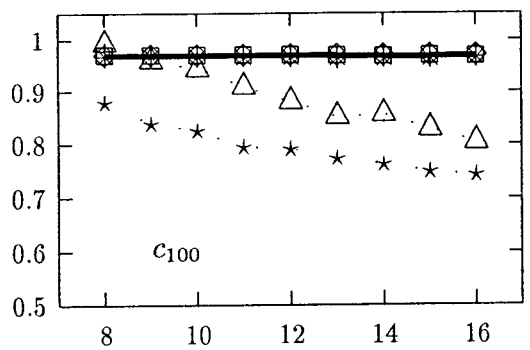

1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{92}


1
0.9
0.8
0.7
0.6
0.5
8
10
12
14
16
c_{96}


1
0.9
0.8
0.7
0.6
0.5
c_{97}
8
10
12
14
16


1
0.9
0.8
0.7
0.6
0.5
c_{98}
8
10
12
14
16


1
0.9
0.8
0.7
0.6
0.5
c_{99}
8
10
12
14
16


1
0.9
0.8
0.7
0.6
0.5
c_{100}
8
10
12
14
16