Probabilistic Learning of Boolean Concepts

Thomas Papadakis

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

June 1986

ISBN 0-315-32231-4

ABSTRACT

Probabilistic Learning of Boolean Concepts

Thomas Papadakis

The problem of boolean concept learning from examples is discussed in this thesis. L. G. Valiant's proposal to consider approximate rather than exact learning has been adopted.

Valiant's algorithms learn in polynomial time an approximation to a boolean concept from error-free or noisy examples, when a bound on the number of literals per disjunts/conjuct is known. These algorithms have been considerably improved here. In the case of error-free examples, our algorithms are non-asymptotically faster than Valiant's; in the case of noisy examples, they are both asymptotically and non-asymptotically faster, and they can also handle more noisy examples.

New algorithms, approximating boolean concepts when no knowledge of the maximum number of literals per disjunct/conjuct is known, have also been designed. In the case of error-free examples, they are asymptotically as fast

as Valiant's; in the case of noisy examples, a user-specified parameter determines the probability that they will be asymptotically as fast as Valiant's.

ACKNOWLEDGEMENTS

I wish to express my gratitude to my supervisor, Prof. D. K. Probst. His guidance and suggestions throughout this research, as well as his patience, understanding, and advice throught my most difficult moments at Concordia have been an invaluable aid.

I also wish to express my deep appreciation to Prof. V. S. Alagar, for everything he taught me when I was working with him, and for the encouragement he gave me on my past and future plans.

I will be always grateful to my high school math teacher, Mr. A. Katsoulakis, for the great love for Mathematics he inspired in me.

Special thanks to my family for their moral support, and to Mr. P. Mylonakis for his help during my stay in Montreal.

Financial support of this research through N.S.E.R.C. and F.C.A.C. is also acknowledged.

To Σοφη

TABLE OF CONTENTS

CHAPTER 1

SOME MACHINE LEARNING CONTEXT

1.1. Introduction

Human intelligence, one of the central themes in many disciplines like philosophy, psychology, and anthropology, has been a very interesting and intricate subject of study since the time of the ancient Greeks. Although no precise and universally acceptable definition of it has been given, most people agree that intelligence consists of the ability to understand a language, to develop personal beliefs, to learn and comprehend, to organize acquired knowledge, to reason logically and draw conclusions, to judge, to set up goals and plans to achieve them, to invent and to create by using acquired knowledge as well as intuition and inspiration, and so on.

Since the human species is, by nature, ambitious it tried, in addition of just studying intelligence, to also generate/reproduce it. The issue of whether the construction of artifacts as intelligent as human beings is feasible (they should be able to compose like Beethoven, or do science like Newton) or even desirable (they should have the freedom to use their intelligence to become notorious criminals - whatever that means) is debatable. However, we will not discuss it here*. What everyone seems to agree on is that **computer systems more intelligent that today's can be constructed**; this is the target of undergoing research.

Since learning is one of the characteristics of intelligence, it follows that Artificial Intelligence (AI),

> the part of Computer Science concerned with designing intelligent computer systems;
>
> ([BAFEI81], p. 3)

is, among others, concerned with the exploration of (machine) learning. ■

An introduction to the machine learning field**, from the AI viewpoint, is given in this chapter. That will give

---

* The feasibility of machine intelligence was first extensively discussed by Allan Turing ([TURIN50]) in 1950; he was quick to recognize possible objections. Herbert Dreyfus became its most famous opponent in the early 70's ([DREYF72]). Pamela McCorduck gave a fair and entertaining treatment of the subject ([MCCOR79], Chapters 8 and 9).

** See also [SCHAN82] for an general discussion on machine learning.

the context of our thesis' work, which is on machine learning.

The importance of machine learning is explained in Section 1.2, and its history is briefly sketched in Section 1.3. A taxonomy of machine learning research is given in Section 1.4, where the category of "learning from examples" is identified to be the one in which our work falls. Next, in Section 1.5, this category of machine learning is presented in more detail, after being narrowed down to the sub-category of "single concept learning from examples"; this is the exact kind of learning our algorithms perform. The issue of noisy data is discussed next in Section 1.6.

Mitchell's version space method, providing a general framework for the description and comparison of many learning algorithms, is discussed in Section 1.7. L. G. Valiant's new idea for learning from examples is stated in Section 1.8; this is the starting point of our work.

Valiant gave a great emphasis to the discovery of polynomial time learning algorithms. The issue therefore of the analysis of algorithms comes into the scene, and is discussed in Section 1.9.

The thesis' organization is finally outlined in Section 1.10, and a summary of Chapter 1 is given in Section 1.11. ■

1.2. Importance of Machine Learning

The importance of machine learning* will be described in the context of each of its primary research areas. Those areas are the following.

I. The transformation of knowledge of humans into knowledge usable by expert systems. An expert system** performs tasks like medical diagnosis, mineral exploration, or VLSI design, which ordinarily require human intelligence. It achieves that by maintaining a knowledge base, where known facts are stored, and by applying a set of inference rules to this knowledge, determining in this way its next action.

Although the knowledge stored in the knowledge base of an expert system can be represented in many different forms, human knowledge is not prepackaged in any of those forms; it is not even known where and how

* The value of machine learning was first extensively and skeptically discussed by H. Simon ([SIMON83]) in 1983. The same year, P. Scott ([SCOTT83]) gave an answer to Simon's views.

** See [YAMAX84], or any AI text, for an introduction to the expert systems field.

human knowledge is stored in the brain, and it is difficult to transform such knowledge into machine-usable form. Such a transformation is nowadays done by Knowledge Engineers. The question therefore is whether machines can take on (part of) the burden of this job. If machines can learn (i.e. transform knowledge), then more sophisticated expert systems will be constructable, and therefore, less clerical work will be left for humans.

The field of expert systems design has faced an enormous commercialization the last five years. Many companies in the U.S.A., Japan, and Europe have launched multi-million programs to develop this technology*. Since even F. Hayes-Roth, one of the expert systems experts, admits that

> when compared with a human expert, today's expert system appears narrow, shallow, and brittle, lacking the human expert's breadth of knowledge and understanding of fundamental principles,
>
> ([HAYES84], p. 264)

it can be expected that significant results will be

----

* See [HAYES84] for a discussion of the state of the art and the future of expert systems.

obtained in the near future.

II. <u>The exploration of the space of solvable learning problems.</u>[4] That is to say, can the "learnable" be reasonably defined, in the same way that "computable" was defined in 1971 by Stephen Cook ([GAJOH79], Chapter 2)? Can some (non-)learnable problems be identified, in the same way that some NP-complete problems were identified ([GAJOH79], Appendix)? Is there anything which can be done about the non-learnable problems, in the same way that approximate solutions were derived for the NP-complete problems ([BAASE78], Chapter 7)?

To the best of our knowledge, L. G. Valiant's work ([VALIA84a], [VALIA84b], and [VALIA85]) is the first to address issues of this kind, and so far it hasn't been referred to by any other researcher.

III. <u>The computer simulation of the human learning process.</u> Even the most basic form of learning, that of memorizing, is done differently by computers and by humans; for a computer it is equally easy to memorize a poem and a page from the telephone directory, but for a person it is not.

Fruitful studies in this area of machine learning may enable psychologists to test, establish, and develop new theories, and therefore, to better understand learning mechanisms. Subsequently, such an understanding may help, on the one hand, educators to improve people's learning performance, and on the other hand, computer scientists to derive better learning algorithms, or to realize the limitations of learning.

Many distinguished scientists consider this area of machine learning to be very important. Herbert Simon, for example, who received the Nobel Prize in Economics in 1978, as well as the ACM Turing award in 1975, and who has also done significant work in cognitive psychology, AI, political science, management studies, operations research, etc. ([RALST83], p. 1324), believes that

> a very high priority [should be given] to research aimed at simulating, and thereby understanding, human learning.
>
> ([SIMON83], p. 35)

Unfortunately, not much attention has been paid to this area, or, at least, no significant results have been obtained. This is probably due to the fact that AI alone is not sufficient to advance the field; the contribution of (at least) cognitive psychology is essential. ■

Our thesis' work falls clearly into category II. It improves and extends L. G. Valiant's ideas, to be outlined in Section 1.8. ■

1.3. History of Machine Learning

A brief history of machine learning is given in this section. A more detailed history can be found in [CAMIM83], and an extensive bibliography, containing 572 references appearing before 1982, can be found in [UTNUD83]. For this reason, not many references have been included in this section, except if they are cornerstones of machine learning, or if they are not included in [CAMIM83] and [UTNUD83]. A bibliography containing recent (i.e. appearing after 1982) references is expected to be found in [MICAM86]*. ■

The 30-year old history of machine learning can be divided into three periods.

The first period started in the late 50's, and was the most ambitious one. Learning systems were simulating humans' neural networks; they were starting with simple

----

* Not yet available.

networks (i.e. with no knowledge), and they were trying to modify them towards an optimal organization (i.e. they were trying to learn), by properly reacting to stimuli. The most successful attempt of this kind was Rosenblatt's Perceptrons ([ROSEN58]). Although this approach didn't get anywhere, it generated many useful ideas and helped the development of the fields of pattern recognition and control theory in the mid 60's.

The second period of machine learning started in the late 60's, and was mainly concerned with symbolic object learning from examples, independently of the application domain. P. H. Winston's Ph.D. thesis (original version in [WINST70]; updated version in [WINST84], Chapter 11) in 1970, marked the beginning of this era, which hasn't come to its end yet*.

The third period of machine learning started in the late 70's, and is concerned with the development of knowledge acquisition tools for expert systems. As already discussed in Section 1.2, this era of machine learning has not even reached its zenith yet. ■

----

* A review of some learning systems of the 70's is given in [MICHI82]. The work done in Edinburgh in the early 80's is described in [MICHI84].

1.4. Classification of Machine Learning Research

The several kinds of machine learning are presented in this section, so that the context of our work will become more clear.

The classification of machine learning can be done, among others, along the following two directions.

I. According to the acquired knowledge's representation. The knowledge acquired by a learning system may be represented by employing any of the known knowledge representation techniques. Since the topic of knowledge representation* is, by itself, a research topic in AI, we will not get into any details here. We will restrict ourselves to mentioning that production rules, feature vectors, formal logic (e.g. propositional logic, first order predicate logic, etc.), scripts and frames, and semantic networks are the most well-known knowledge representation techniques. Several learning systems exist, acquiring knowledge in each of them, as well as in many other hybrid representations**. In addition to

----

* See [BAFEI81], pp. 141-222, for an introduction to the knowledge representation area.

** Michalski's Annotated Predicate Calculus ([MICHA83]) is such an example.

these, there exist systems learning parameters of algebraic expressions, decision trees, formal grammars, automata, etc.

The knowledge representation form used in our thesis is propositional logic. Admittedly, this is not a very powerful knowledge representation. However, it was chosen because, although it is the simplest representation form, its study is already quite complicated, as will become apparent later on.

II. According to the learning strategy used (or equivalently, according to the difference between the representations of the teacher's and the learner's knowledge). The following categories exist.

(a) Rote learning. This is simple information storage, i.e. no knowledge transformation is performed by the learning system. Although someone might argue that this should not even been considered as learning, A. Samuel, with his checkers-playing program ([SAMUE59] and [SAMUE67]), has shown that this may be quite useful. Instead of his program merely searching a few moves ahead in order to evaluate the board positions resulting from all possible moves (and therefore, in order to determine its best next move), his program was also recording the current board position along with its evaluation. Subsequently, when the same board position was encountered, his program used the evaluation already computed. In this way, the actual number of moves his program was searching ahead was increasing. Samuel's program became finally a

> rather better-than-average novice, but definitely not . . . an expert.
>
> ([SAMUE59], p. 218)

(b) Learning by being told (or learning from instruction). This can be applied when the knowledge provided by the teacher is of higher level (more general) than that used by the learner. For example, when Mostow's Foo program ([MOSTO83]), knowing the rules of playing the card game of Hearts, is given the advice "avoid taking points", it can operationalize it into (it can learn) the rule "play lower than the highest card so far in the suit led", which is directly executable, in contrast to the given advice.

(c) Learning by analogy. This can be applied when analogies are detected between new desired knowledge and already possessed knowledge. For example, if someone knows how to rent a bike (search yellow pages, call for information, search for money, etc.), and (s)he wants to rent a car, then (s)he can reason by analogy in order to learn the new task.

This field of machine learning is very new; the first interesting paper ([WINST80]) appeared in December 1980. So far, no significant results have been reported in this area.

(d) Learning from examples. This can be applied when the knowledge provided by the teacher is of lower level than that used by the learned. Consider, for example, the case in which several examples and/or counterexamples of connected graphs (in discrete mathematics) or of 5-card hands containing at least one pair (in poker) are shown to a learner. The learner might want to generalize them, i.e. to learn what a connected graph is and what a pair is.

This is the kind of learning performed in our thesis, and for this reason it is examined in more detail in the next section.

(e) Learning from observation and discovery (also called unsupervised learning). This is the most difficult kind of learning, since the learner acquires knowledge without the help of any teacher. The most successful program of this kind in Lenat's AM ([LENAT82]), which re-discovered quite a lot of number theory results, having started only with some basic concepts of set theory. ■

1.5. Learning from Examples

Learning from examples is, by far, the most extensively studied area of machine learning, and therefore, any attempt to review it adequately would be vain. The interested reader is referred to [COFEI82], [COCKB85], and [LACAR84] for an introduction and a critical survey; [FORSY84] also discusses the same subject.

In this section, the restricted area of **single concept** learning from examples (thereafter referred simply as: learning from examples) will be discussed, since this is the topic of our own work.

Learning from examples is defined as the task in which

> given a set of examples and counterexamples of a concept, a learner induces a general concept description that describes all of the examples and none of the counterexamples.
>
> ([CAMIM83], p. 9)

The examples of the above definition are usually called positive examples (PEs), and the counterexamples negative examples (NEs). An example is either a PE or a NE. ■

Clearly, learning from examples is a form of inductive learning, since it processes by hypothesizing a general description from examples. As such, it **does not preserve truth.**

Clearly also, the above definition preassumes that there exist both an instance language, in which all examples are describable, and a generalization language, in which all candidate concepts to be learned are describable. ■

The most common use of learning from examples is that of classification; a machine is supposed to learn a concept from a set of training instances, so that it can, later on, classify an instance (not necessarily encountered before) as a positive or a negative one. Data compression is another candidate application domain of learning from examples, although, so far, nobody has used learning for this purpose. ■

H. Simon and G. Lea first observed ([SILEA74]) that learning from examples can be viewed as a search of the rule space (or hypothesis space), i.e. the set of all candidate concepts to be learned, guided by training PEs and NEs, drawn from the instance space. All recent work on learning from examples views the learning problem as a search problem. ■

Learning from examples has advanced along several directions, depending on the way each researcher resolved the numerous issues which exist in the field.

Some of these issues are concerned with the representation of the examples and the concepts. Which is the instance language? Which is the generalization language? Is the single-representation trick employed, i.e. is the instance language a subset of the generalization language*? Is the generalization language biased, i.e. does it describe only a proper subset of all existing concepts**? If yes, how is the problem of new terms going to be solved, i.e. how can the rule space be augmented so

----

* When the single representation trick is employed, the rule space can be searched more easily ([COFEI82], pp. 368-369).

** The use of a biased generalization language reduces the size of the rule space to be searched, but it may cause problems if the concept to be learned is not contained in the (reduced) rule space. The issue of selecting biases by hand is discussed in [MITCH80]; in [UTGOF83] and in [UTMIT82] it is proposed that the learning program itself should select the biases.

that the concept to be learned will be included in it?

Some other issues in learning from examples are concerned with the way examples become available to the learner*. Are only PEs, or only NEs, or both PEs and NEs available to the learner? Is there an ORACLE, i.e. does the learner have the freedom to ask the teacher whether a certain instance is a PE or a NE, and can the teacher give an answer**? Is the order of presentation of the examples important? Can the examples be any examples, or do they have to be illustrative examples? Are all the examples available at once, that is, can the learner examine any of them at any time, or are they shown one by one? ■

The generalization language (i.e. propositional logic) used in our work is biased. In Chapters 4 and 5 it will be guaranteed that the generalization language considered can describe the concept to be learned, but in Chapters 6, 7, and 8 the problem of the new terms will be addressed. Our learning algorithms have no access to any ORACLEs. Depending also on the learning task, only PEs, or only NEs,

----

* Examples, in their relation to learning systems and in their own right, are discussed in [RISSL83].

** The availabiltiy of ORACLEs can speed up the search process; if the learner asks the proper question, (s)he can eliminate half of the rule space.

or both PEs and NEs may be required. The order in which the examples are presented is immaterial, and no restrictions concerning their illustrativity are imposed. Finally, our learning algorithms need never to re-examine an example already seen. ■

1.6. Noise

One more issue, concerning the task of learning from examples, is that of the presence or not of noise. Due to its great importance, it is examined separately in this section.

It can be easily realized that in real-life applications, PEs and NEs presented to a learning system are not expected to be error-free. Depending on the particular application domain of the learning algorithm, the sources of noise may vary. Humans may be subjective in characterizing a person as "smart", for example, or they may be wrong in listing the symptoms of a disease; errors may be physically generated over a transmission line; maliciously noisy data may be fed into a system by an enemy; and if something is incorrectly learned and it is going to be used for future knowledge acquisition, it is going to affect the system's behavior.

The last point is worth emphasizing. Since not too many things can be learned from scratch, the knowledge base of an expert system is gradually increasing; always, a large amount of knowledge already possessed by the system is used to obtain a new, small piece of knowledge. For example, a system may be first taught the concept of a "pair" in poker, and then it may use this knowledge to learn the concept of a "full" (three of a kind and a pair). If errors are introduced when the system is learning the "pair" concept, then the "full" concept (to be learned subsequently) will be unavoidably affected. In cases like that, the study of a system's behavior in the presence of noise, gives information about error propagation inside the system.

Although the need for research in learning from noisy data has been acknowledged a long time ago, it is only recently that some partial results have been reported by Quinlan ([QUINL83]) and Segen ([SEGEN85]). L. G. Valiant also, studied the problem of boolean concept learning from noisy data ([VALIA85]). He determined the conditions under which approximate learning can be performed, and he also found that his algorithms take polynomial time. ■

All problems addressed in our thesis will be solved both when the PEs and/or NEs are error-free (Chapters 4 and 6), as well as when they are noisy (Chapters 5, 7, and 8). ■

1.7.- Mitchell's Version Space

T. M. Mitchell's Ph.D. work (first version in [MITCH77] and [MITCH79]; more polished version in [MITCH82]) is described in this section, since it provides a unified framework for describing and comparing several learning algorithms. It will be used in Chapter 4 to describe our stated algorithms in a different way, and also to explore other possible learning strategies.

Mitchell observed that any rule space is partially ordered by the "more-general-than" relation; a concept C_1 is more-general-than a concept C_2 if and only if all PEs of C_2 are also PEs of C_1.

As an example, consider the case in which each instance is a ball which

(i) is either large or small, and

(ii) has two hemispheres, each of them painted either blue or red.

Every such instance can be described by a triplet, such as {blue, red, large} (which is equal to {red, blue, large}), or {blue, blue, small}. The rule space is described by similar triplets, with the additional freedom that a * may appear in any of the three places, indicating that the color or the size of the ball is insignificant. The partially ordered rule space is shown in Figure 1.1. Notice that the generalization language is biased, since the rule "both hemispheres have the same color" is not included in the rule space. Clearly also, the single-representation trick has been employed.

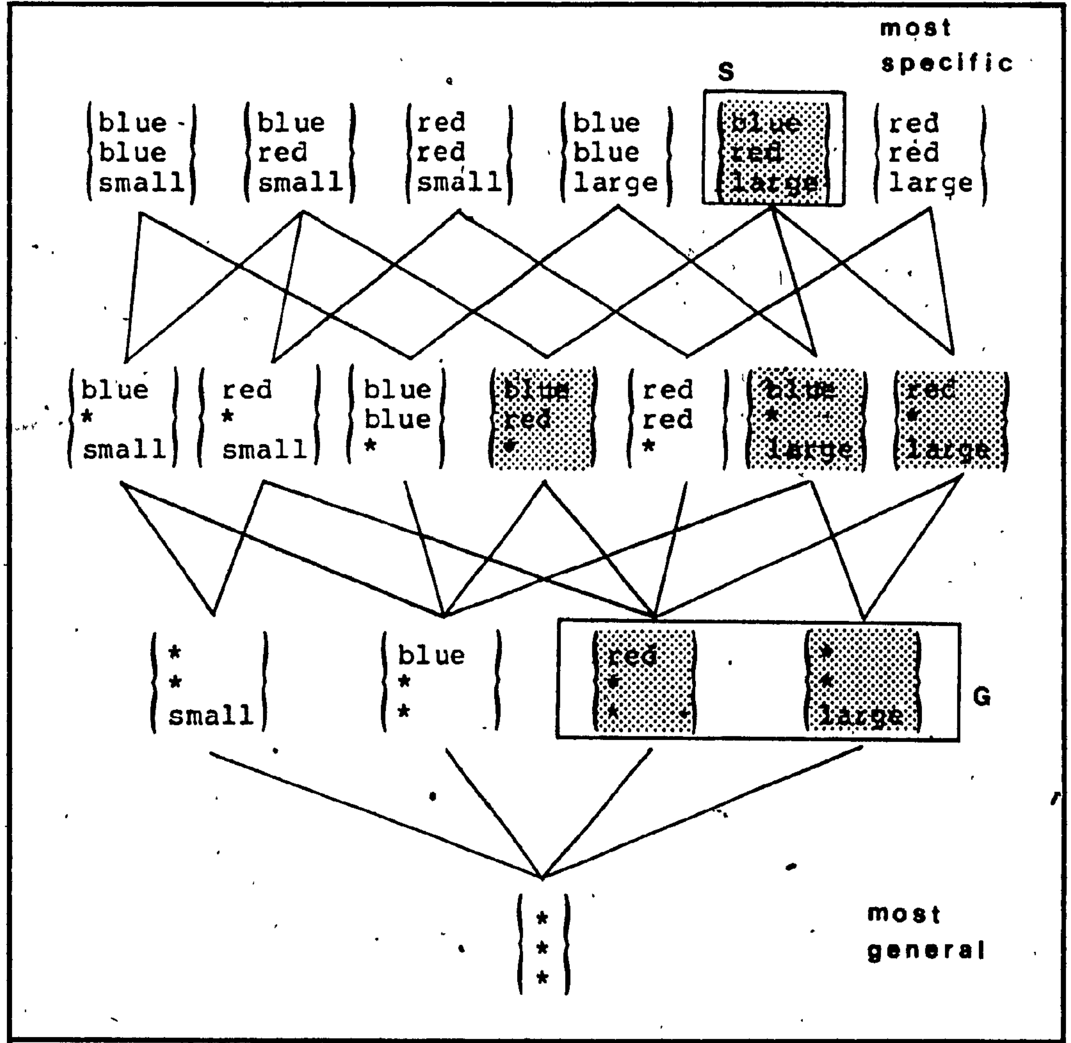


Figure 1.1. Delimiting sets S and G of version space.

The partial ordering of the rule space allows a very compact representation of the (possibly large) set of all plausible (i.e. candidate to be learned) rules. This is achieved by using the set G of the most general rules, and the set S of the most specific rules. For example, if the shaded rules of Figure 1.1 form the set of all plausible rules, then this set can be represented by the 2-element set G and the 1-element set S, also shown in Figure 1.1.

Mitchell's learning method consists of maintaining the sets G and S, delimiting the version space, which is nothing but the set of all rules consistent with all observed instances. G and S are initialized to the set of all instances and to the null set, respectively, and they are updated only when absolutely necessary. If a PE (NE), which is not contained (is contained) in some element of S (G), is seen, then the elements of S (G) are generalized (specialized) as little as possible, so that the new PE (NE) is contained (is not contained) in the updated S (G). What is very important is that each PE (NE) always causes **the most conservative generalization (specialization).** For example, the PE {blue, red, large} will result in the S of Figure 1.1, whereas the NE {blue, blue, small} will result in the G of Figure 1.1.

This algorithm is practical only when an efficient matching test of an instance and a rule exists, and when an efficient computation of the "more-general-than" relation between two rules can be made. Both of those requirements are fulfilled by the example of Figure 1.1; the first one is fulfilled because of the single-representation trick employed, and the second one because of the generalization language used. Clearly also, the algorithm is applicable only when the concept to be learned can be expressed in the generalization language used.

The main advantages of this algorithm is that its output, S and G, gives a concise representation of all plausible answers, it never needs to backtrack, and its performance does not depend on the order of the presentation of the examples. Clearly though, it will not do well if the examples are noisy. ■

1.8. Problems in Learning from Examples; Valiant's Idea

In Section 1.5 it was stated that learning from examples can be viewed as a search of the rule space. Most of the work done by 1984 was concerned with the improvement of those search methods. The ultimate goal was always to find

> the unique concept in the rule space that best covers all of the positive and none of the negative instances.
>
> ([COFEI82], p. 383)

Moreover,

> most work to date [i.e. 1982] assumes that if enough instances are presented, exactly one concept exists that is consistent with the training instances.
>
> ([COFEI82],p. 383)

However, in most of the cases, a huge rule space has to be searched, and therefore, the "enough instances" of the quotation above are actually "too many". That results in a prohibitively large execution time required by any search method. Since that was realized quite early, not much attention was paid to finding and comparing the time complexities of various search methods*; even the fastest algorithm was quite slow. The comparison of different learning algorithms (equivalently: search methods) was done, if at all, in terms of other issues. T. M. Mitchell, for example, suggested that

> in comparing alternative strategies, the important issues concern capabilities [e.g. ability to detect when the concept has been learned] rather than efficiency.
>
> ([MITCH82], p. 218)

* [MITCH82] and [DIMIC83] are two of the few exceptions.

One of the attempts made to overcome the problem of the large number of examples required for learning was to perform "learning from one example". DeJong's "explanatory schema acquisition" technique ([DEJON83]), applicable to natural language understanding, Minton's "constraint-based generalization" technique ([MINTO85]), applicable to game-playing, and Silver's method ([SILVE83]), learning equation solving techniques, are three examples of this kind.

If "learning from one example" sounds too impressive, or even impossible, the researchers themselves have an answer. DeJong confesses that his program

> can acquire a new concept based on only one input example . . . although later inputs may result in refinement of learned concepts.
>
> ([DEJON83], p. 462)

Minton also admits that

> after being tought approximately fifteen examples, the program plays go-moku at a level that is better than novice, but not expert . . . it seems likely that the program could be brought to the expert level by teaching it perhaps fifteen more examples.
>
> ([MINTO85], p. 14)

According to those statements, the idea of "learning from one example" does not seem to be brand new (this is not related to the scientific value of these papers, which is not negligible). ■

L. G. Valiant moved in a different direction. He observed that the problem he wanted to solve, that of learning a boolean expression from a set of examples, required exponential time in the worst case. So, instead of trying to optimize the searching method, that is, instead of trying to find a better exponential algorithm, he tried to find a different, but still reasonable, notion of learnability, which would allow him to learn (according to his definition) a boolean expression in polynomial time.

What he proposed was that a boolean expression should be considered to be learned if just the most common PEs (NEs) are included in (excluded from) it. That is to say, a PE (NE) with very low probability to be seen does not have to be included in (excluded from) the learned concept*. ■

Valiant's work is improved on and expanded in our thesis. ■

----

* [KOORP81] describes another, but finally unsuccessful, attempt for approximate learning.

1.9. Analysis of Algorithms

According to what was stated near the end of the last section, some sort of analysis of (the new) algorithms has to be performed in order to determine whether the new learning algorithms, following Valiant's idea, are indeed better than the old ones. Several issues concerning the analysis of algorithms, in general, are discussed in this section. They provide the context of the analysis of algorithms performed in our thesis. ■

We tend to believe that no algorithm is useful if it is not accompanied by an analysis, allowing its comparison with other algorithms solving the same or similar problems.

Many aspects of an algorithm can be examined through its analysis. Here are some of them.

(a) Time complexity. That is, the time required for the execution of the algorithm.

(b) Space complexity. That is, the amount of storage required for the execution of the algorithm.

(c) Generality, or applicability. That is, the conditions under which the algorithm is applicable. Different algorithms solving the same problem, for example, might tolerate different amount of noise in their input data.

(d) Output quality and reliability. That is, the closeness of an approximate solution to the exact solution(s), and the frequency with which a correct or an acceptable solution will be produced. This is particularly important for algorithms solving difficult problems (e.g. primality test, graph problems, etc.). A fast algorithm giving a/the right answer 95% of the time, for example, might be preferred over a slow algorithm giving the right answer 100% of the time.

Talking about the time complexity of an algorithm, three cases are always distinguished.*

(a) Best-case. That is, the execution time required for the most favorable input, i.e. the minimum time within which the algorithm can be completed.

(b) Average-case. That is, the average of the execution times for all possible inputs. The computation of that assumes knowledge of the distribution on the set of inputs.

(c) Worst-case. That is, the time required for the most unfavorable input, i.e. the maximum time required for the algorithm's execution.

----

* The definitions given here hold for deterministic algorithms; for probabilistic algorithms, i.e. for algorithms making random choice(s) at certain point(s) during their execution ([RABIN77] and [KARP77] are the first papers on this subject) different definitions have to be given.

The best-case time complexity is not of great importance, for obvious reasons. The worst-case time complexity is the one which shows the limitations of an algorithm, whereas the average-case time complexity is more useful, provided of course that the input distribution is known. ■

The analysis of any of the four aspects of an algorithm, mentioned in the beginning of this section, can be either theoretical or empirical.

To make an empirical analysis of an algorithm, one has to gather some statistics about the actual execution time of the algorithm, the space it used, etc., for several inputs.

On the other hand, to make a theoretical analysis of the time complexity of an algorithm, one has to express its execution time in terms of the input size of the problem (e.g. the number of integers to be sorted, or the dimensions of two matrices to be multiplied), rather than in real time. This is achieved by expressing the number of basic operations (e.g. number of comparisons, or number of multiplications of two integers) as a function of the input size, for the best-, the average-, or the worst-case. Similarly, to make a theoretical analysis of the space complexity, the applicability, or the reliability of an algorithm, one has to express results in terms of perhaps several of the input's parameters.

It is not to be thought however, either that both of these analyses are applicable in all cases, or that both of them convey the same amount and quality of information.

An empirical analysis will unavoidably be heavily dependent on the algorithms's implementation, that is, on the computer and the programming language used, the quality of the code written, etc. A theoretical analysis, on the other hand, will yield more elegant, concise, and general results (in addition to being an intellectual activity itself). There are cases, of course, in which the difficulty of analyzing a designed algorithm is prohibitive. In those cases an empirical analysis is (temporarily) justified. ■

Having now, discussed the four aspects of an algorithm which might be usefully analyzed, as well as the two ways in which this analysis can be performed, let's see which aspects of our own algorithms have been analyzed in this thesis, and how this analysis has been performed.

A great emphasis has been given to the analysis of the worst-case time complexity of our algorithms; the analysis of the average-case time complexity cannot be done, since no particular input distribution is assumed, and the best-case analysis is of no interest.

Both the applicability and the reliability of our algorithms have also been studied, since without them even the description of the algorithms would be of no use. Great effort has been put into achieving tight results for these two aspects, and considerable improvements over Valiant's results have been finally achieved.

In contrast to these three aspects, the space complexity of our algorithms has not received much attention, because the analysis is quite straightforward, and the space complexity itself is not high enough to require improving (see remarks after Algorithm 4.1).

The last topic to be discussed is that of the choice of a theoretical or an empirical analysis of our algorithms. Due to the already mentioned advantages of a theoretical analysis, and due also to the manageable complexity of the analysis, a theoretical analysis has been pursued. ■

1.10. Thesis Organization

As already stated in previous sections, our thesis' work improves and extends L. G. Valiant's results on boolean concept learning from examples. The material presented in this thesis is organized as follows.

Chapters 2 and 3 contain the mathematical prerequisites for the analysis of the algorithms, as well as some definitions required for the formal description of the results.

Chapters 4 and 5 are based on Valiant's work, presented in [VALIA84] and [VALIA85]; Chapter 4 examines the problem of learning from error-free data, and Chapter 5 the problem of learning from noisy data. Due to our improved (compared with Valiant's) results of Chapters 2 and 3, our own algorithms in Chapters 4 and 5 are significantly better than Valiant's, in many aspects.

Chapters 6, 7, and 8 address a brand new problem. Algorithms for boolean concept learning from examples are developed, which have less information about the concept to be learned than the algorithms of Chapters 4 and 5.

Chapter 9 concludes the thesis with a summary of the obtained results, and suggestions for further research. ■

1.11. Summary

The topic of our thesis, that is, the problem of approximate boolean concept learning from examples, was described in this chapter. This is an AI problem, examined here from a theoretical standpoint. For these reasons, some AI background (on machine learning), as well as some theoretical background (on analysis of algorithms) has been given. ■

Due to the non-trivial analysis of our algorithms, several mathematical results are required. They are all stated in the next chapter, and motivate the inclusion of the chapter at this point in the thesis. ■

CHAPTER 2

MATHEMATICAL PREREQUISITES AND FUNDAMENTAL THEOREMS FOR THE ANALYSIS OF OUR ALGORITHMS

2.1. Introduction

This chapter contains some mathematical results, mainly from calculus and probability theory, necessary for the analysis of our algorithms. Additional results, from the boolean algebra area, will be given in Chapter 3.

Some of the results stated in this chapter are already well-known, and they are briefly mentioned in Section 2.2. Some others, presented in Section 2.3, had to be established, due to particular problems encountered during the design and analysis of our algorithms. Because of that, their usefulness cannot be justified now. L. G. Valiant, in [VALIA84a], also looked at some of those problems, but the answers he got are worse than ours. The chapter closes with a summary in Section 2.4. ■

Since not even the formal definition of the problem addressed in our thesis has been given yet, the entire

Chapter 2 can be omitted in the first reading, without loss of continuity. ■

2.2. Preliminaries

The standard notation used in the analysis in algorithms, as well as a number of useful and well-known theorems from calculus and probability theory are presented in this section. The proofs of these theorems have been omitted, since they can be found in any text. Some non-standard probability theory definitions and notation have also been included here. ■

2.2.1. Asymptotic Notation

An exact expression for the time complexity of an algorithm, already discussed in Section 1.9, is not always the ultimate aim of an analysis. Although the derivation of such an expression may be a challenging job, the expression itself may be too complicated to be informative; it may contain too many details, not necessarily affecting the algorithm's time complexity, since, anyway, only the number of basic operations is counted, rather than the actual execution time. Because of that, and especially in cases in which the derivation of an exact formula is tedious, an approximate expression, correct within an order of magnitude, is what is sought.

An asymptotic notation is one way of formally describing the order of magnitude. The notation proposed by D. E. Knuth in [KNUTH76] has been adopted and extended here.

Let $f(u_1,u_2,\ldots,u_n)$ be a positive function of n positive real variables. Then:

Definition 2.1.

$$f(u_1,\ldots,u_n) = O(g(u_1,\ldots,u_n))$$

if and only if there are constants c, $u_1',u_2',\ldots,u_n' > 0$, such that $f(u_1,\ldots,u_n) \leq cg(u_1,\ldots,u_n)$, for all $u_i \geq u_i'$, for $i=1,2,\ldots,n$.

Definition 2.2.

$$f(u_1,\ldots,u_n) = \Omega(g(u_1,\ldots,u_n))$$

if and only if there are constants c, $u_1',u_2',\ldots,u_n' > 0$, such that $f(u_1,\ldots,u_n) \geq cg(u_1,\ldots,u_n)$, for all $u_i \geq u_i'$, for $i=1,2,\ldots,n$.

Definition 2.3.

$$f(u_1,\ldots,u_n) = \Theta(g(u_1,\ldots,u_n))$$

if and only if both $f(u_1,\ldots,u_n) = O(g(u_1,\ldots,u_n))$ and $f(u_1,\ldots,u_n) = \Omega(g(u_1,\ldots,u_n))$ are true. ■

For our purposes, the variables $u_1,\ldots,u_n$ will stand for the input size of the problem (e.g. one variable denoting the number of integers to be sorted is sufficient, whereas three variables denoting the dimensions of two matrices to be multiplied are required), and $f(u_1,\ldots,u_n)$ may be either the best-, average-, or worst-case time complexity expression. **Each of those three time complexities can be written by using any of the three asymptotic notations.**

One more remark is in place. Suppose, for example, that $f(u) = 3u^5+4u^3+7$ is the best-, average-, or worst-case time complexity of an algorithm. Then, all of $f(u)=O(u^5)$, $f(u)=\Omega(u^5)$, and $f(u)=\Theta(u^5)$ are obviously correct. However, it is also true that $f(u)=O(u^6)$, $f(u)=O(u^7),\ldots,$ and that $f(u)=\Omega(u^4),\ldots,f(u)=\Omega(1)$. Therefore, the O- and the Ω-notations might be misleading, although correct. For this reason it is preferable to **use the Θ-notation for any of the three time complexities.** If a Θ-expression cannot be found, or it is very complicated to be informative, it is advisable to give both an O- and an Ω-expression for the time complexity under consideration. This is what has been done in our thesis.

It has to be remembered though that **although the Θ-notation is the best one, it has to be used cautiously;** the values of the constants u', c, and c' for which

$$c'g(u) \le f(u) \le cg(u) \quad \forall u \ge u' \tag{2.1}$$

are very important. ■

2.2.2. Bounds of the Logarithmic Function

The following inequality is known from calculus:

$$1-\frac{1}{x} \leq \ln x \leq x-1 \qquad \forall x>0. \qquad (2.2)$$

Equality on both sides holds when x=1. ■

2.2.3. The Quadratic Inequality

The roots of the quadratic formula $F(x) = Ax^2+Bx+C$, for any real numbers A, B, and C, are given by

$$x_{1,2} = \frac{-B\pm\sqrt{B^2-4AC}}{2A}.$$

Suppose that $x_1 \leq x_2$. Then, F(x) takes the sign of A for all x, such that $x<x_1$ and $x_2<x$. F(x) takes the opposite sign of A for all x, such that $x_1<x<x_2$. ■

2.2.4. Expected Value of a Random Variable

Let X be a discrete random variable with probability function:

$$P[X=x_i] = p_i \qquad \forall i=1,2,...$$

Then, the expected value of X is defined as

$$E(X) = \sum_{i\geq 1} p_i x_i. \qquad (2.3)$$

Let X be a random variable as above, and Y=f(X) be another random variable. Then

$$E(Y) = \sum_{i\geq 1} p_i f(x_i). \tag{2.4}$$

If $X_1, X_2, \ldots X_n$ are independent random variables, then

$$E\left(\prod_{i=1}^{n} X_i\right) = \prod_{i=1}^{n}\left(E(X_i)\right). \tag{2.5}$$

Let X be a random variable, and $X>0$ (i.e. X takes only positive values). Then

$$P[X\geq a] \leq \frac{E(X)}{a} \quad \forall a>0. \tag{2.6}$$

This is known as *Markov's inequality* (also called *extended* (or *generalized*) *Chebyshev's inequality*). ■

2.2.5. Bernoulli Trials

A sequence of trials, each of them having only two possible outcomes A and A', is called a sequence of *Bernoulli trials.* Usually, we call A a *success*, and A' a *failure*.

According to our definition, the probability of success does not have to be the same for all trials, implying that **the trials may not be independent.** Although the Bernoulli trials are usually assumed to be independent, we don't follow this convention, because non-independent Bernoulli trials will play a central role in the analysis of our algorithms. The description of both cases is given below. ■

I. <u>Classical Bernoulli trials.</u> Let X be a random variable denoting the number of successes in n independent Bernoulli trials, each having probability of success p, and probability of failure q=1-p. It is then said that X follows the <u>binomial distribution</u> with parameters n and p. Obviously, X can take the values 0,1,2,...,n. To indicate the dependence of X on n and p, we write b(k; n, p) instead of P[X=k], for all k=0,1,...,n. It is known that

$$b(k;\ n,\ p) = \binom{n}{k} p^k q^{n-k} \qquad \forall k=0,1,\ldots,n. \qquad (2.7)$$

If X_1, X_2,... X_n are random variables with probability functions

$$P[X_i=1] = p,\quad P[X_i=0] = q = 1-p \quad \forall i=1,2,\ldots,n,$$

then the random variable

$$X = \sum_{i=1}^{n} X_i$$

follows the binomial distribution, with paramenters n and p, i.e.

$$P\left[\sum_{i=1}^{n} X_i = k\right] = b(k;\ n,\ p) \qquad \forall k=0,1,\ldots,n. \quad (2.8)$$

Since the cumulative distribution function $P[X \leq k]$ will play a central role in our work, the following notation has been used:

$$b(\leq k;\ n,\ p) = P[X \leq k] = \sum_{i=1}^{k} b(i;\ n,\ p). \qquad (2.9)$$

II. Lower-bounded Bernoulli trials. Let Y be a random variable denoting the number of successes in n Bernoulli trials, each having probability of success at least p. Obviously, Y can take the values 0,1,2,...,n. By $b^*(k;\ n,\ \geq p)$ we will denote the $P[Y=k]$*.

Those Bernoulli trials differ from the classical ones in that their exact success probabilities might not be known; only a lower bound for them is known. That means that either all Bernoulli trials are equiprobable (but only a lower bound of their success probability is known), or that each Bernoulli trial has a different success probability (with a known lower bound for the entire set of Bernoulli trials). Clearly, the last case allows non-independent Bernoulli trials to be considered.

One more remark is in place. $b^*(k;\ n,\ \geq p)$ is not a

* Although the notation $b(k;\ n,\ \geq p)$ would be sufficient to distinguish those Bernoulli trials from the classical ones, the $b^*(k;\ n,\ \geq p)$ notation will be used for more emphasis.

unique number, as is the case with $b(k;\ n,\ p)$. For each set $\{p_1,\ldots,p_n\}$, such that $p_i \geq p$, for $i=1,\ldots,n$; if $q_i=1-p_i$, then

$$b^*(k;\ n,\ \geq p) = \sum p_{i_1}\cdots p_{i_k} q_{i_{k+1}}\cdots q_{i_n}, \qquad (2.10)$$

where the sum is taken over all permutations of $\{i_1,\ldots,i_n\} = \{1,\ldots,n\}$, such that $i_1<\ldots<i_k$ and $i_{k+1}<\ldots<i_n$.

For example, if $n=4$ and $k=2$, then for any set $\{p_1,p_2,p_3,p_4\}$ with $p_i \geq p$ for $i=1,2,3,4$, $b^*(2;\ 4,\ \geq p)$ will be equal to

$$p_1p_2q_3q_4+p_1p_3q_2q_4+p_1p_4q_2q_3+p_2p_3q_1q_4+p_2p_4q_1q_3+p_3p_4q_1q_2.$$

Clearly, each $b^*(k;\ n,\ \geq p)$ is the sum of $\binom{n}{k}$ terms. Clearly also, what distinguishes two values of $b^*(k;\ n,\ \geq p)$ is the choise of $p_1,\ldots,p_n$; their assignments, as success probabilities, to particular trials is immaterial. That means that in the previous example, the value of $b^*(2;\ 4,\ \geq p)$ does not depend on which of p_1, p_2, p_3, p_4 is the success probability of the 1st, 2nd, 3rd, 4th trial, as long as each trial has one of p_1, p_2, p_3, p_4 as its success probability.

For the cumulative distribution function $P[Y \leq k]$, the following notation will be used:

$$b(\leq k;\ n,\ \geq p) = P[Y \leq k] = \sum_{i=1}^{k} b^{*}(i;\ n,\ \geq p). \quad (2.11)$$ ■

2.3. Useful Combinatorial Bounds

Some probability theory results, essential for the analysis of our algorithms, are established in this section. The most important of them are Theorems 2.1, 2.2, and 2.5.

The problem addressed in Theorems 2.1 and 2.2 was also addressed by L. G. Valiant, in [VALIA84a], but our results stated here are better than his. The problem addressed Theorem 2.5 was not considered by him.

In contrast to the last section, all results in this section are fully proven, since they cannot be found in the standard probability theory texts. However, none of the proofs is required for the understanding of the rest of the thesis; only the stated definitions and theorems have to be known.

The reader is warned once more that the usefulness of all definitions, lemmas, and theorems of this section will not become apparent until later chapters. Theorems 2.1 and 2.2, for example, will be used in Chapter 4, whereas Theorem 2.5 will be used in Chapters 5, 6, 7, and 8; Lemmas 2.1 to

2.7 are mainly required for the proof of the first two theorems, whereas Theorems 2.3 and 2.4 are required for the proof of Theorem 2.5. ■

The following notation will be used in this section, except if otherwise indicated. The symbols k, n, p, p', etc. will have the obvious meanings implied by the definition of the Bernoulli trials in Subsection 2.2.5. For example, the appearence of $b^*(\leq k;\ n,\ \geq p')$ in an expression will imply that $k \in N_0$, $n \in N$, $0 \leq p' \leq 1$, and $q'=1-p'$. Also, h, h_1, h_2, will be real numbers. ■

The next definition is more general than the one given by L. G. Valiant in [VALIA84a]*.

<u>Definition 2.4.</u> For all $h_1, h_2 > 1$**, and for all $k \in N_0$, we define

$$L_0(h_1,h_2,k) = \min\left\{ n \in N \mid b^*(\leq k;\ n,\ \geq h_1^{-1}) \leq h_2^{-1} \right\}. ■$$

The value of $L_0(h_1,h_2,k)$ dominates the time complexity of our algorithms. Roughly speaking, if t is the input size of the problem, then many of our algorithms have time complexity $\Theta(L_0(h_1,h_2,t^k))$, for some h_1, h_2, and k.

* Valiant has set $h=h_1=h_2$.

** The definition can be given for $h_1>1$ and $h_2 \geq 1$. If k N, the definition can be extended for $h_1,h_2 \geq 1$.

Clearly, the computation of $L_0(h_1,h_2,k)$ is of great importance.

Unfortunately though, no analytic formula is known for $b(\leq k;\ n,\ h^{-1})$, and even less for $b^*(\leq k;\ n,\ \geq h^{-1})$. Therefore, there is not much hope of finding the exact value of $L_0(h_1,h_2,k)$. So, our aim will be to find un upper bound of $L_0(h_1,h_2,k)$. Such a bound is given in Theorems 2.1 and 2.3. Needless to say, the tighter that bound will be, the faster our algorithm will be shown to become. In order to establish our theorems, several lemmas are required first. ■

<u>Lemma 2.1.</u> If $p\leq p'$, then

$$b(k;\ n,\ p') \leq b(k;\ n,\ p) \quad \text{for } k\leq np.$$

Proof: According to (2.7), it suffices to show that

$$(p')^k(1-p')^{n-k} \leq p^k(1-p)^{n-k} \quad \text{for } k\leq np, \qquad (2.12)$$

where $n,k\in N$, and $p'\geq p$.

Consider the function

$$f : [0,1] \dashrightarrow R \quad \text{with } f(p)=p^k(1-p)^{n-k}.$$

It can be verified that

$$f'(p) = (k-np)p^{k-1}(1-p)^{n-k-1},$$

and therefore

$$f'(p) \leq 0 \quad \text{for } k\leq np,$$

which implies that

$$f(p) : \text{decreasing in } [0,1] \text{ for } k\leq np.$$

That establishes (2.12), and the result follows. ■

Lemma 2.2.

$$b^*(k;\ n,\ \geq p) \leq b(k;\ n,\ p) \qquad \text{for} \quad k \leq np.$$

Proof[*]: If all Bernoulli trials are equiprobable, then Lemma 2.2 reduces to Lemma 2.1.

If not all Bernoulli trials are equiprobable, then it suffices to prove that each member of the set defined in (2.10) is at most b(k; n, p). For simplicity, we will consider the case in which n-1 Bernoulli trials have probability of success p, and one trial has probability of success p', for some $p' \geq p$. In that case,

$$b^*(k;\ n,\ \geq p) = \binom{n-1}{k}p^k q^{n-k-1}q' + \binom{n-1}{k-1}p^{k-1}q^{n-k}p'.$$

Therefore, it suffices to show that

$$\binom{n-1}{k}p^k q^{n-k-1}q' + \binom{n-1}{k-1}p^{k-1}q^{n-k}p' \leq \binom{n}{k}p^k q^{n-k}$$
$$\text{for } k \leq np \text{ and } p' \geq p, \qquad (2.13)$$

or, equivalently,

$$\binom{n-1}{k}pq' + \binom{n-1}{k-1}qp' \leq \binom{n}{k}pq \qquad \text{for } k \leq np,\ p' \geq p.$$

By using elementary properties of the binomial coefficients, the last inequality can be rewritten as

----

[*] An alternate proof would be to show that the function

$$f : [p,1] \dashrightarrow R \quad \text{with}$$
$$f(p') = \binom{n-1}{k}p^k q^{n-k-1}(1-p') + \binom{n-1}{k-1}p^{k-1}q^{n-k}p'$$

is decreasing for $k \leq np$. The proof is easy, and is similar to that of Lemma 2.1

$$\binom{n-1}{k}pq'+\binom{n-1}{k-1}qp' \le \binom{n-1}{k}pq+\binom{n-1}{k-1}pq \quad \text{for } k\le np,\ p'\ge p$$

which is equivalent to

$$\binom{n-1}{k}p(q'-q) \le \binom{n-1}{k-1}q(p-p') \quad \text{for } k\le np,\ p'\ge p,$$

and since $q'-q = p-p' \le 0$, the last can be rewritten as

$$\binom{n-1}{k}p \ge \binom{n-1}{k-1}q \quad \text{for } k\le np,$$

or

$$\frac{p}{k} \ge \frac{1-p}{n-k} \quad \text{for } k\le np,$$

which is true, as it can be easily verified. That proves (2.13).

This proof can be generalized if the n Bernoulli trials have success probabilities $p_1,p_2,\ldots,p_n$. ■

Lemma 2.3. (known as Chernoff's bound; see [PAPOU84], p. 121). Let X be a random variable, and $k\in N_0$. Then

$$P[X\ge k] \le e^{-kt}E(e^{tX}) \quad \forall t\ge 0$$

and

$$P[X\le k] \le e^{-kt}E(e^{tX}) \quad \forall t\le 0.$$

Proof: Let $t\in R$. Consider the random variable $Y=e^{tX}$ and the number $a=e^{kt}$. Clearly, $Y>0$ and $a>0$. Therefore, Markov's inequality in (2.6) is applicable for Y and a, and it yields

$$P[e^{tX}\ge e^{tk}] \le \frac{E(e^{tX})}{e^{tk}}.$$

The result now follows. ■

Lemma 2.4.

$$b(\leq k;\ n,\ p) \leq e^{-kt}(pe^{t}+q)^{n} \qquad \forall t\leq 0.$$

Proof: Let $X_1, X_2, \ldots, X_n$ be random variables with probability functions

$$P[X_i=1] = p, \text{ and } P[X_i=0] = q = 1-p, \quad \forall i=1,\ldots,n.$$

According to (2.8), the random veriable

$$X = \sum_{i=1}^{n} X_i \tag{2.14}$$

follows the binomial distribution, with parameters n and p. Therefore

$$P[X\leq k] = b(\leq k;\ n,\ p). \tag{2.15}$$

Lemma 2.3 implies that

$$P[X\leq k] \leq e^{-kt}E(e^{tX}) \qquad \forall t\leq 0.$$

According to (2.14), (2.15), and (2.5), the last inequality can be rewritten as

$$b(\leq k;\ n,\ p) \leq e^{-kt}\left[E\left(e^{tX_i}\right)\right]^{n} \qquad \forall t\leq 0. \tag{2.16}$$

But (2.4) implies that

$$E\left(e^{tX_i}\right) = pe^{t}+q,$$

since X_i takes only the values 0 and 1. The last inequality, together with (2.16) prove the lemma. ■

Lemma 2.5. For all $h_1, h_2 > 1$, and for all $k\in N_0$,

$$\min\left\{n\in N \mid b(\leq k; n, h_1^{-1}) \leq h_2^{-1}\right\} \leq \left\lceil \frac{k+\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)} \right\rceil .$$

Proof: According to Lemma 2.4

$$\min\left\{n\epsilon N \mid b(\underline{<}k;n,h_1{}^{-1})\underline{<}h_2{}^{-1}\right\}\underline{<}$$

$$\underline{<}\ \min\left\{n\epsilon N \middle| e^{-kt}\left(\frac{e^t}{h_1}+1-\frac{1}{h_1}\right)^n \underline{<}\ h_2{}^{-1},\text{ for any } t\underline{<}0\right\}. \quad (2.17)$$

But it can be verified that the inequality

$$e^{-kt}\left(\frac{e^t}{h_1}+1-\frac{1}{h_1}\right)^n \underline{<}\ h_2{}^{-1} \qquad \forall\, t\underline{<}0$$

is equivalent to

$$n \underline{>} \frac{-kt+\ln h_2}{\ln\left(\frac{h_1}{e^t+h_1-1}\right)} \qquad \forall\, t\underline{<}0, \qquad (2.18)$$

which, for t=-1, yields

$$n \underline{>} \left\lceil\frac{k+\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)}\right\rceil,$$

since n must be an integer. The last inequality now, together with (2.17), prove the lemma. ■

<u>Lemma 2.6.</u>

$$h \underline{<} \frac{1}{\ln\left(\frac{eh}{eh+1-e}\right)} \qquad \forall\, h\underline{>}1.$$

Proof: Consider the function

$$f : [1,\infty) \dashrightarrow R \quad \text{with} \quad f(h) = h\ln\left(\frac{eh}{eh+1-e}\right). \quad (2.19)$$

It can be verified that

$$f'(h) = \ln\left(\frac{eh}{eh+1-e}\right) - \frac{e-1}{eh+1-e}.$$

Application now of (2.2) yields

$$\ln\left(\frac{eh}{eh+1-e}\right) \leq \frac{e-1}{eh+1-e} .$$

Therefore, $f'(h) \leq 0$ is true, or equivalently,

$$f(h) : \text{decreasing in } [1, \infty). \tag{2.20}$$

It can be also seen that

$$\lim_{h \to 1} f(h) = 1. \tag{2.21}$$

But (2.20) and (2.21) imply that

$$f(h) \leq 1 \quad \forall h \in [1, \infty),$$

which, according to (2.19), completes the proof. ■

<u>Lemma 2.7.</u>

$$k \leq \frac{k+\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)} h_1^{-1} \qquad \forall k \in N_0, \forall h_1, h_2 \geq 1.$$

Proof: Since for $k=0$ the lemma is obviously true, it suffices to prove that

$$\frac{k+\ln h_2}{kh_1 \ln\left(\frac{eh_1}{eh_1+1-e}\right)} \geq 1 \qquad \forall k \in N, \forall h_1, h_2 \geq 1. \tag{2.22}$$

Consider the function

$$f : [1, \infty) \to R \text{ with } f(k) = \frac{k+\ln h_2}{kh_1 \ln\left(\frac{eh_1}{eh_1+1-e}\right)} \tag{2.23}$$

It can be easily verified that

$$f'(k) = -\frac{\ln h_2}{k^2 h_1 \ln\left(\frac{eh_1}{eh_1+1-e}\right)} \leq 0,$$

which implies that

$$f(k) : \text{decreasing in } [1, \infty). \tag{2.24}$$

It is also obvious that

$$\lim_{k\to\infty} f(k) = \frac{1}{h_1 \ln\left(\frac{eh_1}{eh_1+1-e}\right)}$$

which, according to Lemma 2.6 implies that

$$\lim_{k\to\infty} f(k) \geq 1. \tag{2.25}$$

But (2.24) and (2.25) imply that

$$f(k) \geq 1 \quad \forall k \in [1, \infty),$$

which, according to (2.23), proves (2.22). That completes the proof. ■

Our first main result can now be established.

<u>Theorem 2.1.</u> Let $h_1, h_2 > 1$. Then

$$L_0(h_1, h_2, k) \leq \left\lceil \frac{k + \ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)} \right\rceil \quad \forall k \in N_0.$$

Proof: By setting $p = h_1^{-1}$, and $k = 0, 1, 2, \ldots$ in Lemma 2.2, and by adding the resulting inequalities, we get

$$b^*(\leq k; n, \geq h_1^{-1}) \leq b(\leq k; n, h_1^{-1}) \qquad \text{for } k \leq nh_1^{-1}.$$

According now to Definition 2.4, the proof is immediate from Lemmas 2.5 and 2.7. ■

Near the beginning of this section, it was stated that $L_0(h_1, h_2, k)$ dominates the time complexity of our algorithms. Therefore, its upper bound, given by Theorem 2.1, will be overused in our thesis. For brevity, we will denote it by $L(h_1,h_2,k)$. More precisely, we have:

Definition 2.5. For all $h_1,h_2>1$, and for all $k\in N_0$, we define:

$$L(h_1,h_2,k) = \left\lceil \frac{k+\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)} \right\rceil \quad \forall k\in N_0. \quad \blacksquare$$

Obviously, Theorem 2.1 can now be restated as:

$$L_0(h_1,h_2,k) \le L(h_1,h_2,k) \quad \forall k\in N_0. \tag{2.26}$$ $\blacksquare$

Since k, h_1, and h_2 will characterize the input size of the problem to be solved, our next step is to find an asymptotic expression for $L(h_1,h_2,k)$.

Theorem 2.2. For all $h_1,h_2>1$, and for all $k\in N_0$,

$$h_1(k+\ln h_2) \le L(h_1,h_2,k) \le \frac{e}{e-1} h_1(k+\ln h_2)+1.$$

Therefore,

$$L(h_1,h_2,k) = \Theta(h_1(k+\ln h_2)).$$

Proof: (2.2) implies that

$$\frac{1}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)} \le \frac{e}{e-1} h_1 \quad \forall h_1\ge 1,$$

and, according to Lemma 2.6,

$$h_1 \leq \frac{1}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)} \leq \frac{e}{e-1}\, h_1 \qquad \forall h_1 \geq 1. \qquad (2.27)$$

The result now follows by multiplying the last inequality by $k+\ln h_2$. ■

Keeping always in mind the importance of the upper bound of $L_0(h_1,h_2,k)$, given by Theorem 2.1 (see remarks after Definition 2.4), let's pause for a while, in order to see what improvements we have made over Valiant's results, in [VALIA84a]. He was able to show that

$$\min\left\{n \in N \mid b^*(\leq k;\ n,\ \geq h^{-1}) \leq h^{-1}\right\} \not\leq 2h(k+\ln h).$$

First of all, the fact that he used only h, instead of h_1, and h_2, made his algorithms less flexible (see Chapter 4), and also deprived him of the necessary tools for finding algorithms learning from noisy data as efficient as ours (see Chapter 5). Anyway, if we make the necessary modifications in his proof, in order to include h_1 and h_2, and we add the ceiling function he forgot, we will finally obtain

$$L_0(h_1,h_2,k) \leq L'(h_1,h_2,k) = \left\lceil 2h_1(k+\ln h_2)\right\rceil. \qquad (2.28)$$

Clearly, according to Theorem 2.2, our bound $L(h_1,h_2,k)$ is asymptotically equal to Valiant's bound $L'(h_1,h_2,k)$ in (2.28). However, we have made some improvements.

Since $\frac{e}{e-1} \simeq 1.58$, Theorem 2.2 implies that our bound is better than Valiant's by at least 20%. We say "at least 20%", instead of just "20%", because this difference of 20% is between his $L'(h_1,h_2,k)$, and the upper bound of our $L(h_1,h_2,k)$. If we go back to the proof of Theorem 2.2, we will see that this upper bound was derived from (2.27), and therefore, according to what said in Subsection 2.2.2

$$L(h_1,h_2,k) \dashrightarrow 1.58h_1(k+\ln h_2) \quad \text{when } h_1 \dashrightarrow \infty .$$

Therefore, for small values of h_1, $L(h_1,h_2,k)$ is strictly less than $1.58h_1(k+\ln h_2)$, which justifies our "at least 20%" statement. ■

The next question we can ask is whether an even better bound of $L_0(h_1,h_2,k)$ can be found. Definitely, no bound of $L_0(h_1,h_2,k)$ can be sublinear in k, since, by definition, $n \geq k$ must be true, which implies that $L_0(h_1,h_2,k) \geq k$ must be true as well.

What about h_1 and h_2? From (2.18) we can see, that a tighter bound of $L(h_1,h_2,k)$ could have been obtained if

$$\min_{t \leq 0} \left\{ \frac{-kt+\ln h_2}{\ln\left(\frac{h_1}{e^t+h_1-1}\right)} \right\}$$

could be computed. Unfortunately, we were unable to find this minimum.

But then, how was the choice of $t=-1$ made in (2.18)?

It can be shown (the proof is the same as in Theorem 2.2) that for all $k \in N_0$, $h \geq 1$, and for all **constant** $t \leq 0$

$$c_1h_1(k+\ln h_2) \leq \frac{-kt+\ln h_2}{\ln\left(\frac{h_1}{e^t+h_1-1}\right)} \leq c_2h_1(k+\ln h_2), \qquad (2.29)$$

where

$$c_1 = \begin{cases} -\frac{te^t}{1-e^t} & \text{if } -1 \leq t < 0 \\ \frac{e^t}{1-e^t} & \text{if } t \leq -1 \end{cases} \qquad (2.30)$$

and

$$c_2 = \begin{cases} \frac{1}{1-e^t} & \text{if } -1 \leq t \leq 0 \\ \frac{-t}{1-e^t} & \text{if } t \leq -1 \end{cases} \qquad (2.31)$$

This result implies that independently of the **constant*** value assumed by t, there is no asymptotic improvement of $L_0(h_1,h_2,k)$. The reason now for which the value of $t=-1$ is chosen is twofold. First, the expression for $L(h_1,h_2,k)$ becomes relatively simple. Second, the constant c_2, in (2.31), of the O-expression of $L_0(h_1,h_2,k)$ which can be obtained from (2.29), takes a value very close to its minimum. ■

----

* The question of obtaining a better bound of $L_0(h_1,h_2,k)$ by choosing a value of t depending on h_1, h_2, and k remains open.

Next, we are going to look at some additional results to be used in the analysis of our learning algorithms in Chapters 5, 6, 7, and 8.

We start by extending Definition 2.4 to include the case of a real r, instead of only an integer k.

Definition 2.6. For all $h_1, h_2 > 1$, and $r \in R_0^+$, we define

$$L_0(h_1,h_2,r) = \min\left\{n \in N \mid b^*(\leq r;\ n,\ \geq h_1^{-1}) \leq h_2^{-1}\right\}. \blacksquare$$

Theorem 2.1 now can be extended as follows:

Theorem 2.3. Let $h_1, h_2 > 1$. Then

$$L_0(h_1,h_2,r) \leq \left\lceil \frac{r + \ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)} \right\rceil \quad \forall r \in R_0^+.$$

Proof: Obviously

$$b^*(\leq r;\ n,\ \geq h_1^{-1}) = b^*(\leq \lfloor r \rfloor;\ n,\ \geq h_1^{-1}) \quad \forall r \in R_0^+.$$

Definition 2.6 now implies that

$$L_0(h_1,h_2,r) = L_0(h_1,h_2,\lfloor r \rfloor) \quad \forall r \in R_0^+,$$

which according to Theorem 2.1 implies that

$$L_0(h_1,h_2,r) \leq \left\lceil \frac{\lfloor r \rfloor + \ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)} \right\rceil \quad \forall r \in R_0^+.$$

Since now

$$\lfloor r \rfloor \leq r \quad \forall r \in R_0^+,$$

the result follows. $\blacksquare$

Definition 2.5 can be extended in the obvious way for the bound of Theorem 2.3.

<u>Definition 2.7.</u> For all $h_1, h_2 > 1$, and for all $r \in R_0^+$, we define

$$L(h_1, h_2, r) = \left\lceil \frac{r + \ln h_2}{\ln\left(\frac{eh_1}{eh_1 + 1 - e}\right)} \right\rceil . \qquad ■$$

Obviously, Theorem 2.3 can now be restated as

$$L_0(h_1, h_2, r) \le L(h_1, h_2, r) \quad \forall r \in R_0^+. \qquad (2.32)$$

■

For the sake of completeness, the following theorem is stated.

<u>Theorem 2.4.</u> For all $h_1, h_2 > 1$, and all $r \in R_0^+$

$$L(h_1, h_2, r) = \Theta(h_1(r + \ln h_2)).$$

Proof: As in Theorem 2.2. ■

The last two theorems are now making the computation of an upper bound of

$$\min\left\{ n \in N \mid b^*(\le dn;\ n,\ \ge h_1^{-1}) \le h_2^{-1} \right\} \qquad (2.33)$$

for some $d \in R_0^+$, feasible. It is the computation of this bound which will enable us to design noisy data learning algorithms better than Valiant's (Chapter 5), and will also allow us to design our own algorithms (Chapters 6, 7, 8).

<u>Theorem 2.5.</u> Let $h_1, h_2 > 1$, and $d \in R$, such that

$$0 \leq d < \ln\left(\frac{eh_1}{eh_1+1-e}\right). \tag{2.34}$$

Then

$$\min\left\{n \in N \mid b^*(\leq dn; n, \geq h_1^{-1}) \leq h_2^{-1}\right\} \leq \left\lceil \frac{d+\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)-d} \right\rceil$$

Proof: Let's call

$$m = \left\lceil \frac{d+\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)-d} \right\rceil. \tag{2.35}$$

According to Definition 2.7, if (2.34) holds, then

$$m = L\left(h_1, h_2, d\left(\frac{d+\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)-d}+1\right)\right) \tag{2.36}$$

can be verified. Theorem 2.3 now implies

$$b^*\left(\leq d\left(\frac{d+\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)-d}+1\right); m, \geq h_1^{-1}\right) \leq h_2^{-1}. \tag{2.37}$$

But

$$\lceil x \rceil \leq x+1 \qquad \forall x \in R, \tag{2.38}$$

which implies that

$$b^*(\leq d\lceil x \rceil; m, \geq h_1^{-1}) \leq b^*(\leq d(x+1); m, \geq h_1^{-1}) \quad \forall d, x \in R_0^+$$

Therefore, according to (2.35), (2.37) implies that

$$b^*(\leq dm; m, \geq h_1^{-1}) \leq h_2^{-1}. \tag{2.39}$$

Now, it is clear that it suffices to show that

$$b^*(\leq dm'; m', \geq h_1^{-1}) \leq h_2^{-1} \qquad \forall m' \geq m \tag{2.40}$$

is always true*. The proof will be done by induction on m'.

(i)**Basis:** For m'=m (2.40) reduces to (2.39), which is true.

(ii)**Inductive step:** Assume that

$$b^*(\le dk;\ k,\ \ge h_1^{-1}) \le h_2^{-1} \text{ for some } k \ge m' \ge m. \qquad (2.41)$$

It must be shown that

$$b^*(\le d(k+1);\ k+1,\ \ge h_1^{-1}) \le h_2^{-1}. \qquad (2.42)$$

Theorem 2.3 implies that

$$\min\left\{n\epsilon N \mid b^*(\le d(k+1);\ n,\ \ge h_1^{-1}) \le h_2^{-1}\right\} \le$$

$$\le \left\lceil \frac{d(k+1)+\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)} \right\rceil .$$

Therefore, in order to prove (2.42) it suffices to prove that

$$\left\lceil \frac{d(k+1)+\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)} \right\rceil \le k+1,$$

or, according to (2.38), it suffices to prove that

$$\frac{d(k+1)+\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)} \le k.$$

But it can be verified that this is equivalent to

$$\frac{d+\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)-d} \le k,$$

which, according to the values of k from (2.41) and to

[*] (2.40) cannot be derived directly from (2.36). It can be shown that any m'≥m makes (2.37) true, but that will not be sufficient; the quantity by which d is multiplied in (2.37) should not be constant.

(2.35), is always true.

That proves (2.40). Hence the proof of the theorem is complete. ■

Although the proof of the last theorem is complete, let's briefly explain how we derived it, i.e. how we found the bound of (2.33). If we call $L=L(h_1,h_2,k)$, then Theorem 2.3 implies that we should look for an L having the property

$$\frac{d(L+1)+\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)} = L \tag{2.43}$$

(we wrote L+1 instead of L, in order to take care of the ceiling function). The solution of (2.43) yields the bound of Theorem 2.5. ■

Since m, the upper bound in Theorem 2.5, defined in (2.35), will play an important role in the time complexity of our algorithms, let's see if we can derive an asymptotic expression for it, assuming that h_1 and h_2 are growing.

It can be immediately seen that d is an additive factor to m. Therefore

$$m \geq \frac{\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)},$$

and, according to Lemma 2.6,

$$m = \Omega(h_1 \ln h_2). \tag{2.44}$$

Looking now for an O-expression, we can first observe that d cannot stay as a paramenter in this asymptotic expression for m, since Theorem 2.5 implies that as h_1 increases, the upper bound of d decreases. That means that if d is fixed, h_1 cannot grow beyond a certain value. To be more precise, if

$$d < d_{max} = \ln\left(\frac{eh_1}{eh_1+1-e}\right),$$

then $d_{max}(h_1)$ is decreasing. Also, according to (2.27)

$$\frac{e-1}{e}\frac{1}{h_1} \leq d_{max} \leq \frac{1}{h_1},$$

and moreover, as h_1 increases, d_{max} reaches its lower bound.

Those remarks imply that if an O-expression is to derived for m, that must be done under the assumptions that

$$d < \frac{c}{h_1} \quad \text{for} \quad c < \frac{e-1}{e}. \tag{2.45}$$

In that case, according to (2.27), (2.35) implies that

$$m \leq \frac{\frac{c}{h_1}+\ln h_2}{\frac{e-1}{eh_1}-\frac{c}{h_1}},$$

which implies that

$$m \leq \frac{e}{e-1-ec} h_1\ln h_2 + \frac{ec}{e-1-ec}. \tag{2.46}$$

If c in (2.46) is treated as a constant, then (2.46) implies that

$$m = O(h_1\ln h_2). \tag{2.47}$$

However, c should not be considered as a constant, for two reasons. First, c is an indicator of the different values d can take. Second, when $c \longrightarrow \frac{e-1}{e}$ (see (2.45)), then for the coefficient of the $O(h_1 \ln h_2)$ in (2.47), $\frac{e}{e-1-ec} \longrightarrow \infty$ is true*, hence (2.47) is rather misleading.

If, on the other hand, c is treated as a parameter in (2.46), then the implied expression

$$m = O\left(\frac{e}{e-1-ec}(h_1 \ln h_2 + 1)\right) \tag{2.48}$$

taken together with the conditions in (2.45) is not more informative than m itself.

For these reasons, m itself is going to be used in our thesis, instead of (2.44), and (2.47), or (2.48). ■

2.4. Summary

Several mathematical results were presented in this chapter. They were originally compiled from the literature or derived afresh only when they were seen to be necessary for the analysis of the algorithms. Although it would be more natural to present them in the thesis in this order too, for practical reasons we preferred to include all of

* That should be expected from the definition of m in (2.35); when d approaches its maximum permissible value, then $m \longrightarrow \infty$.

them in a single chapter, instead of having them scattered throughout the thesis.

As stated in Chapter 1, the starting point of our thesis is L. G. Valiant's work on concept learning. However, because of the improvements we have made in his algorithms (Chapter 5), and also because of the new algorithms we have developed (Chapters 6, 7, and 8), some problems not even addressed by Valiant have been answered here (e.g. Theorem 2.5), whereas for some other problems examined by him we obtained better answers (e.g. Theorems 2.1 and 2.2). ■

To be fully prepared to design and study boolean expression learning algorithms, precise definitions of several boolean algebra terms have to be given, as well as a reasonable definition of learnability. This is the subject of the next chapter. ■

CHAPTER 3

BOOLEAN CONCEPT LEARNING

3.1. Introduction

The formal framework for learning boolean expressions, the problem addressed in our thesis, is given in this chapter. The definitions and theorems stated here will be used throughout the thesis.

Although much of this material is due to L. G. Valiant, all terms have been rigorously defined, and all theorems have been proven in detail. This has been done for several reasons. First of all, since Valiant's work appeared in three different sources ([VALIA84a], [VALIA84b], and [VALIA85]), the terminology used is inconsistent. Second, the fact that new algorithms have been designed, and Valiant's algorithms have been improved/corrected in our thesis, necessitated some new definitions and theorems (e.g. Definition 3.9, and Theorems 3.1, 3.2, and 3.3), as well as the modification of some old ones (e.g. Definitions 3.15 and 3.16). Third, the inclusion of all necessary material in this chapter, made the thesis self-contained.

Section 3.2 contains all definitions required for the formal description of our results. Three theorems, stated in Section 3.3, and not appearing in Valiant's work, will be very useful in our subsequent work. The example generators, used to communicate knowledge from the teacher to the learner, as well as the new notion of approximate learning, all of them due to Valiant, are defined in Section 3.4. Finally, the chapter closes with a summary in Section 3.5. ■

3.2. Boolean Algebra Terminology

The definitions of all boolean algebra terms used in our thesis are given in this section. Some of them (e.g. Definition 3.8) have been especially tailored for our needs, i.e. to faciliate the proof of a theorem, to describe more concisely our results, etc. Some others (e.g. Definitions 3.6 and 3.7) are widely used in boolean algebra theory, but they are also stated here, because the terminology in this area has not been standardized yet.

Throughout this section, $x_1,x_2,...,x_t$ will denote boolean variables over the two-valued boolean algebra $\{0,1\}$. That means that each x_i, for $i=1,2,...,t$ can take either the value 0 or the value 1. It is assumed that the definitions of the complement ('), the sum (+), and the product (*) for this two-valued boolean algebra, as well as their properties (e.g. $x_i(x_i+x_j) = x_i$)* are known. ■

We start by giving the definitions concerning the representation of the concepts (to be) learned.

Definition 3.1. A literal is either a variable, or its complement. ■

Definition 3.2. A boolean expression is either a literal, or any expression built up from literals, by using the + and * operators.

The items of Definitions 3.4 to 3.9 below ase examples of boolean expressions. ■

Definition 3.3. A boolean function of t variables is a function $F : \{0,1\}^t \dashrightarrow \{0,1\}$.

For example, if t=2, then the function $F : \{0,1\}^2 \dashrightarrow \{0,1\}$ such that $F(0,0)=0$, $F(0,1)=1$, $F(1,0)=0$, and $F(1,1)=0$ is a boolean function.

Any boolean function can be written as a boolean expression. For example, the function F above can be written as $F(x_1,x_2)=x_1'x_2$. ■

* When no operator appears, multiplication is assumed.

The terms "boolean function", "boolean expression", "boolean concept", and "concept" will be used interchangeably throughout our thesis. ■

Definition 3.4. A fundamental product (fundamental sum) is either a literal, or a product (sum) of literals, with at most one literal from each variable.

For example, x_2', $x_1x_3'x_4$ are fundamental products, whereas $x_1x_3'x_1'$ is not. Also, x_2', $x_1+x_3'+x_4$ are fundamental sums, whereas $x_1+x_3'+x_1'$ is not. ■

Definition 3.5. A k-product (k-sum) is a fundamental product (sum) of at most k literals. ■

Definition 3.6. A Disjunctive Normal Form or DNF (Conjunctive Normal Form or CNF) is either a fundamental product (sum), or a sum (product) of fundamental products (sums).

For example, $x_1+x_3'+x_4$, $x_1x_3'x_4$, $x_1+x_1x_2'$, and $x_1x_3'+x_1'x_2x_4$ are DNFs. Also, $x_1+x_3'+x_4$, $x_1x_3'x_4$, $x_1(x_1+x_2')$, and $(x_1+x_3')(x_1'+x_2+x_4)$ are CNFs.

Any boolean function can be written in DNF and in CNF. This representation though is not unique. In general, the equivalence of two DNFs/CNFs is not easy to be detected. ■

Definition 3.7. A minimal DNF (minimal CNF) is a DNF (CNF) representation of a function such that:

(a) No other DNF (CNF) representation has fewer terms, i.e. fewer fundamental products (sums).

(b) Any other DNF (CNF) representation with the same number of terms will not have fewer occurences of literals.

For example, since $x_1+x_1x_2=x_1$, both expressions are DNFs of the same boolean function, but only x_1 is minimal DNF. Also, since $x_1x_2+x_1'x_2'+x_2x_3' = x_1x_2+x_1'x_2'+x_1'x_3'$, both expressions are minimal DNFs of the same boolean function. Similarly, x_1 is minimal CNF, but not $x_1(x_1+x_2)$, and $(x_1+x_2)(x_1'+x_2')(x_2+x_3')$ and $(x_1+x_2)(x_1'+x_2')(x_1'+x_3')$ are both minimal CNFs of the same boolean function. ■

Definition 3.8. A k-DNF (k-CNF) is either a k-product (k-sum) or a sum (product) of k-products (k-sums).

For example, $x_1+x_3'+x_4$ is a k-DNF for $k\geq1$, $x_1x_3'x_4$ is a k-DNF for $k\geq3$, and $x_1x_2'+x_1$ is a k-DNF for $k\geq2$, although it can be rewritten as x_1, which is a 1-DNF as well. Also, $x_1x_3'x_4$ is a k-CNF for $k\geq1$, $x_1+x_3'+x_4$ is a k-CNF for $k\geq3$, and $(x_1+x_2')x_1$ is a k-CNF for $k\geq2$, although it can rewritten as x_1, which is a 1-CNF as well.

Notice that we are not talking about a function being in k-DNF (k-CNF), but about an expression (i.e. a particular representation of a function) being in k-DNF (k-CNF). The reason for that should be clear from the last example. ■

<u>Definition 3.9.</u> A <u>maximal k-DNF</u> (<u>maximal k-CNF</u>) is a k-DNF (k-CNF) representation of a function F, having the form $F=p_1+...+p_m$ ($F=s_1*...*s_m$), where each p_i (s_i) is a k-product (k-sum), for $i=1,...,m$, and also having the property that there is no k-product (k-sum) p_0 (s_0) different from $p_1,...,p_m$ ($s_1,...s_m$), such that $F=F+p_0$ ($F=F*s_0$).

For example, the maximal 2-DNF of the function $F(x_1,x_2)=x_1'$ is $x_1'+x_1'x_2+x_1'x_2'$, whereas the maximal 2-CNF of the same function is $x_1'(x_1'+x_2)(x_1'+x_2')$.

Clearly, in contrast to the minimal DNF (CNF), there exists a unique **maximal k-DNF (k-CNF)** represenation for every function. ■

A different representation of a boolean function of a small (usually at most 6) number of variables can be achieved by means of a <u>Karnaugh map.</u> The construction of Karnaugh maps is assumed to be known, but as a reminder, an example is given in Figure 3.1. Given the Karnaugh map of a function, a DNF (CNF) expression for the function is obtained by taking the sum (product) of the properly named groups of 1's (0's) in the map. Naming the groups of 1's

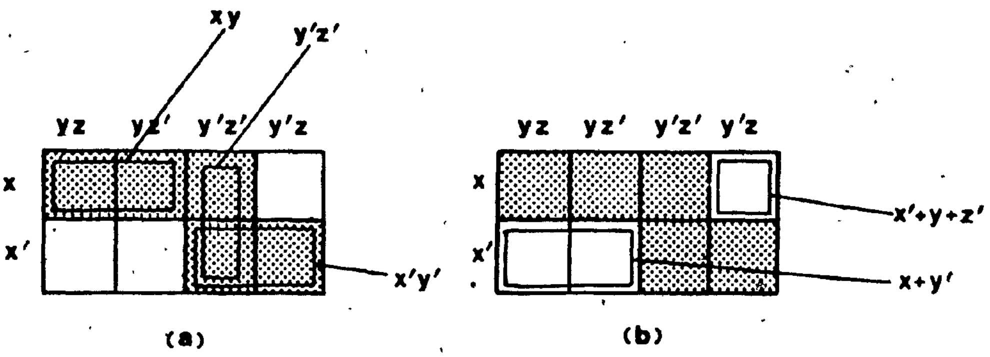


Figure 3.1. Karnaugh map of a function F(x,y,z).

F(x,y,z) = xy+x'y'+y'z' in DNF, from (a).

F(x,y,z) = (x+y')(x'+y+z') in CNF, from (b).

(0's) is done by taking the product (sum) of the labels (the complements of the labels) of the rows and columns forming the particular group.

Karnaugh maps are pictorial representations of boolean functions, and since there is always a **unique Karnaugh map** for every boolean function, they will be extensively used in our thesis as examples, to justify the design decisions made, to illustrate the significance of our results, etc. ■

As a first example, let's see how a Karnaugh map of a k-DNF expression of t variables looks like. Clearly, a group of a single 1 requires t literals (one from each variable) to be expressed as a product; a group of 2 1's requires t-1 literals, a group of 4 1's requires t-2 literals, etc. Therefore, in general, if a function can be expressed in k-DNF, then all groups of 1's in its Karnaugh map contain at least 2^{t-k} 1's.

Reasoning similarly, we can conclude that if a function can be expressed in k-CNF, then all groups of 0's in its Karnaugh map contain at least 2^{t-k} 0's.

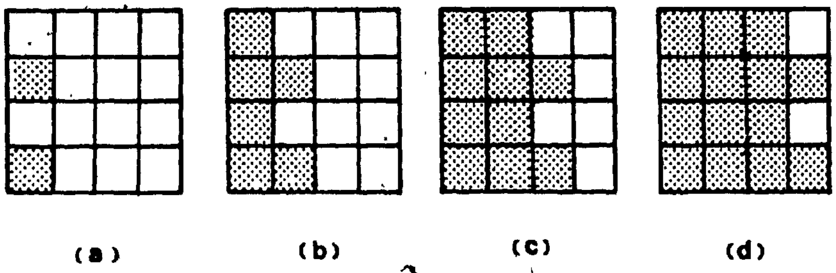


Figure 3.2. Karnaugh maps of four different boolean functions.

An example for the case of 4 variables is shown in Figure 3.2. The function in Figure 3.2(a) requires a 4-DNF, but only a 2-CNF. The functions in Figures 3.2(b) and 3.2(c) can be written in 3-CNF and 3-DNF, whereas the function in Figure 3.2(d) can be written in 2-DNF, but it requires a 4-CNF.

From this example it should be clear that the smaller k is in a k-DNF (k-CNF) expression, the larger groups of 1's (0's) exist in the expression's Karnaugh map. ■

We close this section by giving the definitions concerning the representation of the examples to be encountered by the learning algorithm.

Definition 3.10. A total vector is an element of $\{0,1\}^t$.

Total vectors can be written as fundamental products of exactly t literals. For example, for t=3, v=(1,0,0) is a total vector, meaning that $x_1=1$, $x_2=0$, and $x_3=0$. It can also be written as $v=x_1x_2'x_3'$. ■

Definition 3.11. A partial vector is an element of $\{0,1,*\}^t-\{0,1\}^t$.

Each partial vector has at least one * as one of its coordinates, signifying that the particular coordinate can be either 0 or 1. Partial vectors can also be written as fundamental products. For example, for t=3, v=(1,*,0) is a partial vector, meaning that $x_1=1$, x_2=undefined, and $x_3=0$, and it can also be written as $v=x_1x_3'$. ■

Definition 3.12. A vector is an element of $\{0,1,*\}^t$, i.e. either a total vector, or a partial vector. ■

Definition 3.13. For every vector v, its associated set of total vectors, denoted by T_v, is the set of all total vectors resulting from v by retaining all 0's and 1's of v, and by changing each * of v to either a 0 or a 1.

For example, if v=(1,*,0) is a partial vector, then T_v = {(1,1,0), 1,0,0)}. If v is a total vector, then always T_v={v}. Clearly, any partial vector v is nothing but a short way of writing all total vectors contained in T_v. ■

Given a boolean function F and a total vector v, exactly one of F(v)=1 and F(v)=0 is true. But this is not the case if v is a partial vector. The following definition extends the concept of boolean functions over all vectors.

Definition 3.14. For any boolean function F : $\{0,1\}^t$ --> {0,1}, its associated extended boolean function F' : $\{0,1,*\}^t$ --> {0,1,*} is defined as follows:
For any vector v
(a) F'(v)=1, if and only if F(w)=1 $\forall w \in T_v$.
(b) F'(v)=0, if and only if F(w)=0 $\forall w \in T_v$.
(c) F'(v)=*, otherwise.

For example, if F'(x_1,x_2)=x_2, then F'(*,1)=1, F'(*,0)=0, F'(1,1)=1, F'(1,0)=0, and F'(1,*)=*.

Clearly, boolean functions take only total vectors as arguments, whereas extended boolean functions can take either total or partial vectors. However, it can be immediately seen that if F is a boolean function and v is a total vector, then F(v)=F'(v). Therefore, we can interchangeably talk about a boolean function F and its associated extended boolean function F'. In the rest of the thesis, the term "boolean function" will be used. ■

3.3. Important Theorems

Three important theorems, established by us, are stated in this section. They will allow us to make some critical remarks, to correct, and to improve Valiant's algorithms (Chapters 4 and 5), and to design our own algorithms (Chapters 6, 7, and 8). As is also the case with Chapter 2, the entire section can be omitted in the first reading. ■

<u>Theorem 3.1.</u> Let s be a k-sum, p be a k-product, v be a vector, and $\tilde{v}$ be the vector obtained from v by complementing all its literals. If s, p, v, and $\tilde{v}$ are represented as sets (each set containing the literals specified in each k-sum/k-product/vector), then

(a) $\bigl(p(v) = 1\bigr) \Leftrightarrow \bigl(p \subseteq v\bigr)$.

(b) $\bigl(p(v) = 0\bigr) \Leftrightarrow \bigl(p \cap \tilde{v} \neq \emptyset\bigr)$.

(c) $\bigl(s(v) = 1\bigr) \Leftrightarrow \bigl(s \cap v \neq \emptyset\bigr)$.

(d) $\bigl(s(v) = 0\bigr) \Leftrightarrow \bigl(s \subseteq \tilde{v}\bigr)$.

Proof: Easy. In order to prove (b) for example, assume that $p=u_1u_2\ldots u_m$, and $v=w_1w_2\ldots w_n$. Then

$$p(v)=0$$

is equivalent to

at least one of $u_1,\ldots,u_m$ is made 0 by v

which is equivalent to

at least one of $u_1,\ldots,u_m$ is one of $w_1',\ldots,w_n'$

which is equivalent to

$$p\cap\bar{v}\neq\emptyset\ .$$ ■

Theorem 3.2. Let $p_1,\ldots,p_m$ be k-products, $F = p_1+\ldots+p_m$ be in k-DNF, and v be a vector.

(a) If F is in maximal k-DNF, or if v is a total vector, then

$$\big(F(v)=1\big)\Leftrightarrow\big(\exists i\in\{1,\ldots,m\}\ :\ p_i(v)=1\big).$$

(b) If F is not in maximal k-DNF, and v is a partial vector, then only the "$\Leftarrow$" implication above holds.

(c) Always

$$\big(F(v)=0\big)\Leftrightarrow\big(p_i(v)=0\quad\forall i\in\{1,\ldots,m\}\big).$$

Proof:

(a) "$\Leftarrow$" Trivial.

"$\Rightarrow$" If v is a total vector, it will make each of F's products either 0 or 1, therefore in order for F(v)=1 to hold, at least one of F's products must be made 1 by v.

If F is in maximal k-DNF, then:

(i) If v is a product of at most k literals, it can be easily verified that $F=F+v$ [Indeed, let u be a total vector. If $F(u)=1$, then $F(u)+v(u)=1$, hence $(F+v)(u)=1$. If, on the other hand, $F(u)=0$, then $v(u)=0$, because if $v(u)\neq 0$, then $v(u)=1$, and since $F(v)=1$, we would have $F(u)=1$, which contradicts our assumption that $F(u)=0$; therefore, $(F+v)(u)=0$]. Clearly, $F+v$ is in k-DNF. Since now F is in maximal k-DNF, v must be one of the $p_1,\ldots,p_m$. Let $v=p_i$, for some $i\in\{1,\ldots,m\}$. Then $p_i(v)=1$.

(ii) If v is a product of more than k literals, it can be similarly shown that all vectors resulting form v by keeping at most k of its literals are also in F.

(b) Let $F(x_1,x_2)=p_1+p_2$, where $p_1=x_1'x_2$ and $p_2=x_1'x_2'$, and also let $v=x_1'$. Then $F(v)=1$, but $p_1(v)=*$ and $p_2(v)=*$.

(c) "$\Leftarrow$" Trivial.

"$\Rightarrow$" The proof will be done by contradiction. Suppose that

$$\exists i\in\{1,2,\ldots,m\} : p_i(v)\neq 0 .$$

Then, according to Definitions 3.13 and 3.14

$$\exists i\in\{1,2,\ldots,m\}, \text{ and } \exists v_0\in T_v : p_i(v_0)=1.$$

Part (a) now of this theorem implies that

$$F(v_0) = 1$$

which contradicts the fact that $F(v_0)=0$, derived from our assumption that F(v)=0. ∎

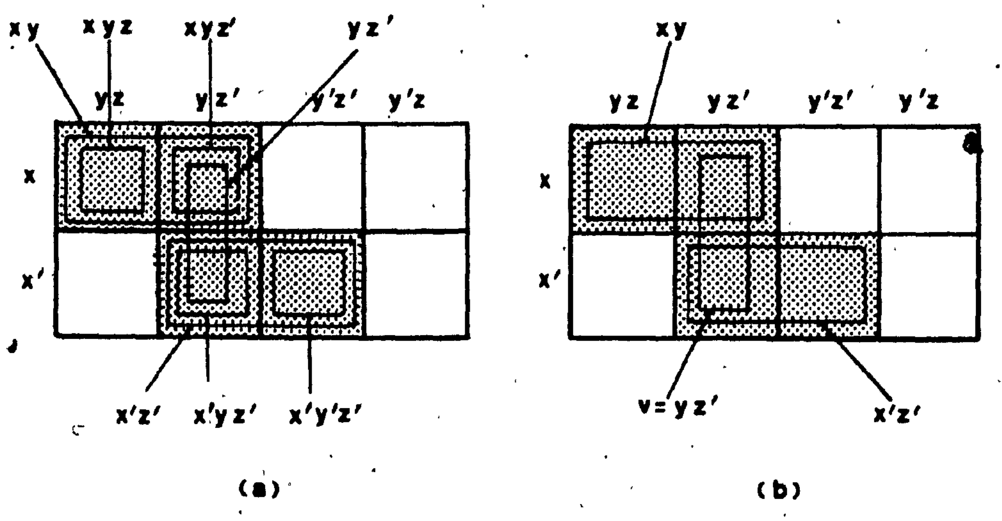


Figure 3.3. Illustration of Theorem 3.2.

The results of Theorem 3.2 can be better understood by looking at the Karnaugh map of the function F. For example, the function of Figure 3.3 can be written in maximal 3-DNF, as

$$F = xyz + xyz' + x'yz' + x'y'z' + xy + yz' + x'z',$$

as shown in Figure 3.3(a). It can also be written in non-maximal 2-DNF or 3-DNF as

$$F = xy + x'z',$$

as shown in Figure 3.3(b). As it can be easily seen, no matter whether F is in maximal 3-DNF or not, any total vector v making it 1 (i.e. any single square of the Karnaugh map falling inside F's region) will also make at least one of its products 1 (i.e. will also fall inside one of the groups of 1's, composing F). Similarly, if F is in maximal 3-DNF (i.e. if F is formed by all possible groups of 1's), then any vector v making it 1 (i.e. any square/rectangle falling inside F's region) will also make one of F's products 1 (i.e. will be or will fall inside one of the groups of 1's, composing F). However, as shown in Figure 3.3(b), this might not be true, if v is a partial vector and F is not in maximal k-DNF. ■

The next theorem is the dual of Theorem 3.2.

Theorem 3.3. Let $s_1,\ldots,s_m$ be k-sums, $F = s_1\ldots s_m$ be in k-CNF, and v be a vector.

(a) If F is in maximal k-CNF, or if v is a total vector, then

$$\big(F(v)=0\big)\Leftrightarrow\big(\exists\, i\in\{1,\ldots,m\} : s_i(v)=0\big).$$

(b) If F is not in maximal k-CNF, and v is a partial vector, then only the "$\Leftarrow$" implication above holds.

(c) Always

$$\big(F(v)=1\big)\Leftrightarrow\big(s_i(v)=1 \;\;\forall i\in\{1,\ldots,m\}\big).$$

Proof: The proof is the dual of that of Theorem 3.2, and it is briefly sketched here.

(a) It can be shown that if v is a vector of at most (more than) k literals, then the sum consisting of the complements of all (of at most k) of its literals will be in F.

(b) Consider $F(x_1,x_2) = (x_1'+x_2)(x_1'+x_2')$, and $v=x_1$.

(c) It can be shown that if $s_i(v)\neq 1$ for some $i\in\{1,\ldots,m\}$, then that contradicts the assumption $F(v)=1$. ■

3.4. Learnability

L. G. Valiant had a new idea for learning from examples. Instead of trying to improve a provably (see Theorem 3.4) exponential algorithm, performing exact learning, i.e. learning the exact concept induced by all error-free PEs/NEs, he gave a new definition of learnability, which is still reasonable, and which also leaves some space for polynomial time learning algoritnms.

Several versions of his definition of learnability are given here. They are preceded by the required definitions of the example generators. ■

3.4.1. Necessity for Approximate Learning

The next theorem suggests that the old notion of exact learning is not satisfactory for our purposes.

Theorem 3.4. Exact learning of a concept from its examples, if no information about the concept is available, takes exponential time.

Proof: Clearly, in order to learn exactly a concept about which no a priori information is known, either all its PEs, or all its NEs have to be seen. But a concept of t variables can have up to 2^t PEs/NEs (if the examples have to be total vectors), or up to 3^t PEs/NEs (if the examples can be either total or partial vectors). In both cases, an exponential number of examples is required, in order to be guaranteed that the exact concept will be learned. Even if each of them takes O(1) time to be processed, the overall execution time is exponential. ■

In this theorem it was not stated whether the exponential time complexity refers to the best-, the average-, or the worst-case. Clearly, depending on the input, i.e. on the order of presentation of the examples, the time complexity may vary; the process of a particular PE or NE, for example, may shorten the process time of subsequent examples. That implies that best-, average-, and worst-case can be distinguished. However, since, independently of the input, an exponential number of examples has to be examined, **all best-, average-, and worst-cases take exponential time.** ■

The issue of whether only total, or both total and partial vectors can be presented as examples has arisen from the proof of the last theorem. Our decision, which will hold for the rest of our thesis, is to **allow both partial and total vectors to be presented as examples,** since some variables might not be significant for the description of the concept to be learned. The existence of a table, for example, is immaterial for the description of a bedroom, but it is essential for the description of an office. In such a case, the example generator should have the freedom to leave non-significant variables unspecified. ■

Attention has to be paid when reading Theorem 3.4, so that no conclusions other than the ones stated in it, are drawn. For example, Theorem 3.4 **does not** say that the exact concept will be learned if 3^t PEs or 3^t NEs are seen. In fact, **this is not true.** If no restrictions are imposed on the example generator*, no matter how many examples are seen, there is no guarantee that the exact concept will be learned; the example generator may simply refuse to produce

----

* Imposing certain restrictions on example generators is not unusual. P.H. Winston, for example, in his influential work on structural learning ([WINST70]; [WINST84], Chapter 11], assumes that in order to learn from a sequence of examples, the NEs have to be "near-misses", and the PEs have to be presented in some pedagogical order.

certain PEs/NEs.

If all PEs/NEs of the concept to be learned have the same probability to be seen, then the only kind of statement which could possibly be made is "if n PEs/NEs are seen, then **probably** the exact concept has been learned", where n is a number to be determined. Notice that the word "probably" in the last statement **is necessary**, since some PEs/NEs may not be seen.

However, the assumption of equiprobability made above is unrealistic. In practice, some PEs/NEs may be very common, and some others very rare. In such a case, even the last (already weak) statement cannot be made; the rare PEs/NEs probably will not appear, and therefore probably the exact concept will not be learned. That means that even if n PEs/NEs, for some $n \geq 3^t$ (i.e. even if an exponential number of PEs/NEs) are seen, even **the conclusion "probably the exact concept has been learned" cannot be drawn.**

The need to relax at least one of our two demands is imperative. Either learning the exact concept should not be our aim, or some information about it should be available in advance. In order to clearly define the context of our work, the demand of exact learning will be dropped first, i.e. a new definition of learnability will be given. ■

Before stating the new definition, let's make our intentions clear by giving an example. If $f=x_1$ is the concept to be learned from PEs, but 99% of its PEs have $x_2=0$, and it didn't happen that a PE with $x_2=1$, or $x_2=*$ has been seen, then learning the concept $g=x_1x_2'$ instead of f will be acceptable, since g approximates f quite well. In that case, our conclusion could be a statement of the form "if n PEs are seen, then probably a good approximation of f has been learned", where n is a number to be determined. Clearly, the word "probably" is necessary again, in order to cover the unfortunate case in which the example generator will not produce some common examples, as it is expected to do. ■

3.4.2. Error-Free Example Generators

A certain (possibly arbitrary) input distribution has to be assumed, in order for the words "probably" and "good approximation" in the last statement to be rigorously defined.

Definition 3.15. For any concept $f(x_1,...,x_t)$ to be learned, a probability distribution $\underline{D^+}$ is defined over all vectors v such that $f(v)=1$, and an other probability distribution $\underline{D^-}$ is defined over all vectors v such that $f(v)=0$. ■

Obviously,

$$\sum_{f(v)=1} D^+(v) = 1 \quad \text{and} \quad \sum_{f(v)=0} D^-(v) = 1 .$$

Clearly also, for every vector $v \in \{0,1,*\}^t$ at most one of $D^+(v)$, $D^-(v)$ is defined; if $f(v)=*$ (see Definition 3.14), then none of them is defined. ■

The distributions D^+ and D^- completely determine the frequencies with which error-free PEs and NEs are generated. More precisely, we have:

Definition 3.16. For every concept f to be learned, the example generator EXAMPLE$^+$ outputs a vector v, such that $f(v)=1$, with probability $p^+(v)$. The example generator EXAMPLE$^-$ outputs a vector v, such the $f(v)=0$, with probability $D^-(v)$. ■

Our definition of D^- and EXAMPLE$^-$ is different from the one given by L. G. Valiant. The reason for doing so, will be explained in Section 4.6. It has to be emphasized though that, following Valiant, no particular distribution is assumed over the set of PEs and NEs. All our results will be expressed in terms of D^+ and D^-.

Some examples of what each of those two example generators can/cannot produce are shown in Figure 3.4. E_1, E_2, and E_3 are three possible outcomes of EXAMPLE$^+$; their probabilities of appearance though are not related. E_4 is a

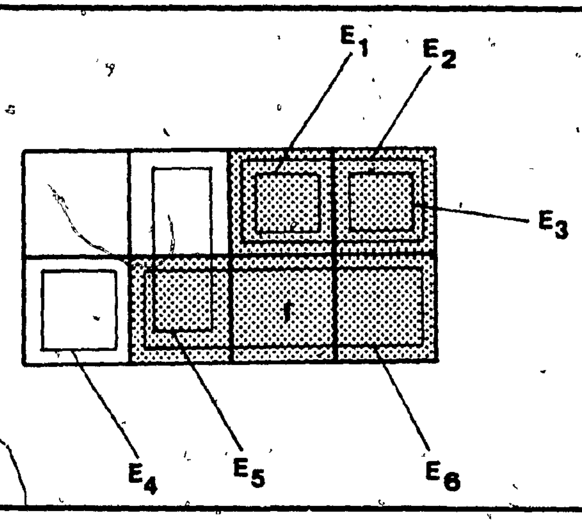


Figure 3.4. Permissible and non-permissible outcomes of EXAMPLE$^+$ and EXAMPLE$^-$.

possible outcome of EXAMPLE$^-$. E_5 cannot be output by either EXAMPLE$^+$ or EXAMPLE$^-$ because $f(E_5)=*$. E_6 also cannot be output by either of the example generators, because it cannot be represented as a vector in the first place. ■

3.4.3. Noisy Example Generators

Since, according to what was stated in Section 1.6, noisy rather than error-free data, should be expected in real life, a rigorous definition of the noisy example generators is in place.

Definition 3.17. If a positive (negative) example generator of a boolean concept f is making errors, by changing one or more of the coordinates of the vector it should output, with probability at most r, for some $0<r<1$, then it is called r-EXAMPLE$^+$ (r-EXAMPLE$^-$). ■

Notice that if, for example, v is the output of r-EXAMPLE$^+$ (similar remarks can be made if v is the output of r-EXAMPLE$^-$), then one of the following might happen:

(a) $f(v)=0$, i.e. v is actually a NE of f. If example generators were error-free, then v should be output only by EXAMPLE$^-$.

(b) $f(v)=*$, i.e. v is neither a PE nor a NE of f. If example generators were error-free, then v could never be presented as an example.

(c) $f(v)=1$, i.e. v is indeed a PE of f. Although in this case the error-free EXAMPLE$^+$ could also produce v as a PE, the error rate r is important, because it changes the frequency with which v is expected to be seen. To be more precise, the probability that v will be produced by EXAMPLE$^+$ is exactly $D^+(v)$, whereas the probability that the same v will be produced by r-EXAMPLE$^+$ is at least $(1-r)D^+(v)$*.

---

* Even if the error probability of the example generator is exactly r, the probability of generation of v will still be at least $(1-r)D^+(v)$; the probability that another vector v' will be erroneously changed to v has to be added to it.

Clearly, each vector can be output by any of r-EXAMPLE$^+$ and r-EXAMPLE$^-$. That was not the case with error-free example generators, where each vector could be output either only by EXAMPLE$^+$, or only by EXAMPLE$^-$, or by none of them. It has to be remembered though, that there are still groups of vectors, like E_6 of Figure 3.4, which cannot be output by either r-EXAMPLE$^+$ or r-EXAMPLE$^-$. ■

Let's try now to justify the given definition of the noisy example generators. First of all, a noisy example generator cannot be defined but in terms of an error-free example generator. That necessitates the introduction of r in Definition 4.17, since that expresses the relationship between the two example generators.

One of the controversial issues in this definition is the fact that each example is characterized as either erroneous or not, i.e. no notion of the amount of error made is introduced. The reason for doing so is twofold.

First, even for the simple case of the binary characterization of an example as erroneous or not, the study of the errors' effects is not easy at all, as it will be seen in Chapter 5.

Second, an appropriate measure of the amount of error made in each particular example is not easy to give. The "probability that n vector coordinates will be altered" could be a reasonable definition, capturing the amount of error made, but since it is expected that a learning algorithm will be more sensitive to the change of certain vector coordinates than to some others', such a definition would not lead to more accurate results. On the other hand, a definition involving the probabilities of the change of each coordinate of an example, could lead to more accurate results, but it would also make the analysis of the learning algorithm prohibitively complicated, and, what is more important, it is totally unrealistic, since such kind of information about an example generator should not be expected to be available.

From what has been said above it should be apparent that the presence of r in Definition 3.17 was necessary, but, probably, any additional details/information relating the noisy and the error-free example genarators would be either unrealistic or not helpful. ■

3.4.4. Definitions of Learnability

The formal definitions of learnability can now be stated. Three kinds of learnability are distinguished; they are all in agreement with what said so far, and they will also prove useful in our work.

Definition 3.18. An algorithm has performed one-sided-error learning of a conncept f, if and only if its output g has either the properties

$$P\left[\sum_{g(v)\neq 1} D^{+}(v) \geq h_1^{-1}\right] \leq h_2^{-1} \text{ and } \sum_{g(v)\neq 0} D^{-}(v) = 0 \quad (3.1)$$

or the properties

$$\sum_{g(v)\neq 1} D^{+}(v) = 0 \text{ and } P\left[\sum_{g(v)\neq 0} D^{-}(v) \geq h_3^{-1}\right] \leq h_4^{-1} \quad (3.2)$$

■

for some $h_1, h_2, h_3, h_4 > 1$.

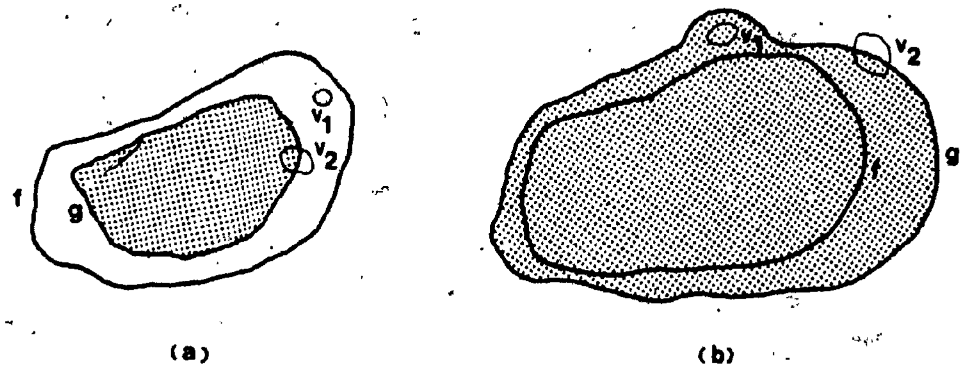


Figure 3.5. One-sided-error learning.

If f is the concept to be learned, and g is the one-sided-error learned concept, then there are two possible Venn diagrams of f and g, shown in Figure 3.5. In Figure 3.5(a), g is entirely inside f, i.e. g is more specific than f; if v_1 and v_2 are the only falsely non-included PEs, then

$$P\left[D^{+}(v_1)+D^{+}(v_2) \geq h_1^{-1}\right] \leq h_2^{-1}.$$

In Figure 3.5(b), g is entirely outside f, i.e. g is more general than f; if v_1 and v_2 are the only falsely non-excluded NEs, then

$$P\left[D^-(v_1)+D^-(v_2) \geq h_3^{-1}\right] \leq h_4^{-1}.$$ ■

The next reasonable extension of the definition of learnability is to allow both a few NEs to be falsely included and a few PEs to be falsely excluded.

Definition 3.19. An algorithm has performed *strong two-sided-error learning* of a concept f, if and only if its output g has the properties

$$P\left[\sum_{g(v)\neq 1} D^+(v) \geq h_1^{-1}\right] \leq h_2^{-1} \text{and } P\left[\sum_{g(v)\neq 0} D^-(v) \geq h_3^{-1}\right] \leq h_4^{-1} \quad (3.3)$$

for some $h_1, h_2, h_3, h_4 > 1$. ■

Clearly, one-sided-error learning is a special case of strong two-sided-error learning. ■

Definition 3.20. An algorithm has performed *weak two-sided-error learning* of a concept f, if and only if its output g has the properties

$$P\left[\sum_{g(v)\neq 1} D^+(v) \geq h_1^{-1}\right] \leq h_2^{-1} \text{and } P\left[\sum_{g(v)=1} D^-(v) \geq h_3^{-1}\right] \leq h_4^{-1} \quad (3.4)$$

or the properties

$$P\left[\sum_{g(v)=0} D^+(v) \geq h_1^{-1}\right] \leq h_2^{-1} \text{and } P\left[\sum_{g(v)\neq 0} D^-(v) \geq h_3^{-1}\right] \leq h_4^{-1} \quad (3.5)$$

for some $h_1, h_2, h_3, h_4 > 1$. ■

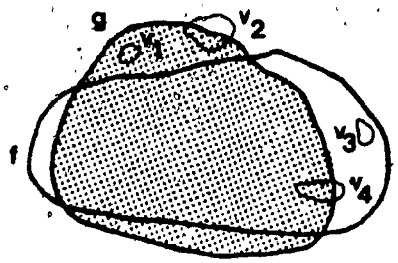


Figure 3.6. Two-sided-error learning.

A Venn diagram of a concept f to be learned and a two-sided-error learned concept g is shown in Figure 3.6. If v_1 and v_2 are the only falsely non-excluded NEs from g, and v_3 and v_4 are the only falsely non-included PEs in g, then

(a) g is strongly two-sided-error learned if and only if

$$P\left[D^+(v_3)+D^+(v_4) \geq h_1^{-1}\right] \leq h_2^{-1}$$

and

$$P\left[D^-(v_1)+D^-(v_2) \geq h_3^{-1}\right] \leq h_4^{-1}.$$

(b) g is weakly two-sided-error learned if and only if

$$P\left[D^+(v_3) \geq h_1^{-1}\right] \leq h_2^{-1}.$$

and

$$P\left[D^-(v_1)+D^-(v_2) \geq h_3^{-1}\right] \leq h_4^{-1},$$

or

$$P\left[D^+(v_3)+D^+(v_4) \geq h_1^{-1}\right] \leq h_2^{-1}$$

and

$$P\left[D^-(v_1) \geq h_3^{-1}\right] \leq h_4^{-1}.$$

Clearly, strong two-sided-error learning is a special case of weak two-sided-error learning. ■

For the rest of the thesis, the term "learning" has been reserved for either one-sided-error, or two-sided-error learning, although the term "approximate learning" will be used sometimes, for more emphasis. Also, throughout the thesis, f will stand for the concept to be learned, and g for the learned concept. ■

3.5. Summary

L. G. Valiant's view of concept learning from examples, as approximate learning, rather than exact learning considered before that, was presented in this chapter. The need for this new notion of learnability, which has been adopted in our work, stemmed from Theorem 3.4.

The definitions of all boolean terms to be used in the rest of our thesis were also stated here, as well as our own results. The last ones will prove very useful in subsequent chapters. ■

Chapter 3 completes the presentation of the material required for the design, the description, and the study of our algorithms, which started in Chapter 2. The next chapter continues with the study of learning algorithms for a particular class of boolean functions. ■

CHAPTER 4

k-CONCEPT LEARNING; KNOWN k; ONE ERROR-FREE EXAMPLE GENERATOR

4.1. Introduction

The first boolean concept learning algorithms are presented in this chapter. They learn k-CNF (k-DNF) expressions, i.e. expressions having a bounded number of literals per disjunct (conjunct), provided that k is known a priori. The motivation for looking into these classes of boolean expressions is given in Section 4.2.

A k-CNF learning algorithm, when k is known, first appeared in [VALIA84a]. A slightly modified version of it is presented here, as Algorithm 4.1, in Section 4.4, along with its detailed analysis. The design of the algorithm follows naturally from the example given in Section 4.3.

Algorithm 4.2 is the dual of Algorithm 4.1. It learns k-DNF expressions, always in the case of a known k, and it is stated in Section 4.6, along with its analysis. Some errors made in its first presentation ([VALIA85]) have been corrected here, and some critical remarks, concerning its applicability, have also been made. Section 4.5 contains an example illustrating the algorithm.

Valiant's approach to k-CNF and k-DNF learning, in the context of Mitchell's version space method, is examined in Section 4.5. The chapter closes with a summary in Section 4.8. ■

4.2. Tractability of k-CNF and k-DNF Learning

The new definitions of (approximate) learning, stated in the last chapter, were necessitated by Theorem 3.4. However, this theorem does not guarantee that under these new definitions of learnability, all boolean concepts are learnable in polynomial time. This issue is investigated in this section. The next theorem gives a partial answer to it.

Theorem 4.1. Learning* a concept $f(x_1,\ldots,x_t)$ only from PEs, or only from NEs, if no information about it is known, takes exponential time in t.

* "Learning" in Theorem 3.4 meant exact learning. "Learning" in this theorem means one/two-sided-error learning.

Proof: We will distinguish two cases.

(a) Only PEs seen.

Suppose that an algorithm trying to learn f is allowed to make over-generalizations, i.e. during its execution, the concept g learned from some PEs can be more general than the concept induced by those PEs. In such a case, the final output of the algorithm might be more general than f. That implies the following:

(i) The second condition of (3.1) might not be true.

(ii) The second condition of (3.2), (3.3), and (3.5) might not be true. That will be the case if, for example, h_3 is chosen so that

$$h_3^{-1} < \min \{D^-(v) \mid \forall v : f(v)=0\}.$$

Since g might be more general than f,

$$\sum_{g(v)\neq 0} D^-(v) > 0$$

might be true, or

$$\sum_{g(v)\neq 0} D^-(v) \geq \min \{D^-(v) \mid \forall v : f(v)=0\}$$

might be true, and therefore

$$P\left[\sum_{g(v)\neq 0} D^-(v) > h_3^{-1}\right] = 1$$

might be true.

(iii) The second condition of (3.4) might not be true; if h_3 is chosen as above, then

$$\sum_{g(v)=1} D^-(v) > \min \{D^-(v) \mid \forall v : f(v)=o\}$$

might also be true.

Therefore, no algorithm learning only from PEs should ever over-generalize; if over-generalization occurs, there is no guarantee that, independently of D^- and h_3, learning will actually be performed.

Let's assume therefore that an algorithm is trying to learn only from PEs, and that it is not allowed to over-generalize. Moreover, let's assume that all PEs are equiprobable. Then, if only 2^{t-1} of them are seen,

$$\sum_{g(v)\neq 1} D^+(v) > \frac{1}{2}$$

might be true (if, for example, all PEs are total vectors, and at least one of them has been seen twice). That implies the following:

(i) The first condition of (3.2) might not be true.

(ii) The first condition of (3.1), (3.3), and (3.4) might not be true. That will be the case if, for example, $h_1>2$. Then

$$P\left[\sum_{g(v)\neq 1} D^+(v) > h_1^{-1}\right] = 1$$

might be true.

(iii) The first condition of (3.5) might not be true, for a similar reason.

Therefore, more than 2^{t-1} PEs are required in order to guarantee that an algorithm never over-generalizing can learn f in polynomial time, only from its PEs, independently of D^+, h_1, and the number of its PEs.

(b) Only NEs seen.

The proof is the dual of the one given in (a). First, it has to be shown that any algorithm learning a concept from NEs, is not allowed to over-specialize, because the first condition of (3.1), (3.2), (3.3), (3.4), and (3.5) might not hold for some D^+ and h_1. Next, it has to be shown that no algorithm not over-specializing can be guaranteed to learn f, if less than 2^{t-1} NEs are seen, because the second condition of (3.1), (3.2), (3.3), (3.4), and (3.5) might not be true, for some D^- and h_3.■

Unfortunately, we were not able to obtain a definite answer on whether learning a concept f from both PEs and NEs can be done in polynomial time, if no a priori information about f is available. Theorem 4.1 though, has posed a new problem: identify the/some classes of boolean fuctions which are learnable in polynomial time, either only from PEs, or only from NEs.

One reason which seems to make concept learning difficult, is the fact that information about the concept (i.e. PEs and NEs) might be given piece-by-piece (i.e. as total vectors) instead of in big chunks (i.e. partial vectors). Another reason is that overlapping between partial and total vectors might occur, resulting in unnecessary processing time, since this situation cannot be easily detected and properly treated. Notice though that the fact that the example generator may repeatedly output a particular example, is not a problem any more; the new definition of learnability has taken care of it.

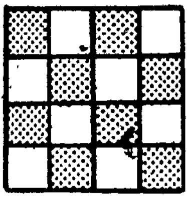


Figure 4.1. A concept difficult to be learned in polynomial time.

Clearly, there are cases in which coping with those problems is impossible. For example, if the concept of Figure 4.1 is to be learned, since no over-generalization is allowed, (almost) all PEs must be seen.

One idea could be to consider the learning of concepts for which partial vectors **can** be presented as examples, i.e. concepts having large regions of 1's or 0's in their Karnaugh maps. However, this requirement alone wouldn't be sufficient; the example generator should also be forced to (frequently) produce partial vectors. But we have already explained in Subsection 3.4.1 that assuming any particular input distribution is unrealistic and undesirable.

Why not try doing something else: try to learn a concept f having large areas of 1's or 0's, but leave the

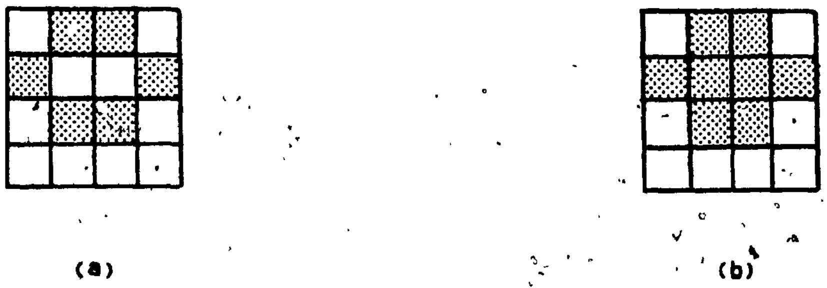


Figure 4.2. Information about a concept's form can speed up learning.

example generator completely free, and take advantage of the known fact that f consists of large regions of 1's or 0's. Consider, for example, the case of learning a 2-CNF concept f, provided that the fact that it is expressible in 2-CNF is known. After seeing some PEs inducing the concept of Figure 4.2(a), the conclusion that f has to be at least the concept of Figure 4.2(b) (and not merely that of Figure 4.2(a)) can be drawn, simply because otherwise f couldn't be written in 2-CNF.

This example suggests that the classes of k-CNF and k-DNF expressions are worth investigating for polynomial time learning. That will be the subject of the rest of this chapter. ■

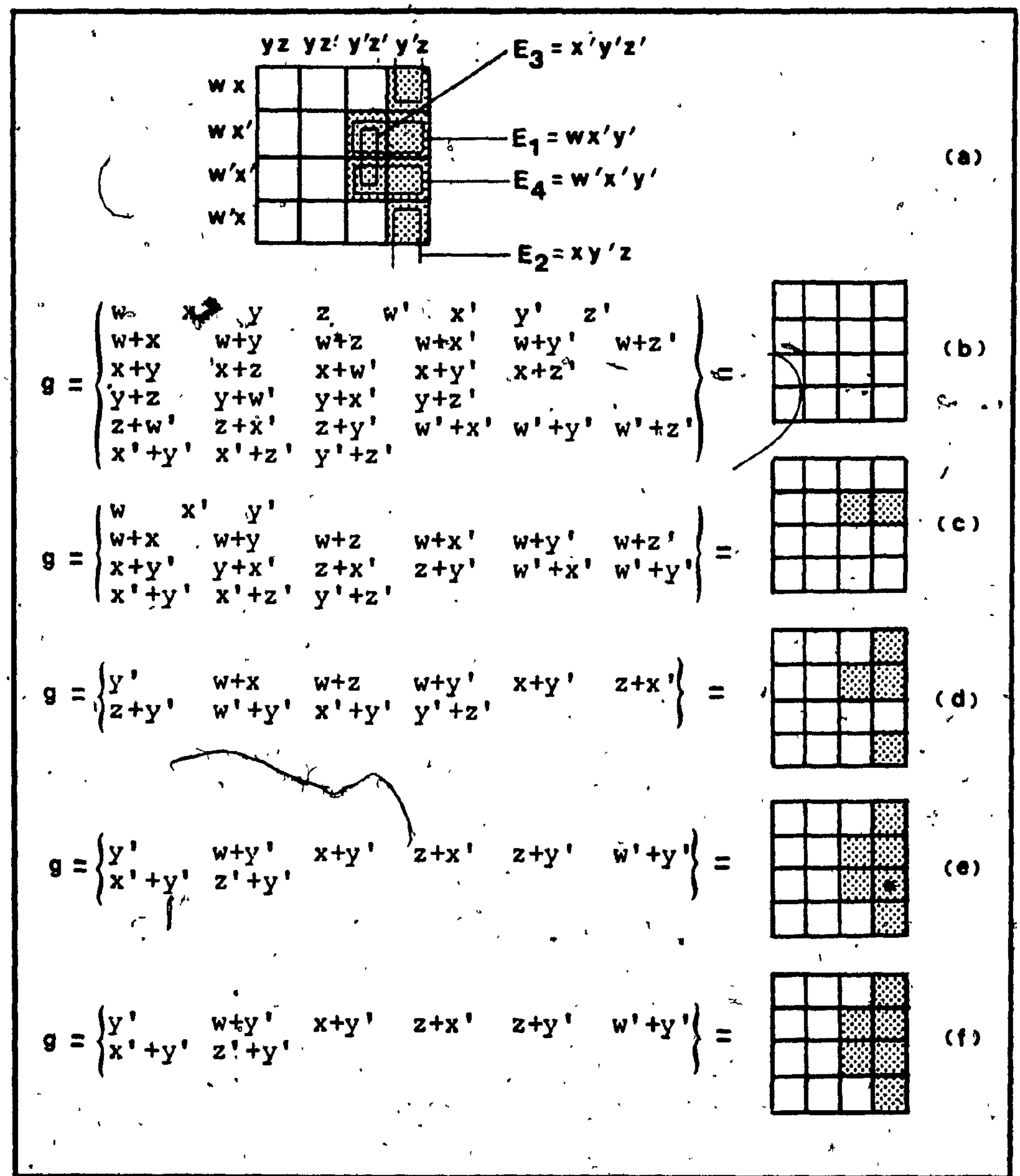
4.3. Example of k-CNF Learning from PEs

An example of learning a k-CNF expression from PEs only, when k is known, is presented in this section. It serves as an introduction to the formal description of the algorithm given in the next section. The various steps of the algorithm follow naturally from the definitions and the remarks of Section 3.2.

Consider the task of learning the 2-CNF expression $f(w,x,y,z)=y'(x'+z)$ from the PEs E_1, E_2, E_3, and E_4 shown in Figure 4.3(a). For simplicity, the case in which the seen PEs cover the entire f, has been considered.

Before the first PE is presented, the only information available about f is that it can be written in 2-CNF. Therefore, it has to be the product of some of the 32 2-sums. For this reason, the output g of the algorithm is initialized to the product of those 32 2-sums, for convenience represented as a 32-element set, shown in Figure 4.3(b). Clearly, this is the most specific concept, i.e. the empty concept.

$E_1=wx'y'$ is the first seen PE. That implies that when w=1, x=0, and y=0, g must be 1, or equivalently, each of the 2-sums of g must be 1. Therefore, each of the 2-sums of g must contain either w, or x', or y'. So what is done next



$$g = \left\{\begin{matrix} w & x & y & z & w' & x' & y' & z' \\ w+x & w+y & w+z & w+x' & w+y' & w+z' \\ x+y & x+z & x+w' & x+y' & x+z' \\ y+z & y+w' & y+x' & y+z' \\ z+w' & z+x' & z+y' & w'+x' & w'+y' & w'+z' \\ x'+y' & x'+z' & y'+z' \end{matrix}\right\} =$$

$$g = \left\{\begin{matrix} w & x' & y' \\ w+x & w+y & w+z & w+x' & w+y' & w+z' \\ x+y' & y+x' & z+x' & z+y' & w'+x' & w'+y' \\ x'+y' & x'+z' & y'+z' \end{matrix}\right\} =$$

$$g = \left\{\begin{matrix} y' & w+x & w+z & w+y' & x+y' & z+x' \\ z+y' & w'+y' & x'+y' & y'+z' \end{matrix}\right\} =$$

$$g = \left\{\begin{matrix} y' & w+y' & x+y' & z+x' & z+y' & w'+y' \\ x'+y' & z'+y' \end{matrix}\right\} =$$

$$g = \left\{\begin{matrix} y' & w+y' & x+y' & z+x' & z+y' & w'+y' \\ x'+y' & z'+y' \end{matrix}\right\} =$$

Figure 4.3. k-CNF learning from PEs.

is to delete from g all 2-sums containing none of w, x', y'. The resulting g is shown in Figure 4.3(c).

Next, the second example E$_2$=xy'z is presented. The new g is obtained in the same way, and it is shown in Figure 4.3(d). Notice that so far, g covers nothing but the concept induced by all seen PEs.

The third PE E_3=x'y'z' changes g to what is shown in Figure 4.3(e). A very important remark can be made here: One additional point, not having been seen as a PE yet, is included in g (the one denoted by a *). But **this is not an over-generalization.** Although that point has not been reported as a PE, it does belong to f, because otherwise f could not be written in 2-CNF; due to the 1-neighbours of that point, the 3-sum w+x+z' would be required.

Similar work on the fourth example E_4=w'x'y' results in the final answer shown in Figure 4.3(f). ■

Let's look now at the form of the final answer. As we can see, it is **not** in minimal CNF, i.e. it is not simplified. In this particular example of course, the minimal CNF can be obtained by dropping the terms w+y', x+y', z+y', w'+y', x'+y', and z'+y' which are implied by y'. However, no such simplification is always possible, and moreover, even in the cases in which it is possible, the resulting expression is **not** always in minimal CNF. For example, if PEs covering the entire 2-CNF concept of Figure 4.4 are seen, and the algorithm described in this section is applied, then the answer g=(x'+z)(y+z)(x+y) will be

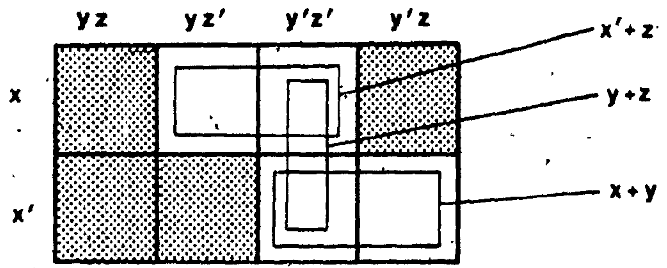


Figure 4.4. The output of the k-CNF learning algorithm cannot be simplified to minimal CNF.

obtained. There is no obvious way of simplifying it to the minimal CNF $(x'+z)(x+y)$.

In general, the algorithm described produces the maximal k-CNF expression of f. That will be proven in the next section. ■

4.4. k-CNF Learning from PEs

The k-CNF learning algorithm, described in the last section, is precisely stated here, followed by its analysis.

First of all, it can be seen that the number of k-sums, initially included in the algorithm's output g, is crucial for its time complexity, since it determines the number of steps to be executed (i.e. the number of k-sums to be examined) for each PE encountered. Obviously, there are 2t 1-sums, $\binom{t}{2}2^2$ k-sums with 2 literals each, $\binom{t}{3}2^3$ k-sums with 3 literals each, etc. Therefore, there is a total of

$$\sum_{i=1}^{k}\binom{t}{i}2^i$$

k-sums. This sum will be referred many times in our thesis, and since, unfortunately, no closed formula exists for it, it will be denoted by A_k. One remark, which can be made about it, is that since

$$\binom{t}{i}2^i = \frac{t(t-1)\ldots(t-i+1)}{i!}2^i = \theta(t^i),$$

if i is considered to be a constant, we can write

$$A_k = A_k(t) = \sum_{i=1}^{k}\binom{t}{i}2^i = \theta(t^k) \qquad \forall k<t^*. \qquad (4.1)$$

Clearly, also, the minimum value of A_k is 2, achieved when k=t=1. ■

The formal description of the algorithm follows.

Algorithm 4.1.

Task : Learn f from error-free PEs.

----

* L. G. Valiant in [VALIA84a] used the looser bound $A_k=O(t^{k+1})$; he observed that $2^i \leq t^i$.

Assumptions: (a) $x_1,\ldots,x_t$ are sufficient to express f.

(b) f can be written in k-CNF; k is known.

Parameters : $h_1, h_2 > 1$.

```
1. g <-- $\{s_i \mid s_i$ is a k-sum $\forall i=1,\ldots,A_k\}$
2. repeat $L(h_1,h_2,A_k)$ times
3.   begin
4.     v <-- EXAMPLE$^+$
5.     for each $s_i \in g$ do
6.       if $s_i(v) \neq 1$
7.         then g <-- $g-\{s_i\}$
8.   end                                              ■
```

Clearly, the space complexity of Algorithm 4.1 is nothing but the space required for the storage of the initial g, in line 1, which, according to (4.1), is $\Theta(t^k)$, under the reasonable assumption that each k-sum takes constant space.

A non-asymptotic improvement of the space complexity can be achieved by observing that the first PE seen, always causes the deletion of some k-sums from g. Therefore, g can be initialized, after the first PE v is presented, to the set of all k-sums compatible with (made 1 by) v.

All algorithms in our thesis have the same space complexity. For this reason, it will not be mentioned again. ■

The next theorem provides some information about the output of Algorithm 4.1.

Theorem 4.2. The output g of Algorithm 4.1 has the properties:

(a) $\sum_{g(v)\neq 0} D^{-}(v) = \sum_{g(v)=1} D^{-}(v) = 0$.

(b) $P\left[\sum_{g(v)\neq 1} D^{+}(v) \geq h_1^{-1}\right] \leq h_2^{-1}$.

(c) g is in maximal k-CNF.

Proof: Let $f=s_1s_2\ldots s_m$ be the concept to be learned in maximal k-CNF. Let's call $L = L(h_1,h_2,A_k)$, and let also $g_0 = \{s_i \mid i=1,\ldots,m,m+1,\ldots,A_k\}$ be the initial g in line 1, and g_i be the value of g after the i-th execution of the **repeat**-loop, for $i=1,2,\ldots,L$.

(a) Let v be a PE. By definition, $f(v)=1$, and according to Theorem 3.3(c), $s_i(v)=1$ holds for all $i=1,2,\ldots,m$. Therefore, the test of line 6 implies that none of the k-sums of f will ever be deleted from any g_i, for $i=0,1,\ldots,L-1$. Since now $g=g_L$, it follows that **all k-sums of f will be included in g** as well.

Let now u be a vector such that $f(u)=0$. Theorem 3.3(a) implies that $s_i(u)=0$ must be true for some $i\in\{1,2,\ldots,m\}$. According to what said before, s_i will be included in g as well, therefore $g(u)=0$ will be true, as well. That proves that if $f(u)=0$ for some vector u, then $g(u)=0$

which is equivalent to

$$\sum_{g(v)\neq 0} D^{-}(v) = 0,$$

and that

if $f(u)=0$ for some vector u, then $g(u)\neq 1$

which is equivalent to

$$\sum_{g(v)=1} D^{-}(v) = 0.$$

That completes the proof.

(b) Let

$$X_i = \sum_{g_i(v)\neq 1} D^{+}(v) \quad \forall i\in\{0,1,\ldots,L\}.$$

It will be shown that

$$P\left[X_L \geq h_1^{-1}\right] \leq h_2^{-1}.$$

Each seen PE, v, can be interpreted as a Bernoulli trial, with "success" being "v makes at least one k-sum of the current g_i 0/*", or "v causes the deletion of at least one k-sum from the current g_i". The number of successes occurred is at most A_k-m, since, according to what said in (a), all k-sums of f will be retained in g. Therefore, the number of successes is at most A_k (since the only known fact about m is that $m\geq 0$). Each X_i now, for $i=0,1,\ldots,L-1$, denotes the probability of success of the (i+1)-th trial, i.e. the probability of deleting at least one k-sum during the (i+1)-th execution of the **repeat**-loop, when obtaining g_{i+1} from g_i. Clearly, the sequence $X_0,X_1,\ldots,X_L$ is non-increasing, with $X_0=1$ (since, for example, both x_1 and x_1' belong to g_0, and hence at least one of them will be deleted). Therefore, the event

"$X_L \geq h_1^{-1}$"

implies (but it is not equivalent to) that

> "less than A_k successes have occurred in $L(h_1,h_2,A_k)$ Bernoulli trials, each with success probability at least h_1^{-1}".

Hence,

$$P\left[X_L \geq h_1^{-1}\right] \leq b^*\left(\leq A_k;\ L(h_1,h_2,A_k),\ \geq h_1^{-1}\right),$$

which, according to Definitions 2.4, 2.5, and Theorem 2.1, implies that

$$P\left[X_L \geq h_1^{-1}\right] \leq h_2^{-1}.$$

Hence, the proof is complete.

(c) It will be shown that after the i-th iteration of the **repeat**-loop, for all $i=0,1,\ldots,L$, g_i is in maximal k-CNF. The proof will be done by induction on i.

(i) **Basis:** g_0 is in maximal k-CNF, since it contains all k-sums.

(ii) **Inductive step:** Let $g_i=s_1\ldots s_q s_{q+1}\ldots s_{q+n}$, for some $q,n\in N_0$, be the value of g after the i-th iteration, for some $i\in\{0,1,\ldots,L-1\}$. Assume that g_i is in maximal k-CNF. Let v be the PE generated by EXAMPLE$^+$ in the (i+1)-th iteration, and let $g_{i+1}=s_1\ldots s_q$ be the value of g after the (i+1)-th iteration. It will be proven that g_{i+1} is in maximal k-CNF. The proof will be done by contradiction.

Suppose that g_{i+1} is not in maximal k-CNF. Then, there must exist a k-sum s different from $s_1,\ldots,s_q$, such that $g_{i+1}=g_{i+1}*s$. Since g_i is in maximal k-CNF, s has to be one of $s_{q+1},\ldots,s_{q+n}$ (because otherwise $g_i=g_i*s$ would be true, for some s different from $s_1,\ldots,s_{q+n}$). So,

$$g_{i+1} = g_{i+1}*s_j, \text{ for some } j\in\{q+1,\ldots,q+n\}.$$

Hence, for the particular vector v,

$$\exists j\in\{q+1..q+n\}: g_{i+1}(v)=g_{i+1}(v)*s_j(v). \quad (4.2)$$

But according to lines 6 and 7 of the algorithm, $s_1(v)=\ldots=s_q(v)=1$, therefore,

$$g_{i+1}(v) = 1 \qquad (4.3)$$

and

$$s_j(v) \neq 1 . \quad \forall j\in\{q+1,\ldots,q+n\},$$

which implies that

$$\forall j\in\{q+1,\ldots,q+n\} \left(\exists v_0\in T_v : s_j(v_0)=0\right). (4.4)$$

Since now (4.2) and (4.3) must also be true for the vector v_0 of (4.4), it follows that they contradict with (4.4). Therefore, g_{i+1} must be in maximal k-CNF, which completes the proof of the inductive step.

The inductive proof now of part (c) is complete. ■

Two remarks can be made about Algorithm 4.1, both of them stemming from the proof of Theorem 4.2(b). First, if m, the number of k-sums of f, were known, then fewer PEs would be required, resulting in a lower running time.

Second, even for the case of an unknown m, $L(h_1,h_2,A_k-1)$ PEs would be sufficient, since h>0, and therefore the number of successes is at most A_k-1, rather than just A_k. However, since this is not a very significant improvement, A_k has been used for simplicity. ■

Theorem 4.2 implies that Algorithm 4.1 performs one-sided-error learning. The difference between the properties of g in part (a) of the theorem, and Definition 3.18 is not important. In fact, although in general

$$P\left[\sum_{g(v)\neq 0} D^-(v) \geq h_3^{-1}\right] \leq h_4^{-1}$$

is not equivalent to

$$P\left[\sum_{g(v)=1} D^-(v) \geq h_3^{-1}\right] \leq h_4^{-1}$$

(this is the reason for which the distinction between weak and strong two-sided-error learning was made), it can be proven that, independently of the learning algorithm,

$$\sum_{g(v)\neq 0} D^-(v) = 0 \qquad (4.5)$$

is equivalent to

$$\sum_{g(v)=1} D^-(v) = 0. \qquad (4.6)$$

The proof goes as follows. Since (4.5) is equivalent to

$$f(v)=0 \Rightarrow g(v)=0, \text{ for all vectors } v, \qquad (4.7)$$

and (4.6) is equivalent to

$$f(v)=0 \Rightarrow g(v)\neq 1, \text{ for all vectors } v, \qquad (4.8)$$

it suffices to show that (4.7) and (4.8) are equivalent. Clearly, (4.7) implies (4.8). To show now that (4.8) implies (4.7), assume that v is a vector such that $f(v)=0$. If v is a total vector, then (4.8) implies that $g(v)\neq 1$, therefore $g(v)=0$. If v is a partial vector, then the assumption $f(v)=0$ implies that $f(v_0)=0 \;\forall v_0 \in T_v$. Therefore, (4.8) implies that $g(v_0)=0 \;\forall v_0 \in T_v$. Hence $g(v)=0$. ■

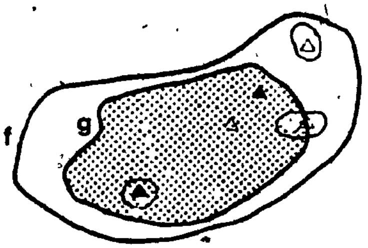


Figure 4.5. Venn diagram of f and g of Algorithm 4.1. A △ stands for a non-seen PE, whereas a ▲ stands for a seen PE.

A Venn diagram of the concept f to be learned, and the output g of Algorithm 4.1 is shown in Figure 4.5. According to part (a) of the last theorem, g is more specific than f. From the course of the algorithm also, it should be clear that all seen PEs are correctly included in g. Some non-seen PEs are possibly included in g as well, because it is known that f can be written in k-CNF (see, for example, Figure 4.3(e)). Unavoidably though, some non-seen PEs will not be included in g, but part (b) of the theorem implies that it can become highly improbable (i.e. the probability can be at most h_2^{-1}) that their weight (i.e. the sum of their probabilities to be presented as PEs) will be higher than even a small quantity (i.e. greater than h_1^{-1}). ■

L. G. Valiant, in his algorithm in [VALIA84a], has set $h=h_1=h_2$. Clearly, our **algorithm is more flexible than Valiant's**, in the sense that it gives more freedom to the user to specify the reliability (s)he requires from the algorithm. ■

The price the user has to pay for a better approximation is given by the next theorem.

<u>Theorem 4.3.</u> The worst-case time complexity of Algorithm 4.1 is

$$T(t,h_1,h_2) = \Theta\left(h_1 t^{k+1}(t^k+\ln h_2)\right).$$

Proof: According to Theorem 3.1(c), execution of line 6 requires $\Theta(t)$ time, since each of v and s contain at most t literals, and the intersection of two t-element sets can be computed in $\Theta(t)$ time. In the worst case, the number of s_i in g will be A_k, or $\Theta(t^k)$ according to (4.1). Since lines 4 and 7 require constant time, lines 4 to 7 require $\Theta(t^{k+1})$ time in the worst case. According to Theorem 2.2 now, lines 2 to 8 require $\Theta\left(h_1 t^{k+1}(t^k+\ln h_2)\right)$ time in the worst-case, and since line 1 requires $\Theta(t^k)$ time, the result follows. ■

This theorem says that not only Algorithm 4.1 is polynomial in t, but also that it is (sub)linear in the user-specified parameters h_1 and h_2. That means that the price the user has to pay for a better approximation, is not too high. ■

From the proof of the theorem also, it should be clear that the number of PEs required by Algorithm 4.1 (i.e. the number of iterations of the **repeat**-loop) is crucial for its time complexity. Valiant's algorithm, in [VALIA84a], requires $\lceil 2h(k+lnh)\rceil$ PEs. According to our discussion after Theorem 2.2, that implies that our **algorithm is non-asymptotically faster than Valiant's**, when applied in the restricted case of $h=h_1=h_2$ (that Valiant's algorithm is applicable), or when the modified Valiant's results in (2.28) make his algorithm applicable when $h_1 \neq h_2$. Clearly though, our **algorithm is asymptotically as fast as Valiant's.** ■

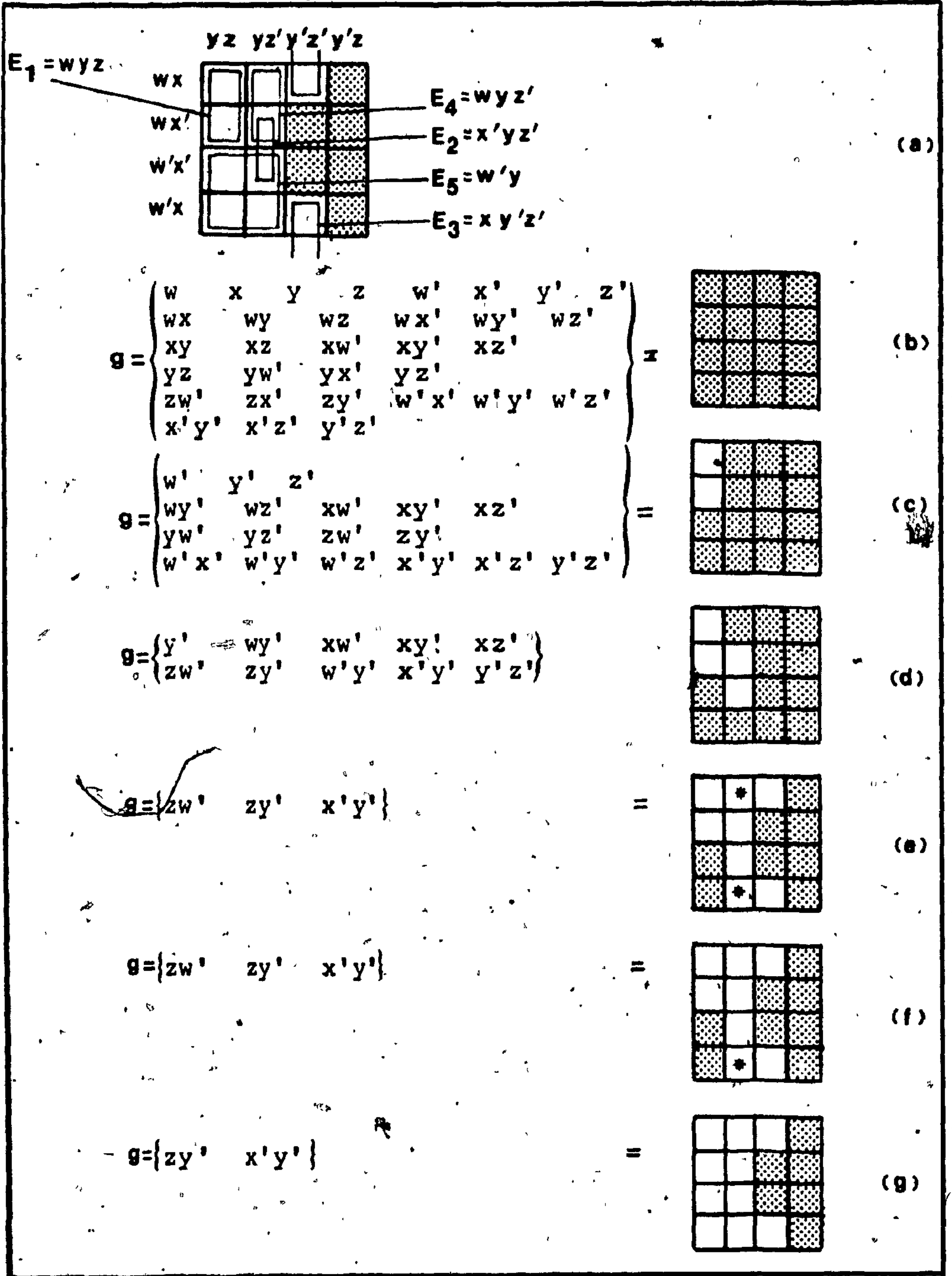
Valiant's definition of learnability, as well as his algorithm, can now be appreciated. Not only he was able to **learn in polynomial time a close approximation of a class of boolean functions** without knowing anything about the input distribution, but he was also able to **make conclusions about the closeness of the output to the exact solution.** As stated in Section 1.8, that was not the case with previous works in learning; not too many persons paid attention to the time complexity of their algorithms, and not too many persons drew conclusions about the closeness of their answers to the exact solution. ■

4.5. Example of k-DNF Learning from NEs

Now that the algorithm for k-CNF learning from PEs has been understood, its dual algorithm, performing k-DNF learning from NEs, can be presented. An example of learning the same boolean function of Figure 4.3(a), which can also be written in 2-DNF as $f(w,x,y,z) = x'y'+zy'$, but from NEs this time, is shown in Figure 4.6. Again, for simplicity, it is assumed that the NEs seen cover the entire space outside f; they are E_1, E_2, E_3, E_4, and E_5 in order of presentation, and they are shown in Figure 4.6(a).

Here, g is initialized to the sum of all 32 2-products, as ~~shown~~ in Figure 4.6(b), i.e. to the most general concept. The first NE E_1=wyz implies that when w=y=z=1, g must be 0, therefore each of the 2-products of g has to be 0. That results in the deletion of all 2-products containing none of w', y', z' and the g shown in Figure 4.6(c) is obtained.

For every NE seen, the current g is specialized, if necessary, so that it will not match the new example. The



$$g=\begin{Bmatrix} w & x & y & z & w' & x' & y' & z' \\ wx & wy & wz & wx' & wy' & wz' & & \\ xy & xz & xw' & xy' & xz' & & & \\ yz & yw' & yx' & yz' & & & & \\ zw' & zx' & zy' & w'x' & w'y' & w'z' & & \\ x'y' & x'z' & y'z' & & & & & \end{Bmatrix} =$$

$$g=\begin{Bmatrix} w' & y' & z' & & & \\ wy' & wz' & xw' & xy' & xz' & \\ yw' & yz' & zw' & zy' & & \\ w'x' & w'y' & w'z' & x'y' & x'z' & y'z' \end{Bmatrix} =$$

$$g=\begin{Bmatrix} y' & wy' & xw' & xy' & xz' \\ zw' & zy' & w'y' & x'y' & y'z' \end{Bmatrix}$$

$$g=\{zw' \quad zy' \quad x'y'\} =$$

$$g=\{zw' \quad zy' \quad x'y'\} =$$

$$g=\{zy' \quad x'y'\} =$$

Figure 4.6. k-DNF learning from NEs.

values of g upon presentation of E_2, E_3, E_4, and E_5 are shown in Figures 4.6(d), 4.6(e), 4.6(f), and 4.6(g) respectively. Notice that, as it is also the case with the k-CNF learning, upon presentation of an example (E_3=xy'z'), some points (denoted by ✱) not seen as NEs are excluded from g. Again, **this is not over-specialization;** those points have to be excluded in order for the boolean function to be expressible in 2-DNF. The final answer g=x'y'+zy' obtained, in addition of being in maximal 2-DNF as expected, it also happened to be in minimal DNF. Obviously, this will not be always the case. ■

L. G. Valiant mentioned the feasibility of k-DNF learning from NEs in [VALIA84a], and he presented a detailed algorithm for it in [VALIA85] (although he admitted that this new algorithm is "essentially the dual" of the one performing k-CNF learning in [VALIA84a]). However, in both places, he made a mistake. He defined

> a probability distribution D^- on the set of vectors such that f(v)≠1,
>
> ([VALIA84a], p. 1137)

and he claimed that by dualizing the k-CNF learning algorithm

> for any k, the class of DNF expressions having a bound k on the length of each disjunct can be learned in polynomial time from negative examples

alone (i.e. from a source of vectors such that $D^-(v) \neq 0$).

([VALIA84a], p. 1140)

According to our terminology, that means that $EXAMPLE^-$ outputs vectors v such that $f(v) \neq 1$. But clearly, this is wrong, since in the example of Figure 4.6, w and w' are two such vectors, and if they are presented as NEs, the empty concept will be finally learned. This is the reason for which the Definitions 3.15 and 3.16 of D^- and $EXAMPLE^-$ are different from those given by Valiant. ■

4.6. k-DNF Learning from NEs

The k-DNF learning algorithm, described in the last section, is precisely stated here. Long discussions have been omitted, since they would be the duals of those made for Algorithm 4.1.

Algorithm 4.2.

Task : Learn f from error-free NEs.

Assumptions: (a) $x_1,\ldots,x_t$ are sufficient to express f.

(b) f can be written in k-DNF; k is known.

Parameters : h_3, $h_4 > 1$.

```
1. g <-- $\{p_i \mid p_i$ is a k-product $\forall i=1,\ldots,A_k\}$
```

```
repeat $L(h_3,h_4,A_k)$ times
  begin
    $v$ <-- EXAMPLE$^-$
    for each $p_i \in g$ do
      if $p_i(v) \neq 0$
        then $g$ <-- $g-\{p_i\}$
  end
```
■

The duals of Theorems 4.2 and 4.3 hold for Algorithm 4.2. They are stated below.

Theorem 4.4. The output g of Algorithm 4.2 has the properties:

(a) $\sum_{g(v)\neq 1} D^+(v) = \sum_{g(v)=0} D^+(v) = 0.$

(b) $P\left[\sum_{g(v)\neq 1} D^-(v) \geq h_3^{-1}\right] \leq h_4^{-1}.$

(c) g is in maximal k-DNF.

Proof: The dual of that of Theorem 4.2. ■

Clearly, Algorithm 4.2 performs one-sided-error learning. A Venn diagram of f and g of Algorithm 4.2 is shown in Figure 4.7. According to part (a) of the last theorem, g is more general than f. From the course of the algorithm also, it should be clear that all seen NEs are correctly excluded from g. Some non-seen NEs are possibly excluded from g as well, since it is known that f can be

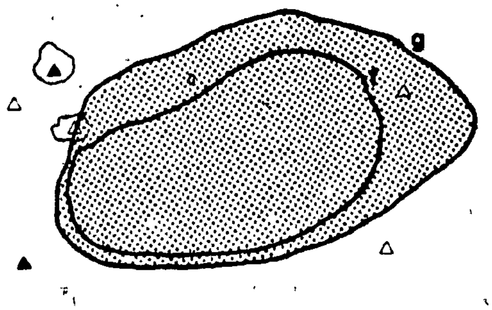


Figure 4.7. Venn diagram of f and g of Algorithm 4.2. A △ stands for a non-seen NE, whereas a ▲ stands for a seen NE.

written in k-DNF (see, for example, Figure 4.6(e)). Part (b) of the last theorem implies that the probability that the weight of the falsely non-excluded NEs will be higher than even a small value (h_3^{-1}), can become quite low (h_4^{-1}). This increased confidence in the algorithm's output requires additional execution time, as indicated by the following theorem. ■

Theorem 4.5. The worst-case time complexity of Algorithm 4.2. is

$$T(t,h_3,h_4) = \Theta\left(h_3 t^{k+1}(t^k+\ln h_4)\right).$$

Proof: As in Theorem 4.3. ■

Although L. G. Valiant observed that a k-DNF learning algorithm can be constructed, he also commented that this might not be helpful, because

> the main source of difficulty is the fact that the problem of determining whether a non-total vector implies [i.e. is making 1] a function specified by a DNF formula is NP-hard*.
>
> ([VALIA84a], p. 1139)

This is a true statement, but what Valiant failed to notice is that the output of Algorithm 4.2 is in maximal k-DNF. Therefore, according to Theorems 3.1(a) and 3.2(a), **determining whether a vector makes the output of Algorithm 4.2 equal to 1, is not NP-hard;** it takes at most $\Theta(t^{k+1})$ time. According also to Theorems 3.1(b) and 3.2(c), determining whether a vector makes the output of the algorithm equal to 0, requires the same amount of time. ■

4.7. Discussion

In this section, Algorithms 4.1 and 4.2 are examined in the context of Mitchell's version space, presented in Section 1.7.

----

* In [VALIA85] though, where Valiant gave the detailed description of the algorithm, he mentioned nothing about the usefulness of DNFs as concept classifiers.

As it was stated in Section 1.5, the task of concept learning from examples can be viewed as a search of the rule space. In the case of boolean concept learning, the rule space is the set of all boolean functions, and it is partially ordered under the more-general-than relation. An example for the case of 2 variables is shown in Figure 4.8.

Algorithm 4.1 starts with the most specific concept (in the top of Figure 4.8), which it generalizes (i.e. it moves downwards) as long as new PEs are seen. The generalization done is the most conservative one, as explained in Section 4.3, and therefore, the final output is the least general concept compatible with all PEs seen. Similarly, Algorithm 4.2 starts with the most general concept (in the bottom of Figure 4.8), which it specializes in the most conservative way, on the appearence of new NEs, and its final answer is the least general concept compatible with all NEs seen.

What is worth noticing here, is that in the case of boolean function learning, **each observed PE (NE) results in the most conservative generalization (specialization).** In other words, the sets G and S, delimiting the version space, always contain one element. ■

There is one idea which is worth investigating. Mitchell's version space guarantees that the correct answer is always contained in a set of plausible answers, delimited

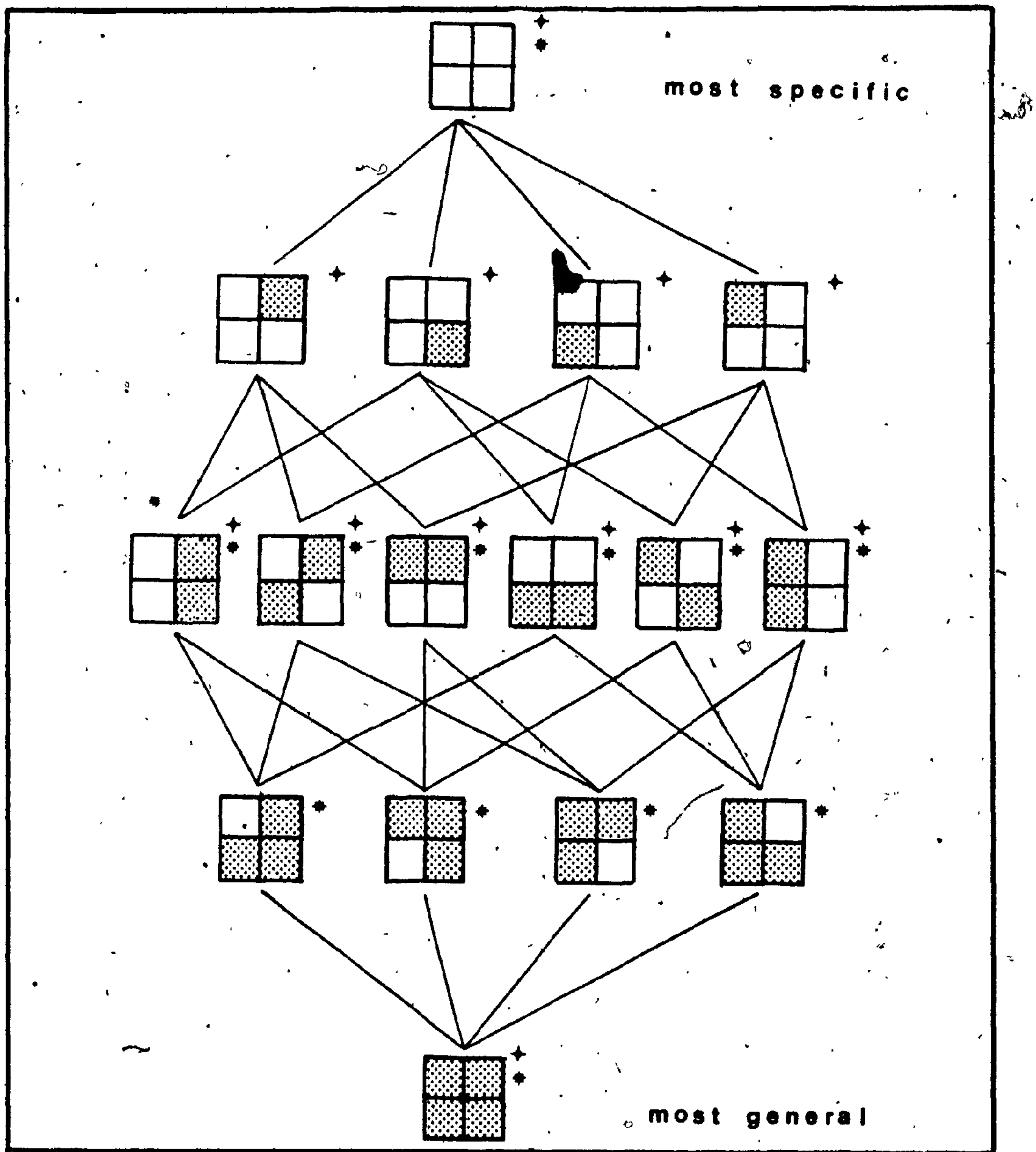


Figure 4.8. Partial ordering of 2-variable boolean functions.

Functions in 1-CNF are indicated by a ✦. Functions in 1-DNF are indicated by a ✱.

by S and G. On the other hand, this chapter's algorithms have the property that probably the answer found is quite close to the correct answer. Obviously, this difference between the two approaches is due to the fact that version space moves in both directions in the space of all possible answers, whereas Algorithms 4.1 and 4.2 move in only one direction. Is it possible to design an algorithm which, instead of moving only downwards (e.g. k-CNF learning), or only upwards (e.g. k-DNF learning), it will move in both directions, producing in this way a set of plausible aaswers such that the correct answer would be one of them with probability 100%?

Our first attempt is to see whether the k-CNF learning algorithm from PEs, and the k-DNF learning algorithm from NEs can be simultaneously applied. Unfortunately, in order to do so, the concept to be learned should be in **both** k-CNF and k-DNF. If it is in k-CNF, but not in k-DNF, then the two algorithms will not be searching the same space, as indicated in Figure 4.8, and the entire version space approach will collapse. But considering the learning of concepts expressible in both k-CNF and k-DNF, imposes severe restrictions on the (already restricted) classes of polynomially learnable boolean functions. For this reason, this idea will not be pursued further.

Our second attempt is to see whether, in addition of applying Algorithm 4.1, i.e. in addition of learning k-CNF from PEs (by generalizing), an algorithm learning k-CNF from NEs (by specializing) could be designed.

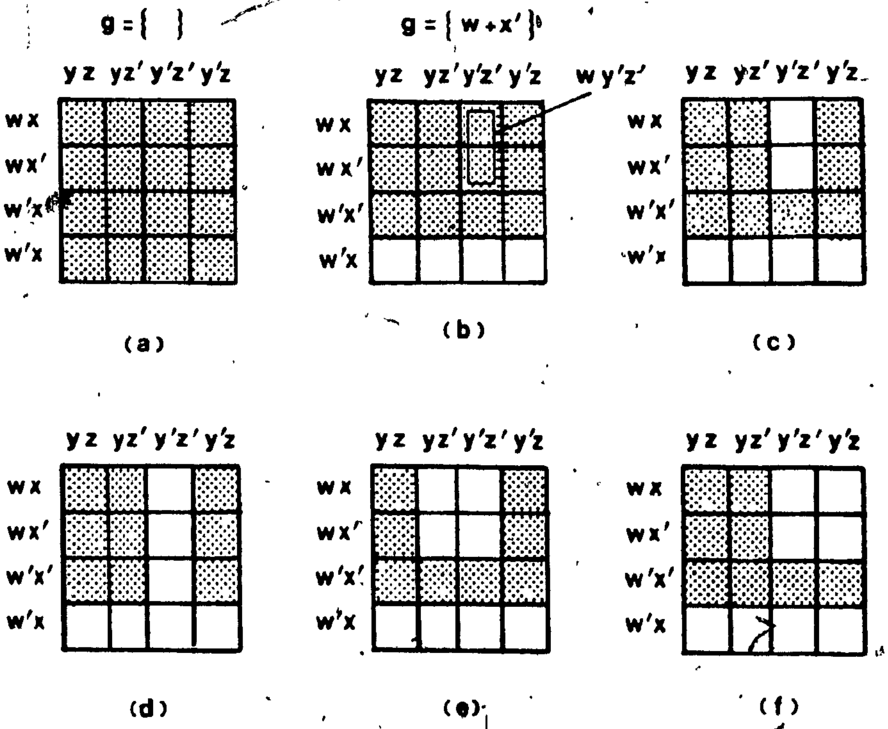


Figure 4.9. Attempt for k-CNF learning from NEs.

Let's look at the example of Figure 4.9. Suppose that a 2-CNF concept is to be learned from NEs. Clearly, the learning algorithm performing this task, should initialize its output g to the empty set, as shown in Figure 4.9(a), and then for each seen NE $v=u_1...u_q$, it should add the term $u_1'+...+u_q'$ to g. For example, if the first NE seen is v=w'x, then the resulting g would be the one shown in Figure 4.9(b).

Suppose now that the next NE seen is v=wy'z'. Obviously, g cannot become the concept of Figure 4.9(c) because this is in 3-CNF and not in 2-CNF; g has to become one of the concepts shown in Figures 4.9(d), 4.9(e), or 4.9(f). Therefore, one of the following has to be done: either the algorithm has to keep track of all three possible concepts in Figures 4.9(d), 4.9(e), and 4.9(f), or the NE seen has to be ignored since it did not result in a 2-CNF, or an ORACLE has to be used, which, when presented with data (the vectors w'y'z', wyz', and wy'z in the example under consideration), it will tell the learner whether the data is a NE or not.

None of those solutions is considered satisfactory. The first solution has been investigated, and it led to a complicated search, in addition of increasing the space complexity. The second solution is clearly unacceptable; consider, for example, the case in which a very common NE is repeatedly ignored because it is presented to the learning algorithm in a "bad moment", i.e. when its consideration forces the learned concept not to be in k-CNF any more. Finally, the idea of an ORACLE was rejected; although L. G. Valiant himself did use ORACLEs in k-DNF learning, he admitted that

> there is the additional philosophical difficulty that availability of the [k-DNF] expression does not itself imply a polynomial time algorithm for ORACLE*. On the other hand, the theorem [analyzing a learning algorithm using ORACLEs] does say that if an agent has some, may be ad hoc, black box for ORACLE whose workings are unknown to him or her, it can be used to teach someone else a DNF expression that approximates the function.
>
> ([VALIA84a], p. 1140)

Our feeling is that it is not useful or reasonable to assume that the teacher already knows the concept (s)he want to teach the learner.

For those reasons, the idea of learning k-CNF expressions from NEs only, is abandoned.

Since our second attempt, to perform Mitchell's bidirectional conservative search, was not successful, Valiant's one-directional search method will be the basis of our work. ■

----

* As already mentioned in Section 4.6, Valiant thought that for the k-DNF output g of his algorithm, the question of determining whether g(v)=1 or not was NP-hard.

4.8. Summary

The need to look for particular classes of boolean concepts which could be (approximately) learnable in polynomial time, was suggested by Theorem 4.1. The classes of k-CNF and k-DNF expressions, when the value of k is known, were identified to be two such classes.

Algorithm 4.1, performing one-sided-error polynomial learning of k-CNF expressions from PEs, and Algorithm 4.2, performing one-sided-error polynomial learning of k-DNF expressions from NEs were presented and analyzed; in both cases k should be known. These algorithms are based on Valiant's work. Their twofold importance was stressed; they perform approximate learning in **polynomial time**, and **it is known how good** this approximation is. Valiant's mistake in the definition of the NEs required for the k-DNF learning was corrected in Section 4.6. It was also pointed out that, in contrast to what he believed, the **k-DNFs** learned by Algorithm 4.2 **are good concept classifiers**; this is also the case for the k-CNFs learned by Algorithm 4.1. Also, due to our results of Chapter 2, **our algorithms** presented here **are faster and more flexible** than his.

Next, the relation between Valiant's method and Mitchell's version space was discussed, and it was found that a direct application of Mitchell's bidirectional search wouldn't be profitable for boolean concept learning. ■

After having thoroughly studied the problem of k-CNF and k-DNF learning from error-free data, when k is known, we are now ready to look at the problem of k-CNF ~~and~~ k-DNF learning from noisy data, always in the case of a known k. This is investigated in the next chapter. ■

CHAPTER 5

k-CONCEPT LEARNING; KNOWN k; ONE NOISY EXAMPLE GENERATOR

5.1. Introduction

As already explained in Section 1.6, in real-life applications, erroneous rather than error-free data will be fed into a learning system. Although one could argue that the task of k-CNF and k-DNF learning from error-free data, **for a known k** (studied in Chapter 4), is not a real-life application, it is true that the study of the learnability of those classes of boolean functions, always for a known k, but from **noisy data** this time, is not easy at all, and therefore it is worth pursuing. If not anything else, such a study will help us to better understand this particular problem, and thereafter to solve more down-to-the-earth problems.

An algorithm for k-DNF learning from noisy NEs, due to L. G. Valiant, first appeared in [VALIA85]. The idea underlying this algorithm is explained in Section 5.2, and it forms the starting point for the design of our own algorithms in this chapter.

Algorithm 5.1, learning k-CNF expressions from noisy PEs, is presented in Section 5.3, followed by a characterization of its output in Section 5.4. Several useful remarks have been included in Section 5.5, and the algorithm's time complexity analysis is given in Section 5.6. A comparison between Algorithm 5.1 and Valiant's algorithm is made in Section 5.7.

Algorithm 5.2 is the dual of Algorithm 5.1. It performs k-DNF learning from noisy NEs, and it is briefly stated in Section 5.8, along with its analysis.

Finally, the chapter closes with a summary is Section 5.9. ■

5.2. Example of k-CNF Learning from PEs

Since the k-CNF learning algorithm from noisy PEs will be based on the k-CNF learning algorithm from error-free PEs, let's look once more at the last algorithm, from a slightly different angle.

As it has already been explained, g is initialized to the set of all k-sums, i.e. to the set of all k-sums of f, augmented by any additional k-sums not in f. Each k-sum of f will be made 1 by all PEs, whereas each k-sum not in f will be made 0/* by at least one PE. Some of the last k-sums will be made 0/* by many PEs, and some others by only a few, depending on the PEs' distribution D+.

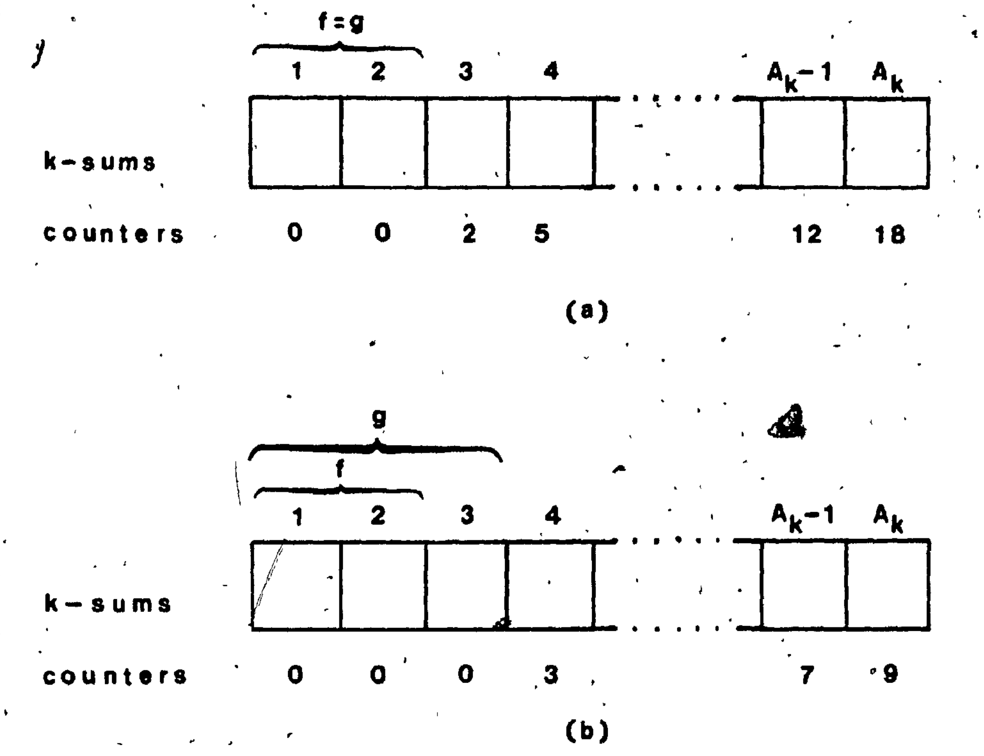


Figure 5.1. k-CNF learning from PEs, by using counters.

If a counter is associated with each k-sum, counting the number of times the k-sum is made 0/* by a PE during the execution of Algorithm 4.1, then a possible situation is shown in Figure 5.1. All counters of the k-sums of f will definitely be zero. If now all PEs are seen, then the counters of all k-sums not in f will be non-zero, as shown in Figure 5.1(a), whereas, if not all PEs are seen, then the counters of only some of the k-sums not in f will be non-zero, as shown in Figure 5.1(b). Theorem 4.2 guarantees that if a sufficient number of PEs is seen, then probably only the k-sums, which have low counters, will decrease their counters to zero, and therefore will be falsely included in g; the remaining k-sums will have non-zero counters, hence they will be deleted sometime in the course of Algorithm 4.1.

Suppose now that some of the PEs seen are erroneous. That may change some of the zero counters of the k-sums of f to non-zero, which implies that over-generalization may occur. According to our definitions of learnability, the probability of that happening has to be minimized, if learning is to be performed. To do so, instead of deleting a k-sum when a PE making it 0/* appears, **a counter will be kept for each k-sum initially in g, and at the end** (i.e. when sufficient number of PEs has been seen), **all k-sums whose counter is greater than a certain threshold d, will be deleted.** Clearly, the higher d is, the less probable it is that a k-sum will be falsely deleted from g, i.e. the less probable it is that over-generalization will occur.

In addition to what was just said, the presence of the erroneous PEs will have one more effect. The counters of the k-sums not in f may either increase or decrease, and some of them may become zero. The k-sums whose counters become zero, will be unavoidably falsely included in g. The k-sums, now, whose counters will become quite low, will be possibly falsely included in g, due to the threshold d introduced above, resulting in an under-generalization. Clearly again, the higher d is, the more probable it is that many k-sums will be falsely included in g, i.e. the more probable it is that an extensive under-generalization will take place. ■

The above described idea for k-CNF learning from noisy PEs is quite simple, and it first appeared in [VALIA85]. However, the crucial point in the design of the algorithm is the choice of the threshold d, so that over-generalization will probably not occur, and under-generalization will be probably not too high; according to what was said above, these two demands are conflicting. Also, since d will affect the number of PEs which have to be seen, and therefore the execution time of the algorithm as well, d has to be chosen so that it minimizes the number of PEs required.

The detailed description of the algorithm is given in the next section. Our choice of d is different from that of L. G. Valiant in [VALIA85], and as it will prove in Section

5.7, it is much better than his.

5.3. k-CNF Learning Algorithm from PEs

The algorithm described in the last section is precisely stated here.

Algorithm 5.1.

Task : Learn f from noisy PEs.

Assumptions: (a) $x_1, \ldots, x_t$ are sufficient to express f.

(b) f can be written in k-CNF; k is known.

(c) Error rate r of PEs generator is known.

Parameters : h_1, h_2, $h_4 > 1$.

```
1. g <-- $\{s_i \mid s_i \text{ is a k-sum}, \forall i=1,2,\ldots,A_k\}$
2. zero a counter COUNT[i] for each $s_i$
3. a <-- $\ln(h_4)$
4. b <-- $\ln(h_2)$
5. x <-- $\ln\left(\frac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}\right)$
6. y <-- $\ln\left(\frac{eh_1A_k(1-r)^{-1}}{eh_1A_k(1-r)^{-1}+1-e}\right)$
7. d <-- $\frac{y(1+a) + b(1-x)}{a+b+x+y}$
8. M <-- $\left\lceil\frac{a+b+1}{x+y-1}\right\rceil$
9. repeat M times
```

```
10.  begin
11.    v <-- r-EXAMPLE$^+$
12.    for each $s_i \in g$ do
13.      if $s_i(v) \neq 1$
14.        then   COUNT[i] <-- COUNT[i]+1
15.  end
16. delete from g all $s_i$ having $\frac{\text{COUNT[i]}}{M} \geq d$   ■
```

The next theorem establishes the necessity of certain conditions in order for the algorithm to be applicable.

<u>Theorem 5.1.</u> Algorithm 5.1 is applicable if and only if

$$0 < r < r_{max} = \frac{\sqrt{(eA_k+2-e)^2+4(e-1)}-(eA_k+2-e)}{2(e-1)} \qquad (5.1)$$

holds for the error rate r of the PEs generator, and also

$$1 < h_1 < h_{max} = \frac{(1-r)(er+1-r)}{eA_k r} \qquad (5.2)$$

holds for the user-specified parameter h_1. Moreover,

$$r \dashrightarrow r_{max} \text{ if and only if } h_{max} \dashrightarrow 1. \qquad (5.3)$$

Proof: Clearly, M, defined in line 8 of Algorithm 5.1, has to be a positive number. Therefore, its denominator has to be positive, or equivalently

$$1 - \ln\left(\frac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}\right) < \ln\left(\frac{eh_1A_k(1-r)^{-1}}{eh_1A_k(1-r)^{-1}+1-e}\right)$$

must hold. Since both sides of the last inequality are positive, this is equivalent to

$$\frac{e}{\dfrac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}} < \frac{eh_1A_k(1-r)^{-1}}{eh_1A_k(1-r)^{-1}+1-e},$$

the solution of which yields

$$h_1 < \frac{(1-r)(er+1-r)}{eA_kr}.$$

That proves the necessity of (5.2), since, obviously, $h_1>1$ is required as well.

From (5.2) now, it is clear that

$$1 < \frac{(1-r)(er+1-e)}{eA_kr}$$

must hold. But it can be easily verified that the last inequality is equivalent to

$$(1-e)r^2 - (eA_k+2-e)r + 1 > 0,$$

which, according to what was stated in Subsection 2.2.3, yields (5.1).

Clearly also, (5.3) is true, since (5.1) was obtained from the solution of $1 < h_{max}$, from (5.2). ■

5.4. Output of k-CNF Learning Algorithm from PEs

The complete analysis of Algorithm 5.1 is quite lengthy. For this reason, it has been split into several sections. The properties of the output of Algorithm 5.1 are studied in this section. ■

The following lemma will be primarily used in the proof of Theorem 5.2.

Lemma 5.1. The following is true:

$$\ln\left(\frac{ehA}{ehA+1-e}\right) \leq \frac{1}{A}\ln\left(\frac{eh}{eh+1-e}\right) \quad \forall h,A \geq 1.$$

Proof: Let h R, with h≥1. Consider the function

$$f : [1, \infty)\dashrightarrow R \quad \text{with} \quad f(A) = \frac{A\ln\left(\frac{ehA}{ehA+1-e}\right)}{\ln\left(\frac{eh}{eh+1-e}\right)}. \tag{5.4}$$

It can be verified that

$$f'(A) = \frac{1}{\ln\left(\frac{eh}{eh+1-e}\right)}\left\{\ln\left(\frac{ehA}{ehA+1-e}\right)+\frac{1-e}{ehA+1-e}\right\}. \tag{5.5}$$

Consider now the function

$$g : [1, \infty)\dashrightarrow R, \text{ with } g(A)=\ln\left(\frac{ehA}{ehA+1-e}\right)+\frac{1-e}{ehA+1-e}. \tag{5.6}$$

It can be verified that

$$g'(A) = \frac{(1-e)^2}{A(ehA+1-e)^2} > 0,$$

and therefore, according to (5.6),

$$g(A) : \text{increasing in } [1, \infty). \tag{5.7}$$

Also,

$$\lim_{A\dashrightarrow\infty} g(A) = 0. \tag{5.8}$$

(5.6) now, together with (5.7) and (5.8) imply that

$$g(A) \leq 0 \quad \forall A\geq 1. \tag{5.9}$$

Therefore, (5.5) implies that $f'(A)\leq 0$, and according to (5.4)

$$f(A) : \text{decreasing in } [1, \infty). \tag{5.10}$$

Obviously, also,

$$\lim_{A \to 1} f(A) = 1, \qquad (5.11)$$

and (5.4) now, together with (5.10) and (5.11) imply that

$$f(A) \leq 1 \quad \forall A \geq 1.$$

The result now follows immediately. ■

Theorem 5.2. The output g of Algorithm 5.1 has the properties:

(a) $P\left[\sum_{g(v)\neq 1} D^{+}(v) \geq h_1^{-1}\right] \leq h_2^{-1}$.

(b) $P\left[\sum_{g(v)\neq 0} D^{-}(v) > 0\right] \leq h_4^{-1}$.

(c) g is in n-CNF, for some $n \leq k$; g is not necessarily in maximal n-CNF.

Proof: Let $f=s_1 s_2 \ldots s_m$ be the concept to be learned in maximal k-CNF, and g be the final output of the algorithm.

(a) Let V_i be the set of PEs making 0/* at least one k-sum s_j (of the initial g in line 1), having COUNT[j] < dM, after the i-th iteration of the **repeat**-loop, for i=0,1,...,M. It can be seen that if v is a PE, then

$$\left(g(v)\neq 1\right) \Leftrightarrow \left(v \in V_M\right). \qquad (5.12)$$

Indeed, $v \in V_M$ holds, if and only if v makes 0/* at least one k-sum s_j, having COUNT[j] < dM after the M-th iteration. But this is equivalent to that v makes 0/* at least one k-sum of the output g of the algorithm, which, according to Theorem 3.3(c), is equivalent to $g(v)\neq 1$.

According to (5.12) now, it suffices to prove that

$$P\left[\sum_{v\in V_M} D^+(v) \geq h_1^{-1}\right] \leq h_2^{-1}. \qquad (5.13)$$

Each output, v, of r-EXAMPLE$^+$, can be interpreted as a Bernoulli trial, with "success" being "v makes at least one k-sum s_j, having currently COUNT[j] < dM, 0/*". That means that a trial is successful if and only if it contributes to the possible future deletion of a k-sum. Clearly, M trials have been performed, and the number of successes occurred is at most (dM)(A_k-m), and therefore, at most dMA$_k$.

If p (p') denotes the probability that r-EXAMPLE$^+$ will change a vector v from inside (outside) V_i to a vector v', not making (making) 0/* any (some) $s_j \in g$ having COUNT[j] < dM, then the probability of success of the i-th trial is

$$P[\text{success}] = P[v : \text{inside } V_i] * (1-p) + \\ + P[v : \text{outside } V_i] * p',$$

and therefore, since $r \geq p$,

$$P[\text{success}] \geq \left(\sum_{v\in V_i} D^+(v)\right)(1-r).$$

Since now $V_{i+1} \subseteq V_i$, for all i=0,1,...,M-1, the last inequality implies that

$$P[\text{success}] \geq \left(\sum_{v\in V_M} D^+(v)\right)(1-r).$$

Hence, the event

"$\sum_{v\in V_M} D^+(v) \geq h_1^{-1}$ holds after the last run"

implies that

"the probability of success of each of the M

> Bernoulli trials (in the last run) is at least $h_1^{-1}(1-r)$, and at most dMA_k successes have occurred",

and therefore,

$$P\left[\sum_{v\in V_M} D^+(v) \geq h_1^{-1}\right] \leq b^*\left(\leq dA_k M; M, \geq h_1^{-1}(1-r)\right) \quad (5.14)$$

But Theorem 2.5 implies that

$$\min\left\{n\in N \mid b^*\left(\leq dA_k n; n, \geq h_1^{-1}(1-r)\right) \leq h_2^{-1}\right\} \leq L_1, \quad (5.15)$$

where

$$L_1 = \left\lceil \frac{dA_k + \ln h_2}{\ln\left(\dfrac{eh_1(1-r)^{-1}}{eh_1(1-r)^{-1}+1-e}\right) - dA_k} \right\rceil, \quad (5.16)$$

provided that condition (2.34) of Theorem 2.5 is met, or, according to the definition of d in the algorithm, provided that

$$\frac{y(1+a)+b(1-x)}{a+b+x+y} A_k < \ln\left(\frac{eh_1(1-r)^{-1}}{eh_1(1-r)^{-1}+1-e}\right). \quad (5.17)$$

But Lemma 5.1 implies that in order to prove (5.17), it suffices to prove

$$\frac{y(1+a)+b(1-x)}{a+b+x+y} < y,$$

which is equivalent to

$$(b+x)(1-x-y) < 0,$$

which is always true, since $b+x>0$ by definition, and Theorem 5.1 implies that $M>0$, therefore $1-x-y<0$. That establishes (5.17), and therefore (5.15) as well. To complete now the proof, it suffices to show that

$$L_1 \leq M. \tag{5.18}$$

But Lemma 5.1 again implies that

$$\frac{A_k}{\ln\left(\dfrac{eh_1(1-r)^{-1}}{eh_1(1-r)^{-1}+1-e}\right)-dA_k} \leq \frac{1}{\ln\left(\dfrac{eh_1A_k(1-r)^{-1}}{eh_1A_k(1-r)^{-1}+1-e}\right)-d},$$

which, according to the definition of y in the algorithm, implies that

$$\left\lceil \frac{dA_k + \ln h_2}{\ln\left(\dfrac{eh_1(1-r)^{-1}}{eh_1(1-r)^{-1}+1-e}\right) - dA_k} \right\rceil \leq \left\lceil \frac{d + \ln h_2}{y - d} \right\rceil. \tag{5.19}$$

According now to (5.16), the left-hand side of the last inequality is equal to L_1, and according to the definitions of b, d, and M in the algorithm, the right-hand side can be verified to be is equal to M. That establishes (5.18), which completes the proof, since (5.15) implies that

$$b^*\left(\leq dA_kM;\ M,\ \geq h_1^{-1}(1-r)\right) \leq h_2^{-1},$$

which, according to (5.14), proves (5.13).

(b) The event

$$\text{"}\sum_{g(v)\neq 0} \mathbb{D}^-(v) > 0\text{"}$$

is equivalent to

"there exist a vector v, such that $f(v)=0$ and $g(v)\neq 0$",

or, according to Theorem 3.3(a), is equivalent to

"there exist a vector v, such that $s_i(v)=0$ for some $i\in\{1,2,\ldots,m\}$, and $g(v)\neq 0$".

That implies (but is not equivalent to) that

"at least one of $s_1,\ldots,s_m$ is missing from g".

But that implies that r-EXAMPLE$^+$ made at least dM errors (again, the inverse implication is not true, since what is required is that at least dM errors will affect the same k-sum).

If each output, v, of r-EXAMPLE$^+$ is interpreted as a Bernoulli trial, with "success" being "v is not erroneous", then M trials have been performed, each with probability of success at least 1-r, and at most M-dM successes have occurred (since at least one of $s_1,\ldots,s_m$ is missing from g). Therefore,

$$P\left[\sum_{g(v)\neq 0} D^-(v)>0\right] \leq b^*\left(\leq(1-d)M;\ M,\ \geq(1-r)\right) \tag{5.20}$$

But Theorem 2.5 implies that

$$\min\left\{n\in N \mid b^*\left(\leq(1-d)n;\ n,\ \geq(1-r)\right)\leq h_4^{-1}\right\}\leq L_2, \tag{5.21}$$

where

$$L_2 = \left\lceil \frac{(1-d)+\ln h_4}{\ln\left(\dfrac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}\right)-(1-d)} \right\rceil, \tag{5.22}$$

since it can be verified that condition (2.34) of Theorem 2.5 holds. Indeed, according to the definition of d in the algorithm, it suffices to prove that

$$1-\frac{y(1+a)+b(1-x)}{a+b+x+y}< x. \tag{5.23}$$

But this is equivalent to

$$(a+x)(1-x-y) < 0,$$

which is always true, since M>0. That establishes (5.21).

According now to the definitions of d, a, x, and M in the algorithm, it can be verified that

$$L_2 = M. \tag{5.24}$$

From (5.21) now, and (5.24), it can be derived that

$$b^*\left(\le(1-d)M;\ M,\ \ge(1-r)\right) \le h_4^{-1},$$

which, together with (5.20) prove the result.

(c) Since, according to what was said in (b), some of $s_1,\dots,s_m$ may be missing from g, g may not have any k-sums with k literals, i.e. g may be in (not necessarily maximal) n-CNF for some n<k. ■

Theorem 5.2 implies that Algorithm 5.1 performs a special kind of strong two-sided-error learning (see Definition 3.19). The question which naturally arises, is whether by allowing

$$P\left[\sum_{g(v)\neq 0} D^-(v) \ge h_3^{-1}\right] \le h_4^{-1}, \tag{5.25}$$

instead of part (b) of the theorem, a lower execution time, or a wider range of permissible values of r and h_1, could be achieved.

Unfortunately, the answer to this question is "no". **Even the inclusion of a single NE may be disastrous.** If, for example, $D^-(v)=1$, for a vector v included in g, since g is not tested against NEs, such an error will never be detected. ■

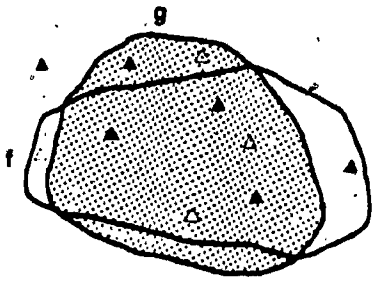


Figure 5.2. Venn diagram of f and g of Algorithm 5.1. A △ stands for a non-seen vector, whereas a ▲ stands for a vector presented as a PE.

A Venn diagram of the concept f to be learned and the output g of Algorithm 5.1 is shown in Figure 5.2.

In contrast to Algorithm 4.1, some **seen (real) PEs may not be included in g** now. This will happen if a PE is seen only a few times, and no other PE encompassing it is seen sufficient number of times. Algorithm 5.1 will then think that this PE is actually a NE falsely reported as a PE. But on the other hand, as is also the case with Algorithm 4.1, some non-seen PEs may be included in g, due to the known fact that f can be written in k-CNF (see Figure 4.3(e)).

Independently of the reason for which a PE may or may not be included in g, Theorem 5.2(a) guarantees that the probability h_2^{-1}, that the weight of the falsely non-included PEs will be greater than a certain quantity

h_1^{-1}, can be made arbitrarily small. However, (5.2) implies that the quantity h_1^{-1} cannot become arbitrarily small.

In contrast also to Algorithm 4.1, Algorithm 5.1 may over-generalize, i.e. g may contain some NEs as well. This will happen if a NE, or a number of NEs encompassing it, is repeatedly presented as a PE; if a NE is presented only a few times as a PE, then it will not be included in g, due to the d threshold, except if it is forced to, by the known fact that f can be written in k-CNF, as explained above. The last fact may result in some more unfortunate situations; it may force, for example, **a NE, not even presented as a PE, to be falsely included in g.**

Independently again of the reason of the false non-exclusion of a NE from g, Theorem 5.2(b) guarantees that the probability h_4^{-1} that even a single NE will be falsely non-excluded from g can become arbitrarily small. ■

Let's look now at the very important consequences of part (c) of the last theorem, although they are immediate after our Theorems 3.1 and 3.3. The output g of Algorithm 5.1 **can determine fast** (in at most $O(t^{k+1})$ time) **whether g(v)=1 for any vector v, and whether g(v)=0 for any total vector v. However, determining whether g(v)=0 for a partial vector v, is NP-hard.** ■

5.5. Discussion on k-CNF Learning Algorithm from PEs

A justification of the seemingly ad hoc choice of d and M in Algorithm 5.1, as well as a logical explanation of the fact that r and h_1 are upper-bounded, is given first. Those remarks give us a deeper insight into the problem and its solution, and also reassure us that the obtained results are meaningful.

Next, a remark is made on the value of M; M determines the time complexity of Algorithm 5.1, and its asymptotic behavior is examined in the next section. ■

5.5.1 Choice of d and M

Suppose that d is unknown. Then, from the proof of Theorem 5.2 (see (5.16), (5.17), and (5.22), (5.23)), it is obvious that:

(a) In order for property (a) of Theorem 5.2 to hold, at least

$$L_1 = \left\lceil \frac{dA_k + \ln h_2}{\ln\left(\frac{eh_1(1-r)^{-1}}{eh_1(1-r)^{-1}+1-e}\right) - dA_k} \right\rceil \tag{5.26}$$

iterations are required, provided that

$$dA_k < \ln\left(\frac{eh_1(1-r)^{-1}}{eh_1(1-r)^{-1}+1-e}\right).$$

(b) In order for property (b) of Theorem 5.2 to hold, at least

$$L_2 = \left\lceil \frac{1 - d - \ln h_4}{\ln\left(\frac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}\right) - (1-d)} \right\rceil \quad (5.27)$$

iterations are required, provided that

$$1-d < \ln\left(\frac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}\right).$$

Therefore, in order for both (a) and (b) of Theorem 5.2 to hold, at least $\max\{L_1,L_2\}$ iterations are required, provided that

$$d_1 < d < d_2,$$

where $d_1 = 1 - \ln\left(\frac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}\right)$

and $d_2 = \frac{1}{A_k} \ln\left(\frac{eh_1(1-r)^{-1}}{eh_1(1-r)^{-1}+1-e}\right)$. (5.28)

That means that **Theorem 5.2 holds for all d in the range specified by (5.28)**. The problem now is to find the "best" d, minimizing the number of iterations, $\max\{L_1,L_2\}$.

If L_1 and L_2 are considered as functions of d, then L_1 is increasing, and L_2 is decreasing. That implies that the d satisfying the condition $L_1=L_2$ is the best choice, i.e. it minimizes the $\max\{L_1,L_2\}$ (see Figure 5.3).

Although the solution d of the equation $L_1=L_2$ can be obtained, it leads to very tedious computations subsequently. But Lemma 5.1, formulated for this very

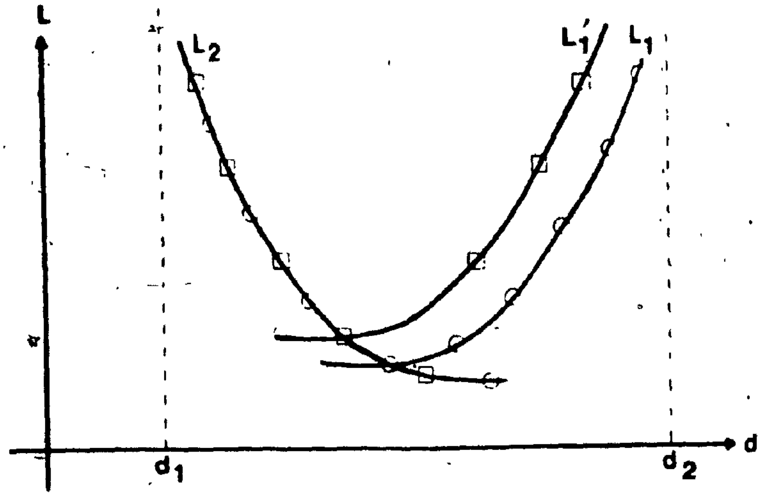


Figure 5.3. Graph of $L_1=L_1(d)$, $L_2=L_2(d)$, and $L_1'=L_1'(d)$.

The —⊖—⊖— line is the $\max\{L_1,L_2\}$. The —⊟—⊟— line is the $\max\{L_1',L_2\}$.

purpose, implies (see derivation of (5.19)) that $L_1 \le L_1'$, where

$$L_1' = \left\lceil \frac{d + \ln h_2}{\ln\left(\frac{eh_1A_k(1-r)^{-1}}{eh_1A_k(1-r)^{-1}+1-e}\right) - d} \right\rceil . \qquad (5.29)$$

Obviously now, in order for (a) and (b) of Theorem 5.2 to hold, at least $\max\{L_1',L_2\}$ iterations are required, provided that

$$d_{\min} < d < d_{\max},$$

$$\text{where} \quad d_{min} = 1 - \ln\left(\frac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}\right)$$

$$\text{and} \quad d_{max} = \ln\left(\frac{eh_1A_k(1-r)^{-1}}{eh_1A_k(1-r)^{-1}+1-e}\right). \qquad (5.30)$$

Again, the d satisfying $L_1'=L_2$ is the best choice, and the d of Algorithm 5.1 is this very d, which when substituted in L_1' in (5.29), or in L_2 in (5.27), yields the M of the algorithm. It has to be remembered though, that a **lower execution time could have been obtained if the equation $L_1=L_2$ had been solved instead.** ■

5.5.2. Bounds on r and h_1

The reason for which r and h_1 cannot grow beyond a certain value, as implied by (5.1) and (5.2), **even in the expense of running time** (in that case Algorithm 5.1 simply doesn't work), will be explained here.

The rationalization of those two facts, already rigorously proven in Theoren 5.1, will be faciliated by our discussion in the last subsection.

Clearly, property (a) of Theorem 5.2 implies that the leftmost inequality of (5.30) has to be true, which, in turn implies that

$$\text{when } r \text{ increases, } d_{min} \text{ increases.} \qquad (5.31)$$

This is so, because when the example generator makes more errors, more counters of the k-sums of f tend to falsely become non-zero. Therefore, the algorithm has to ask for more evidence that a k-sum has to be deleted from g, since the maximum probability h_4^{-1}, that even one k-sum of f will be falsely deleted, remains fixed.

Property (b) of Theorem 5.2 implies that the rightmost inequality in (5.30) has to be true. That implies that

when r increases, d_{max} decreases. (5.32)

This is so, because when the example generator makes more errors, more counters of the k-sums not in f tend to falsely decrease, or become zero, implying that more PEs tend to be falsely non-included in g. Therefore, the algorithm should require less evidence that a k-sum has to be deleted from f, since the maximum weight of the falsely non-included PEs has not been increased.

The same inequality also implies that

when h_1 increases, d_{max} decreases. (5.33)

That happens, because as the user becomes more fussy about the weight of the PEs falsely non-included in g, or equivalently, about the number of k-sums falsely included in g, the algorithm has to decrease its threshold, so that it will not be fooled too much by the errors of the example generator, i.e. it will not falsely keep too many k-sums in g, which it shouldn't.

The last implication of the rightmost inequality of (5.30) is that

when A_k increases, d_{max} decreases. (5.34)

The reason for this is that as A_k increases, since the weight of the PEs (falsely non-included in g, and therefore) making 1 the k-sums not in f remains fixed, it becomes less probable that a k-sum not in f will be made 1 by a PE. Therefore, in order for the algorithm not to be fooled too much (by keeping too many k-sums which it shouldn't), it has to decrease its threshold.

The reason now for which r is upper-bounded is obvious. (5.31) and (5.32), already rationalized, imply that as r increases, the range of permissible values of d, in (5.30), is shrinking down from both sides. Therefore, when r grows beyond a certain value, it can be expected that $d_{max}<d_{min}$ will occur, which allows no value for d. Therefore, r should not be allowed to grow arbitrarily.

On the other hand, if h_1 increases, (5.33) implies that the range of d, in (5.30), is shrinking down again, but only from the right side this time, and therefore the undesirable situation $d_{max}<d_{min}$ may occur again. To prevent that, h_1 shouldn't be allowed to grow arbitrarily.

But what about (5.34)? Doesn't it imply that A_k cannot grow arbitrarily? The answer is "yes", provided that r is

fixed, so that d_{min}, in (5.30), remains fixed. However, since this is undesirable, r will not be allowed to be fixed, as it will be explained in the next section; r will be somehow related to A_k, so that when A_k grows, i.e. when d_{max} decreases, d_{min} will decrease too, so that $d_{min}<d_{max}$ will always hold. ■

5.5.3. Conservatism of M

The number of iterations of Algorithm 5.1 dominates the algorithm's time complexity. Therefore, before studying M's asymptotic behavior, it does make sense to check whether a lower M is sufficient for most of the cases (i.e. for most of the sequences of the PEs presented), and if so, to find the circumstances under which a number of iterations as large as M is absolutely necessary.

From the proof of Theorem 5.2, it should be obvious that M is very conservative; it is large enough to guarantee that properties (a) and (b) of the theorem hold in the **worst case**. This case arises when **all** PEs are turned to non-PEs by the noisy example generator (because if a PE is turned to another PE, none of the counters of the k-sums of f will be falsely increased), each of those non-PEs affects the **same** k-sum of f (because the increase of many counters up to dM-1 is of no harm), and moreover, the deletion of this particular k-sum does affect g (the false deletion, for example, of x_1 from $g = x_1x_2(x_1+x_2')$ does not affect g).

For this reason, when r-->0, the d of Algorithm 5.1 does not tend to 1, i.e. Algorithm 5.1 does not tend to Algorithm 4.1. ■

5.6. Time Complexity of k-CNF Learning Algorithm from PEs

The time complexity of Algorithm 5.1 will be studied in this section. As already mentioned in the last subsection, the key point to that is the derivation of an asymptotic expression for the number of iterations M, i.e. the study of the behavior of M, when A_k (equivalently: when t), h_1, h_2, and h_4 are growing.

One could say that in an asymptotic expression for M, the error rate r should be a parameter; the more desirable idea of studying the behavior of M when $\frac{1}{1-r}$ --> ∞, is not applicable, because, according to (5.1), r is upper-bounded, i.e. r cannot always approach the value 1. However, a more careful examination of (5.1) reveals that as A_k increases, r_{max} decreases. That implies that if r is a parameter, i.e. if r has a fixed value, **A_k cannot grow beyond a certain value**, therefore, no value of M exists when A_k --> ∞.

One more problem, concerning the asymptotic behavior of M, arises from (5.2). According to that, h_1 also cannot grow arbitrarily, and therefore, no M exists when $h_1 \dashrightarrow \infty$. Moreover, the permissible values of h_1 are harder to study, since they depend on both A_k and r, with r itself depending on A_k.

The above remarks suggest that before the study of the asymptotic behavior of M is attempted, the relation between r, h_1, and A_k has to be fully understood. Lemma 5.2 examines the relation between r and A_k, which seems to be more tractable than that between r, h_1, and A_k, examined in Lemma 5.3. ■

<u>Lemma 5.2.</u> For

$$r_{max} = \frac{\sqrt{(eA_k+2-e)^2+4(e-1)} - (eA_k+2-e)}{2(e-1)}$$

the following are true:

(a) $r_{max}(A_k)$: decreasing $\forall A_k \geq 2$.

(b) $\frac{.3679}{A_k} \simeq \frac{1}{e}\frac{1}{A_k} \leq r_{max} \leq \frac{\sqrt{e^2-8e}\ -e-2}{e-1}\frac{1}{A_k} \simeq \frac{.3954}{A_k}$,

hence

$r_{max} = \Theta\left(\frac{1}{A_k}\right)$.

Moreover, r_{max} approaches its lower bound, when $A_k \dashrightarrow \infty$.

Proof:

(a) In order to show that $r_{max}(A_k)$ is decreasing, it suffices to show that its first derivative is negative. But

$$r_{max}'(A_k) = \frac{e}{2(e-1)}\left(\frac{eA_k+2-e}{\sqrt{(eA_k+2-e)^2+4(e-1)}} - 1\right),$$

and since $4(e-1)>0$,

$$r_{max}'(A_k) < 0 .$$

(b) Upper and lower bounds for

$$A_k r_{max}(A_k) = \frac{A_k\left((eA_k+2-e)^2+4(e-1)-(eA_k+2-e)\right)}{2(e-1)} \qquad (5.35)$$

will be derived. First, it will be shown that

$$A_k r_{max}(A_k) \ : \text{decreasing in } [2, \infty). \qquad (5.36)$$

To do so, it suffices to prove that

$$(A_k r_{max}(A_k))' < 0,$$

or, equivalently, that

$$\frac{1}{2(e-1)}\left(\frac{A_k(eA_k+2-e)(2eA_k+2-e)+4(e-1)A_k}{\sqrt{A_k(eA_k+2-e)^2 + 4(e-1)A_k^2}} - (2eA_k+2-e)\right) < 0$$

which is equivalent to

$$\frac{A_k(eA_k+2-e) + \dfrac{4(e-1)A_k}{2eA_k+2-e}}{\sqrt{A_k(eA_k+2-e)^2+4(e-1)A_k^2}} < 1.$$

By considering the square of both sides of the last inequality, and by performing some algebraic operations, the

$$A_k > \frac{4(e-1)+(e-2)^2}{2e(e-2)} \simeq 1.8922$$

is derived, which is always true, since $A_k \geq 2$. That proves (5.36), which, in turn, implies that

$$\lim_{A_k \to \infty}\left(A_k r_{max}(A_k)\right) \le A_k r_{max}(A_k) \le 2r_{max}(2) \quad (5.37)$$

But (5.35) implies that

$$2r_{max}(2) = \frac{e^2+8e - e - 2}{e-1}. \qquad (5.38)$$

Also, multiplication of the numerator and the denominator of $A_k r_{max}(A_k)$ in (5.35) with the conjugate of the numerator yields

$$A_k r_{max}(A_k) = \frac{2A_k}{\sqrt{(eA_k+2-e)^2+4(e-1)} + (eA_k+2-e)},$$

or

$$A_k r_{max}(A_k) = \frac{2}{\sqrt{\left(\frac{eA_k+2-e}{A_k}\right)^2 + \frac{4(e-1)}{A_k^2}} + \frac{eA_k+2-e}{A_k}}.$$

Therefore,

$$\lim_{A_k \to \infty}\left(A_k r_{max}(A_k)\right) = \frac{1}{e}.$$

The last equality now, together with (5.37) and (5.38) prove the result. ■

The relation between r_{max}, the maximum acceptable error rate by Algorithm 5.1, and its bounds

$$r_1 = \frac{1}{e}\frac{1}{A_k}$$

and

$$r_2 = \frac{\sqrt{e^2+8e} - e - 2}{e-1}\frac{1}{A_k},$$

as established in Lemma 5.2, is shown in Figure 5.4. Any error rate, lying in the shaded area (depending on A_k) is acceptable by Algorithm 5.1. ■

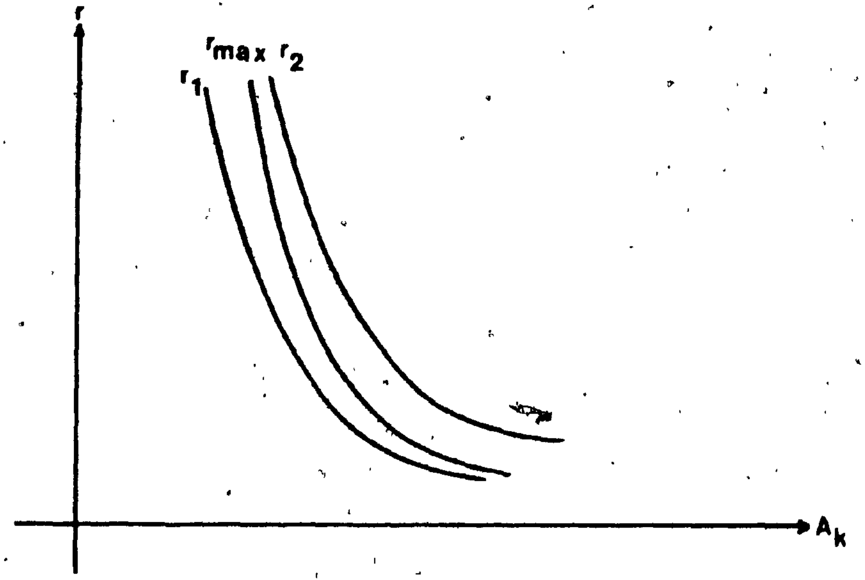


Figure 5.4. Graph of r_{max}, and its bounds r_1 and r_2.

Lemma 5.2 is very important, because it allows us to set

$$r = \frac{c}{A_k}, \quad \text{for some } c < \frac{1}{e}, \qquad (5.39)$$

when studying the asymptotic time complexity of Algorithm 5.1, **without restricting it to a particular range of acceptable error rates.** Other values of r, with

$$r \geq \frac{1}{eA_k}$$

are acceptable error rates as well (if they are lying between r_{max} and r_1 in Figure 5.4), but only for small values of A_k. Hence, they need not be considered. ■

As was already stated in the beginning of the section, the relation between h_1, r, and A_k has to be studied next. Since this result will be used in M's asymptotic expression, according to what was said above, it can be assumed, without loss of generality, that (5.39) holds.

<u>Lemma 5.3.</u> If c is a constant, with

$$c < \frac{1}{e} \tag{5.40}$$

and

$$r = \frac{c}{A_k}, \quad \text{for all } A_k \tag{5.41}$$

and also

$$h_{max} = \frac{(1-r)(er+1-r)}{eA_k r},$$

then, the following are true:

(a) $h_{max}(A_k)$: decreasing $\forall A_k \geq 2$.

(b) $\frac{1}{ec} \leq h_{max} \leq \frac{(2-c)(2+ec-c)}{4ec}$.

Moreover, h_{max} approaches its lower bound, when $A_k \dashrightarrow \infty$.

Proof: According to (5.41), the definition of h_{max} can be rewritten as

$$h_{max} = h_{max}(A_k) = \frac{(A_k-c)(ec+A_k-c)}{ecA_k^2}, \tag{5.42}$$

or

$$h_{max}(A_k) = \frac{A_k^2 + c(e-2)A_k - c^2(e-1)}{ecA_k^2}. \tag{5.43}$$

(a) To show that $h_{max}(A_k)$ is decreasing, it suffices to show that its first derivative is negative. But

$$h_{max}'(A_k) = \frac{A_k\left(-e(e-2)c^2A_k+2e(e-1)c^3\right)}{e^2c^2A_k{}^4}.$$

Therefore

$$h_{max}'(A_k) < 0 \qquad (5.44)$$

is equivalent to

$$A_k > \frac{2(e-1)c}{e-2},$$

which is always true, since $A_k \geq 2$, and, according to (5.40),

$$\frac{2(e-1)c}{e-2} < \frac{2(e-1)}{e(e-2)} \simeq 1.76.$$

That proves (5.44), as required.

(b) Part (a) implies that

$$\lim_{A_k \dashrightarrow \infty}\left(h_{max}(A_k)\right) \leq h_{max}(A_k) \leq h_{max}(2). \qquad (5.45)$$

But (5.42) implies that

$$h_{max}(2) = \frac{(2-c)(ec+2-c)}{4ec} \qquad (5.46)$$

and (5.43) implies that

$$\lim_{A_k \dashrightarrow \infty}\left(h_{max}(A_k)\right) = \frac{1}{ec}. \qquad (5.47)$$

The result now follows from (5.45), (5.46), and (5.47).■

The relation between h_{max}, the maximum value of the user-specified parameter h_1 acceptable by Algorithm 5.1, and its bounds, as established in Lemma 5.3, is shown in Figure 5.5. ■

As was also the case with Lemma 5.2, Lemma 5.3 allows

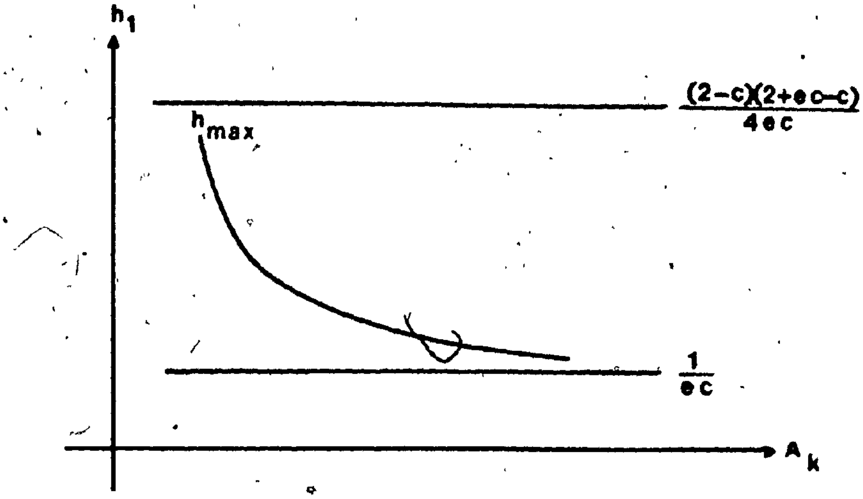


Figure 5.5. Graph of h_{max} and its bounds.

us to assume that $h_1 < \frac{1}{ec}$ holds, when studying the asymptotic time complexity of Algorithm 5.1, **without restricting it to a particular range of permissible values of h_1.** ■

The time complexity of Algorithm 5.1 can now be derived.

<u>Theorem 5.3.</u> If c is a constant, with

$$c < \frac{1}{e} \tag{5.48}$$

and

$$r = \frac{c}{A_k} \quad \text{for all } A_k \tag{5.49}$$

is an error rate acceptable by Algorithm 5.1, and also

$$h_1 < \frac{1}{ec} \tag{5.50}$$

holds for the user-specified parameter h_1, then the worst-case time complexity of Algorithm 5.1 is

$$T(t,h_2,h_4) = O\left(\frac{h_1 t^{2k+1} \ln(h_2 h_4)}{1-ech_1}\right) \tag{5.51}$$

and

$$T(t,h_2,h_4) = \Omega\left(\frac{h_1 t^{3k+1} \ln(h_2 h_4)}{(1-ch_1)t^k-c}\right). \tag{5.52}$$

Proof: Upper and lower bounds of the expression

$$E = \ln\left(\frac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}\right)+\ln\left(\frac{eh_1A_k(1-r)^{-1}}{eh_1A_k(1-r)^{-1}+1-e}\right)-1, \tag{5.53}$$

i.e. the denominator of M, will be derived first. By using elementary properties of the logarithms, it can be verified that when (5.49) holds, (5.53) implies

$$E=\ln\left(\frac{eh_1A_k^3}{eh_1A_k^3-(e-1)(1-ech_1)A_k^2-(e^2-3e+2)cA_k+c^2(e-1)^2}\right) \tag{5.54}$$

which, according to (2.2), implies

$$E \geq \frac{(e-1)(1-ech_1)A_k^2+(e^2-3e+2)cA_k-c^2(e-1)^2}{eh_1A_k^3}. \tag{5.55}$$

But

$$(e^2-3e+2)cA_k - c^2(e-1)^2 > 0 \tag{5.56}$$

is equivalent to

$$A_k > \frac{c(e-1)^2}{e^2-3e+2},$$

which is always true, since $A_k \geq 2$, and according to (5.48)

$$\frac{c(e-1)^2}{e^2-3e+2} < \frac{(e-1)^2}{e(e^2-3e+2)} \simeq .88.$$

That establishes (5.56), and therefore (5.55) implies

$$E \geq \frac{(e-1)(1-ech_1)}{eh_1A_k},$$

or, according to (5.50),

$$\frac{1}{E} \leq \frac{eh_1A_k}{(e-1)(1-ech_1)} \; . \tag{5.57}$$

According now to Lemma 2.6, (5.53) implies

$$E \leq (1-r) + \frac{1-r}{h_1A_k} - 1,$$

which, according to (5.49), yields

$$E \leq \frac{(1-ch_1)A_k - c}{h_1A_k{}^2} \; . \tag{5.58}$$

But

$$(1-ch_1)A_k - c > 0 \tag{5.59}$$

is, according to (5.50), equivalent to

$$A_k > \frac{c}{1-ch_1},$$

which is always true, since $A_k \geq 2$, and (5.48) and (5.50) imply that

$$\frac{c}{1-ch_1} < \frac{\frac{1}{e}}{1-\frac{1}{e}} \simeq .58.$$

That establishes (5.59), and therefore (5.58) implies that

$$\frac{1}{E} \geq \frac{h_1A_k{}^2}{(1-ch_1)A_k-c} \; . \tag{5.60}$$

According now to the definition of M in the algorithm, (5.57) and (5.60) imply

$$\frac{h_1A_k{}^2(\ln(h_2h_4)+1)}{(1-ch_1)A_k - c} \leq M \leq \frac{e}{e-1}\,\frac{h_1A_k(\ln(h_2h_4)+1)}{1 - ech_1},$$

or, according to (4.1),

$$M = O\left(\frac{h_1 t^k \ln(h_2 h_4)}{1-ech_1}\right) \tag{5.61}$$

and

$$M = \Omega\left(\frac{h_1 t^{2k} \ln(h_2 h_4)}{(1-ch_1)t^k - c}\right). \tag{5.62}$$

It can now be seen that lines 11 to 14 of the Algorithm 5.1 take at most $\Theta(t^{k+1})$ time (the proof is as in Theorem 4.3). Since they are executed exactly M times, (5.61) and (5.62) imply that execution of lines 9 to 15 takes, in the worst case,

$$O\left(\frac{h_1 t^{2k+1} \ln(h_2 h_4)}{1-ech_1}\right)$$

and

$$\Omega\left(\frac{h_1 t^{3k+1} \ln(h_2 h_4)}{(1-ch_1)t^k - c}\right)$$

time. Clearly, lines 1, 2, and 16 require at most $\Theta(t^k)$ time, and lines 3 to 8 require $\Theta(1)$ time. Therefore, none of them increases the overall running time, and the result follows. ■

Theorem 5.3 is very important, since it implies that even if noisy PEs of a concept f are fed into a learning system, provided that the noise is not too high, a probably good approximation of f, and not merely **of the concept induced by the examples presented as PEs,** can be learned in time polynomial in the number of variables t. Moreover, the algorithm is applicable, i.e. when (5.66) and (5.68) hold. To see that, our M and his M^* have to be compared directly, and

$$\frac{2\ln h + 1}{\ln\left(\frac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}\right)+\ln\left(\frac{ehA_k(1-r)^{-1}}{ehA_k(1-r)^{-1}+1-e}\right)-1} <$$

$$< 36hA_k\ln(hA_k) \qquad (5.71)$$

has to be proven. But, according to (5.68) and the established (5.57), it suffices to prove

$$\frac{2e\ln h + e}{(e-1)(4-e)} < 9\ln(hA_k),$$

or

$$\frac{2e\ln(hA_k) + e}{(e-1)(4-e)} < 9\ln(hA_k).$$

But this is equivalent to

$$.19 = \frac{e}{9(e-1)(4-e)-2e} < \ln(hA_k),$$

which is always true, since $A_k \geq 2$, therefore $\ln(hA_k) > .69$. That proves (5.71), and therefore our claim has been established. ■

The reason for the drastic improvement of our algorithm over Valiant's is discussed next.

If we parallelize Valiant's proof in [VALIA85] with

---

this are important, even when the asymptotical superiority of an algorithm has been shown.

ours, we can see that instead of him deriving (5.14) and (5.20), he derived the much weaker inequalities

$$P\left[\sum_{v\in V_M} D^+(v) \ge h_1^{-1}\right] \le A_k b^*\left(\le dM; M, \ge (h_1A_k)^{-1}(1-r)\right) \quad (5.72)$$

and

$$P\left[\sum_{g(v)\ne 0} D^-(v) > 0\right] \le A_k b^*\left(\le (1-d)M;\ M,\ \ge (1-r)\right), \quad (5.73)$$

respectively.

To derive (5.72), he considered each PE seen as a Bernoulli trial, with "success" being "k-sum s_i is made 0/* by the output of r-EXAMPLE$^+$". Also, he observed that each $v\in V_M$ makes 0/* at least one falsely included k-sum in g, that there are at most A_k such falsely included k-sums in g, and that each of them is made 0/* only by PEs belonging to V_M, and by at least one such v V_M. Hence, at least one of the falsely included k-sums in g has probability at least $(h_1A_k)^{-1}$ to be made 0/* by a PE (in V_M). Therefore,

$$P\left[\sum_{v\in V_M} D^+(v) \ge h_1^{-1}\right] \le \sum_{i=m+1}^{m+q}\left(P\begin{bmatrix} s_i \text{ has probability} \\ \ge (h_1A_k)^{-1} \text{ to be} \\ \text{made 0/* by a PE}\end{bmatrix}\right),$$

where $s_{m+1},\ldots,s_{m+q}$ are the k-sums falsely included in g. Clearly, for each of them, less than dM successes have occurred, which implies (5.72).

To derive (5.73), he considered each PE seen as a Bernoulli trial, with "success" being "no error made by r-EXAMPLE$^+$" (as we did), but next he used the weaker inequality

$$P\left[\sum_{g(v)\neq 0} D^{-}(v)>0\right] \leq \sum_{i=1}^{m} P[s_i : \text{missing from } g],$$

which, obviously, implies (5.73).

It is worth noticing though that even if we were to use (5.72) and (5.73), instead of (5.14) and (5.20), our number of iterations would be

$$\left\lceil \frac{\ln h_2 h_4 A_k{}^2 + 1}{\ln\left(\frac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}\right)+\ln\left(\frac{eh_1 A_k (1-r)^{-1}}{eh_1 A_k (1-r)^{-1}+1-e}\right)-1} \right\rceil,$$

which can be proven (in a way similar to that (5.71) was proven) to be non-asymptotically better than Valiant's number of iterations M^*. Definitely, this improvement is due to our theorems in Chapter 2, which, in addition to that, have almost automated the derivation of the required number of iterations for our algorithms, as it became apparent from Subsection 5.5.1. ■

5.8. k-DNF Learning from NEs

The dual of Algorithm 5.1, performing k-DNF learning from noisy NEs is briefly presented in this section, followed by its analysis. The description of the algorithm is given first. ■

Algorithm 5.2.

Task : Learn f from noisy NEs.

Assumptions: (a) $x_1,\ldots,x_t$ are sufficient to express f.

(b) f can be written in k-DNF; k is known.

(c) Error rate r of NEs generator is known.

Parameters : h_2, h_3, $h_4 > 1$.

```
g <-- $\{p_i \mid p_i$ is a k-product, $\forall i=1,2,\ldots,A_k\}$
zero a counter COUNT[i] for each $p_i$
a <-- $\ln(h_2)$
b <-- $\ln(h_4)$
x <-- $\ln\left(\frac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}\right)$
y <-- $\ln\left(\frac{eh_3A_k(1-r)^{-1}}{eh_3A_k(1-r)^{-1}+1-e}\right)$
d <-- $\frac{y(1+a)+b(1-x)}{a+b+x+y}$
M <-- $\left\lceil\frac{a+b+1}{x+y-1}\right\rceil$
repeat M times
  begin
    v <-- r-EXAMPLE$^-$
    for each $p_i \epsilon g$ do
      if $p_i(v) \neq 0$
        then  COUNT[i] <-- COUNT[i]+1
  end
delete from g all $p_i$ having $\frac{\text{COUNT[i]}}{M} \geq d$ ■
```

The duals of Theorems 5.1, 5.2, and 5.3 hold for Algorithm 5.2. They are all stated here, for easy reference. First, the theorem concerning the maximum permissible values of r and h_3 is stated.

Theorem 5.4. Algorithm 5.2 is applicable if and only if

$$0 < r < r_{max} = \frac{\sqrt{(eA_k+2-e)^2+4(e-1)}-(eA_k+2-e)}{2(e-1)} \tag{5.74}$$

holds for the error rate r of the NEs generator, and also

$$1 < h_3 < h_{max} = \frac{(1-r)(er+1-r)}{eA_k r} \tag{5.75}$$

holds for the user-specified parameter h_3. Moreover,

$$r\text{-->}r_{max} \quad \text{if and only if} \quad h_{max}\text{-->}1 \tag{5.76}$$

Proof: As in Theorem 5.1. ■

The properties of the output of the algorithm are stated next.

Theorem 5.5. The output g of Algorithm 5.2 has the properties:

(a) $P\left[\sum_{g(v)\neq 1} D^+(v) > 0\right] \leq h_2^{-1}$.

(b) $P\left[\sum_{g(v)\neq 0} D^-(v) \geq h_3^{-1}\right] \leq h_4^{-1}$.

(c) g is in n-DNF, for some $n \leq k$; g is not necessarily in maximal n-DNF.

Proof: The proofs of (a), (b), and (c) are the duals of (b), (a), and (c) of Theorem 5.2. ■

This theorem implies that Algorithm 5.2 performs a special kind of strong two-sided-error learning. Some PEs, (encompassed by PEs) frequently presented as NEs, may be non-included in g. However, part (a) of the last theorem implies that the probability that even one PE of f will be falsely non-included in g, can become arbitrarily small (at most h_2^{-1}). On the other hand, the (NEs encompassed by) rarely seen, real, NEs may be non-excluded from g, but all frequently seen, real, NEs will be correctly excluded from g. Also, some non-seen NEs may be excluded from g. Part (b) of the theorem implies that the probability that the weight of the falsely non-excluded NEs from g will be greater than a certain quantity (h_3^{-1}) can become arbitrarily small (at most h_4^{-1}). However, according to Theorem 5.4, h_3^{-1} cannot become arbitrarily small.

The implications of part (c) of the same theorem are important. According to Theorem 3.2, for the output g of Algorithm 5.2, **it is NP-hard to determine whether g(v)=1 for a partial vector v;** what **can be determined fast** (in at most $O(n^{k+1})$ time) **is whether g(v)=1 for a total vector v, and whether g(v)=0 for any vector v.** ■

Lemmas 5.2 and 5.3 now imply that next theorem gives the asymptotic time complexity of Algorithm 5.2 for **the** entire range of permissible values of r and h_3.

<u>Theorem 5.6.</u> If c is a constant, with

$$c < \frac{1}{e} \tag{5.77}$$

and

$$r = \frac{c}{A_k} \quad \text{for all } A_k \tag{5.78}$$

is an error rate acceptable by Algorithm 5.2, and also

$$h_3 < \frac{1}{ec} \tag{5.79}$$

holds for the user-specified parameter h_3, then the worst-case time complexity of Algorithm 5.2 is

$$T(t,h_2,h_4) = O\left(\frac{h_3 t^{2k+1} \ln(h_2 h_4)}{1-ech_3}\right) \tag{5.80}$$

and

$$T(t,h_2,h_4) = \Omega\left(\frac{h_3 t^{3k+1} \ln(h_2 h_4)}{(1-ch_3)t^k - c}\right). \tag{5.81}$$

Proof: As in Theorem 5.3. ■

5.9. Summary

Polynomial two-sided-error learning algorithms, from noisy data, for k-CNF and k-DNF expressions, when k in known a priori, were presented in this chapter. The main idea for the design of those algorithms is due to L. G. Valiant, but besides that, our approach was entirely different.

Due to our own results in Chapter 2, we were able to design algorithms which are faster than Valiant's, both asymptotically and non-asymptotically, more flexible, and also applicable to a wider range of example generators' error rates.

Our own results of Section 3.3, also, implied that the output of the k-CNF (k-DNF) learning algorithm can always determine fast whether a vector is a PE (NE), but not whether it is a NE (PE). According also to the definition of learnability, the outputs of our algorithms are probably good approximations of the true concept, and not merely of the concept induced by the output of the noisy example generator. ■

Chapter 5 completed the study of k-CNF/k-DNF learning, if k is known, which started in Chapter 4. Admittedly though, the fact that the value of k is assumed to be known, before even the learning process starts, is unrealistic. Therefore, the next problem to be solved is that of learning a boolean concept, for which the value of **k is not known.** Chapter 6 examines this problem in its simplest case, i.e. when error-free examples are encountered. ■

CHAPTER 6

k-CONCEPT LEARNING; UNKNOWN k; TWO ERROR-FREE EXAMPLE GENERATORS

6.1. Introduction

The algorithms of Chapters 4 and 5 can learn an approximation of a k-CNF or k-DNF concept f, if the value of k in known. Although the task of designing and analyzing such algorithms was not easy at all, and therefore it was worth doing, it is admittedly much more realistic to assume that no such a priori information about the form of f will be available, i.e. that only a set of PEs and/or NEs will be presented to the learning machine. The task of approximate learning in such an environment is of much greater importance and practical use, and it has not been condidered at all by L. G. Valiant.

The simplest case of this problem in the one in which all PEs and NEs are error-free. This is the subject of this chapter. The cases in which one, or both, of the example generators are noisy, are discussed in Chapters 7 and 8. ■

The algorithms of Chapter 4 form the basis for the design of our algorithms in this chapter. Some unsuccessful modifications of the Chapter 4 algorithms are first attempted in Section 6.2. A CNF learning algorithm is presented in Section 6.3. Its dual algorithm, performing DNF learning, is given in Section 6.4. Section 6.5 contains a hybrid algorithm, learning either a CNF or a DNF.

A second version of each of those three algorithms is given in Sections 6.6, 6.8, and 6.9. A comparison between the algorithms of Sections 6.3 and 6.6 is made in Section 6.7. This comparison also holds for the algorithms of Sections 6.4 and 6.8, as well as for those of Sections 6.5 and 6.9.

The differences and similarities between the algorithms of Chapters 4 and 6, as well as the importance of the algorithms of this chapter, are summarized in Section 6.10.■

6.2. First Ideas

Instead of trying to design brand-new algorithms, performing k-CNF learning when k is not known, let's see first whether the algorithms of Chapter 4 can be modified, so that they can cope with the new problem of the unknown k.

One attempt to solve this problem could be to try first to learn (by some means) the exact value of k, and then to apply Algorithm 4.1. However, such an effort is out of the question, since it has always been assumed that only a proper subset of all PEs and/or NEs will be seen, fact which is prohibitive for the computation of the exact k.

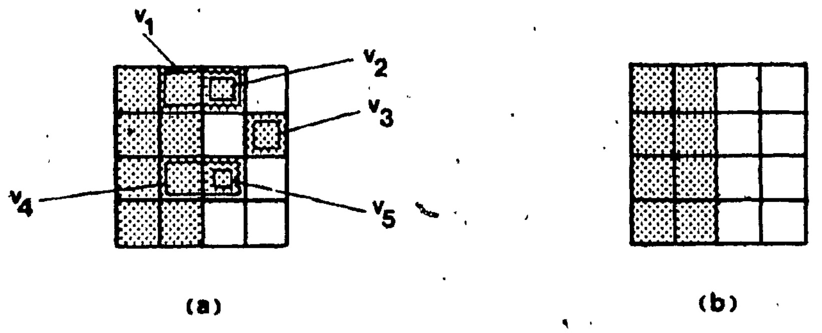


Figure 6.1. A close approximation of a k-CNF concept can considerably under-estimate k.

The next idea, naturally arising, would be to try first to approximate k (from the subset of all PEs and/or NEs which are seen), and then to apply Algorithm 4.1. To be more precise, our only hope would be to "approximate k with high probability", rather the just "approximate k"; the (rare) case in which PEs v with high $D^+(v)$ do not appear quite often, as it is expected, has to be taken care of. Unfortunately, even this cannot be done. If, for example, the PEs v_1, v_2, v_3, v_4, and v_5 of the 4-CNF concept in

Figure 6.1(a) are very rare, then its close approximation in Figure 6.1(b) is in 1-CNF. That implies that even a k'-CNF close approximation of a k-CNF concept may have k' much lower than k.

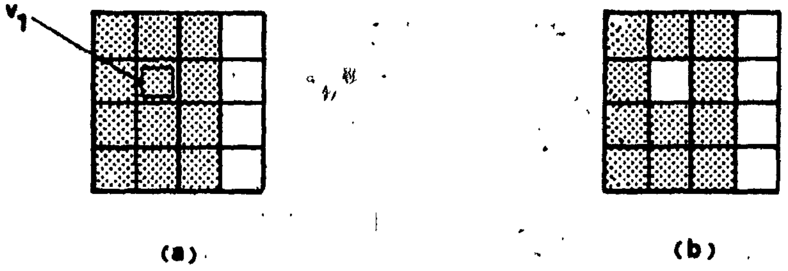


Figure 6.2. A close approximation of a k-CNF concept can considerably over-estimate k.

Is it then at least true that a k'-CNF close approximation of a k-CNF concept always has k' lower than k? Or, in other words, is it possible to restrict the range of the permissible values of k? As the example of Figure 6.2 shows, even that is not possible. If all 12 PEs of the 2-CNF concept shown in Figure 6.2(a), which contain the total vector v_1, are very rare, then its close approximation shown in Figure 6.2(b) is in 4-CNF. Therefore, k' may be much higher than k.

These large differences between k and k' can be explained by observing that given a k-CNF concept, any subset of its PEs may form a close approximation of it, depending on D^+. On the other hand, any subset of its PEs may either under-estimate or over-estimate k, depending on the relative position of the PEs included in this subset. Clearly, since D^+ is independent of the relative position of the PEs, **a close approximation of a k-CNF concept can considerably under-/over-estimate k.** ■

Since all our attempts to learn or approximate k have been fruitless, we will now try to **learn an approximation of a k-CNF concept without having, or being able to obtain, any information about k.**

The first idea towards this direction could be to learn a t-CNF (t is the number of variables in the concept), by applying Algorithm 4.1 for k=t. Although such an approach will definitely give the correct answer, it will not be pursued further, because it takes exponential time on t (see Theorem 4.3).

Another (working) idea for k-CNF learning, if k is not known and it cannot be learned or approximated, is presented in the next section. ■

6.3. CNF Learning - Case I

One reasonable way of attacking the problem of k-CNF learning, without having any idea about the value of k, is to see what happens when Algorithm 4.1 is applied, with g initialized to the set of all n-sums, for n=1,2,...,t. An example is shown in Figure 6.3. If the 3-CNF concept of Figure 6.3(a) is to be learned (when all PEs are seen) by applying Algorithm 4.1, then the outputs of the algorithm when g is initialized to the set of all 1-sums, 2-sums, and 3-sums are shown in Figures 6.3(b), 6.3(c), and 6.3(d), respectively. As it can be immediately seen, the lower n is, the more general g becomes. This should be expected, since the output of Algorithm 4.1, when g is initialized to the set of all n-sums, is a subset of (more general than) the output of the algorithm when g is initialized to the set of all (n+1)-sums.

Now, the main idea underlying our new algorithm, follows naturally. First, try to learn a 1-CNF expression from PEs, by applying Algorithm 4.1. Next, test this g against a sufficient number of NEs. If even one NE makes it 1/*, then over-generalization has occurred, and therefore at least 2-CNF is required. In such a case, try to learn a 2-CNF expression g, by applying Algorithm 4.1 again, test it against a sufficient number of NEs, etc. If, after n trials, no NE makes the n-CNF, learned from PEs, 1/*, then

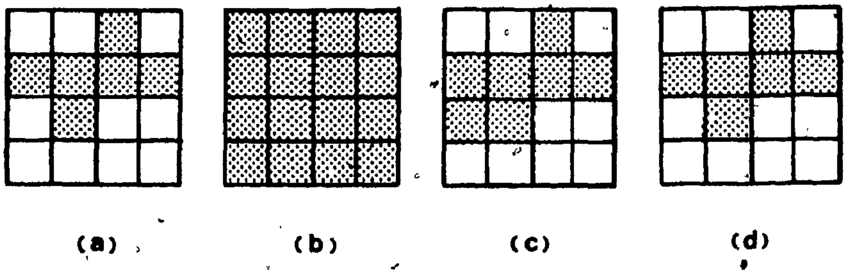


Figure 6.3. Trials for k-CNF learning without knowledge of k.

probably no over-generalization has occurred. Notice once more, that the word "probably" has to be included in the last statement, since the NEs indicating that over-generalization has occurred, may be very rare, and therefore they might not be seen at all. ■

The test of the learned n-CNF concept g against the NEs, in addition of being used by the CNF learning algorithm just described, will also be used by subsequent algorithms. For this reason, it is stated seperately below.

Function conflict-max-g-NE : boolean

Task : Return the value TRUE, if at least one of the seen error-free NEs of f makes g 1/*; otherwise return FALSE.

Assumptions: g is in maximal n-CNF, for some $1 \le n \le t$.

Parameters : h_3, h_4; set up in calling routine.

```
1. repeat $\left\lceil \frac{\ln h_4}{\ln\left(\frac{h_3}{h_3-1}\right)} \right\rceil$ times
2.  begin
3.     v <-- EXAMPLE
4.     if  $s_i(v)\neq 0$  $\forall s_i \in g$        [if $g(v)\neq 0$]
5.       then
6.         begin
7.           conflict-max-g-NE <-- TRUE
8.           STOP
9.         end
10.  end
11. conflict-max-g-NE <-- FALSE
```

■

It is very important to notice that, according to Theorem 3.3(a), the test of line 4 is equivalent to $g(v)\neq 0$, because g is **maximal.** The importance of that will become apparent in Chapter 8, when g will not be maximal any more, and the function above will have to be modified, to test for $g(v)=1$, instead of $g(v)\neq 0$. ■

Our algorithm can now be stated concisely.

<u>Algorithm 6.1.</u>

Task : Learn f from error-free PEs and NEs.

Assumptions: $x_1,\ldots,x_t$ are sufficient to express f.

Parameters : h_1, h_2, h_3, $h_4 > 1$.

```
1. n <-- 1                               [set n to 1]
2. call Algorithm 4.1 with k<--n         [learn n-CNF from PEs]
3. if conflict-max-g-NE                  [if over-generalized]
4.    then
5.      begin
6.        n <-- n+1                      [increment n]
7.        go to 2                        [and try again]
8.      end
```

Since all the algorithms of Chapters 6, 7, and 8 are addressing the problem of learning a k-CNF/k-DNF in the absence of knowledge of k, they all follow, in principle, the same method: they try to learn succesively a 1-CNF and/or a 1-DNF, a 2-CNF and/or a 2-DNF, etc., until they succeed (with success being appropriately defined and detected). The necessity for the following definition can now be easily understood.

<u>Definition 6.1.</u> Each trial of learning a n-CNF and/or a n-CNF, for n=1,2,...,t, is called a <u>run</u>.

For example, as far as Algorithm 6.1 ~~is~~ concerned, each execution of lines 2 to 8 is a run. In each run, g is initialized to the empty concept, which expands (is made more general) as new PEs are encountered (in line 2). The rate of expansion depends on the value of n. The final output g of the algorithm is nothing but the value of g after execution of line 2, i.e. g does not change (expands or shrinks) when tested against NEs (in line 3). ■

The following lemma, although simple, will be very useful.

Lemma 6.1. If f can be written in k-CNF, then the maximum number of runs of Algorithm 6.1 is k.

Proof: In Theorem 4.2(a) it was shown that when f can be written in k-CNF and g is initialized to the set of all k-sums, then the output g of Algorithm 4.1 will be more specific than f. That implies that when n=k, the function "conflict-max-g-NE" will yield the value FALSE, which proves the lemma. ■

The first result on the performance of Algorithm 6.1 concerns the quality of its output.

Theorem 6.1. The output g of Algorithm 6.1 has the properties:

(a) $P\left[\sum_{g(v)\neq 1} D^{+}(v) \geq h_1^{-1}\right] \leq h_2^{-1}$.

(b) $P\left[\sum_{g(v)\neq 0} D^{-}(v) \geq h_3^{-1}\right] \leq h_4^{-1}$.

(c) If f can be written in k-CNF, then g is in maximal n-CNF, for some $n \le k$.

Proof:

(a) The proof is similar to that of Theorem 4.2, and it is briefly sketched here. Suppose that f can be written in k-CNF, and that n is the number of runs executed. Let also g_0 be the initial value of g, when Algorithm 4.1 is called in the n-th run, and let g_i be the value of g after the i-th execution of the repeat-loop of Algorithm 4.1, when it is called in the n-th run, for $i=1,2,\ldots,L(h_1,h_2,A_n)$. Let also define

$$X_i = \sum_{g_i(v)\neq 1} D^+(v) \qquad \text{for } i=1,2,\ldots,L(h_1,h_2,A_n),$$

i.e. X_i values are defined only for the last run. If each seen PE is interpreted as a Bernoulli trial, with "success" being the "deletion of at least one k-sum from the current g_i", then the number of successes occurred is at most A_n-m, where m is the number of k-sums of f also contained on g_0 (m will be equal to the number of k-sums of f, only if n=k), or, at most A_n. Hence

$$P\left[\sum_{g(v)\neq 1} D^+(v) \ge h_1^{-1}\right] \le$$

$$\le b^*\left(\le A_n;\ L(h_1,h_2,A_n),\ h_1^{-1}\right) \le h_2^{-1},$$

which proves the result.

(b) Suppose that

$$\sum_{g(v)\neq 0} D^-(v) \ge h_3^{-1}$$

holds, after execution of the last run. If each NE, v, seen during the last run, is inteprepted as a Bernoulli trial, with "success" being "v makes g 1/*", then

$$\left\lceil \frac{\ln h_4^{-1}}{\ln\left(1-h_3^{-1}\right)} \right\rceil$$

independent and equiprobable Bernoulli trials have been performed, each of them having probability of success at least h_3^{-1}, and no success has occurred (since this is the last run). Hence,

$$P\left[\sum_{g(v)\neq 0} D^-(v) \geq h_3^{-1}\right] = b\left(0; \left\lceil \frac{\ln h_4^{-1}}{\ln\left(1-h_3^{-1}\right)} \right\rceil, h^{-1}\right)$$

for some $h^{-1} \geq h_3^{-1}$.

According to (2.7), the last inequality implies

$$P\left[\sum_{g(v)\neq 0} D^-(v) \geq h_3^{-1}\right] \leq \left(1-h_3^{-1}\right)^{\left\lceil \frac{\ln_4^{-1}}{\ln\left(1-h_3^{-1}\right)} \right\rceil},$$

which, according to elementary properties of the logarithmic and exponential functions, can be rewritten as

$$P\left[\sum_{g(v)\neq 0} D^-(v) \geq h_3^{-1}\right] \leq h_4^{-1}.$$

That completes the proof.

(c) According to Lemma 6.1, g is in n-CNF, for some $n \leq k$. The proof that g is maximal is the same as that of Theorem 4.2(c). ■

Before explaining the significance of Theorem 6.1, let's make a technical remark concerning the number of NEs required in each run of Algorithm 6.1.

According to the interpretation of each seen NE as a Bernoulli trial, stated in the proof of Theorem 6.1(b), at least X NEs should be seen, where X satisfies the condition

$$b(0;X,h^{-1}) \le h_4^{-1} \quad \text{for some } h^{-1} \ge h_3^{-1}, \tag{6.1}$$

which, according to Theorem 2.1, implies that

$$X = L(h_3,h_4,0) = \left\lceil \frac{\ln h_4}{\ln\left(\frac{eh_3}{eh_3+1-e}\right)} \right\rceil \tag{6.2}$$

NEs are sufficient. However, an other way of finding an X satisfying (6.1), is to use (2.7), and to rewrite (6.1) as

$$\left(1-h_3^{-1}\right)^X \le h_4^{-1}, \tag{6.3}$$

which implies that

$$X = \left\lceil \frac{\ln h_4^{-1}}{\ln\left(1-h_3^{-1}\right)} \right\rceil \tag{6.4}$$

NEs are sufficient. It can be easily verified that the X of (6.4), which is the number used in Algorithm 6.1, is better (smaller) than the X of (6.2).

This result can be easily understood if we recall that the effort put to establish Theorem 2.1 was justified by the fact that there exists no closed formula for $b(\le k;X,h^{-1})$ for $k=0,1,\ldots,X$ (see discussion after Definition 2.4). This fact forced us to derive an **upper bound** of $b(\le k;X,h^{-1})$ (see Lemma 2.4) in order to establish Theorem 2.1, losing unavoidably some ground. However, when $k=0$, there is no problem in computing the **exact** value of $b(\le 0;X,h^{-1}) = b(0;X,h^{-1})$. Therefore, the X of (6.4), obtained after computing the exact value of $b(0;X,h^{-1})$ in (6.3), is better than the X of (6.2), obtained by using the bound of Theorem 2.1. ■

Let's now look back at the information provided by Theorem 6.1. According to parts (a) and (b) of the theorem, Algorithm 6.1 performs strong two-sided-error learning, and therefore, the Venn diagram of f and g of the algorithm is the one shown in Figure 3.6. All seen PEs, and possibly some non-seen ones (consider the example of Figure 4.3(e), or the case in which less than k runs are executed), are correctly included in g. Also, all seen NEs, possibly again with some non-seen ones (consider the case in which k runs are executed), are correctly excluded from g. The falsely non-included (non-excluded) PEs (NEs) are bounded, as usually, by h_1 and h_2 (h_3 and h_4).

Part (c) of the theorem implies that the output of Algorithm 6.1 can determine fast (in $\Theta(t^{k+1})$ time, in the worst case) whether any vector is a PE of a NE of the learned concept. ■

The study of the time complexity of Algorithm 6.1 completes the algorithm's analysis.

<u>Theorem 6.2.</u> If f can be written in k-CNF, then the worst-case time complexity of Algorithm 6.1 is

$$T(t,h_1,h_2,h_3,h_4) = \Theta\left(t^{n+1}\left(h_1t^n+h_1\ln h_2+h_3\ln h_4\right)\right)$$

for some $n \leq k$.

Proof: First it will be shown that

$$\frac{1}{\ln\left(\frac{h_3}{h_3-1}\right)} = \Theta(h_3). \tag{6.5}$$

Indeed, according to (2.2),

$$\frac{1}{\ln\left(\frac{h_3}{h_3-1}\right)} \leq h_3 \tag{6.6}$$

holds. Consider now the function

$$f : [1, \infty) \dashrightarrow R \quad \text{with} \quad f(h_3) = h_3\ln\left(\frac{h_3}{h_3-1}\right). \tag{6.7}$$

It can be verified that

$$f'(h_3) = \ln\left(\frac{h_3}{h_3-1}\right) - \frac{1}{h_3-1},$$

which, according to (2.2), implies that $f'(h_3) \leq 0$, or equivalently,

$$f(h_3) : \text{decreasing in } [1, \infty),$$

which implies that

$$f(h_3) \leq f(2) = 1.39 \quad \forall h_3 \geq 2,$$

or, according to (6.7),

$$\frac{1}{\ln\left(\frac{h_3}{h_3-1}\right)} \geq \frac{1}{1.39}\, h_3 \quad \forall h_3 \geq 2. \tag{6.8}$$

The last inequality, together with (6.6), prove (6.5). According now to Theorem 4.3, execution of line 2 of Algorithm 6.1, in the j-th run, takes $\Theta(h_1t^{j+1}(t^j+\ln h_2))$ time in the worst case. Theorem 3.1(d), on the other hand, implies that each "$s_i(v) \neq 0$" test in line 4 of the "Conflict-max-g-NE" function, takes at most $\theta(t)$ time. Since this test is performed at most $A_j = \theta(t^j)$ times in the j-th run, the execution of line 3 of Algorithm 6.1 in the j-th run takes at most

$$\theta\left(t^{j+1}\frac{\ln h_4}{\ln\left(\frac{h_3}{h_3-1}\right)}\right)$$

time, or, according to the established (6.5), at most

$$\theta(t^{j+1}h_3\ln h_4)$$

time in the j-th run. According now to Lemma 6.1, the result follows. ■

Although we could set k, instead of n, in the time complexity expression of the theorem above, we preferred not to do so, because we wanted to emphasize one of the main advantages of our algorithm over the algorithms of Chapter 4, namely the possibility that Algorithm 6.1 may terminate in fewer than k runs.

Consider, for example, the task of learning the concept of Figure 6.4(a) when v_1 is a very rare NE. Clearly, Algorithm 4.1 cannot be applied if the fact that k=4 is not known; however, this knowledge is not required by our Algorithm 6.1. Moreover, even if Algorithm 4.1 has this knowledge and Algorithm 6.1 doesn't, Algorithm 4.1 will probably be more expensive than Algorithm 6.1, since it will

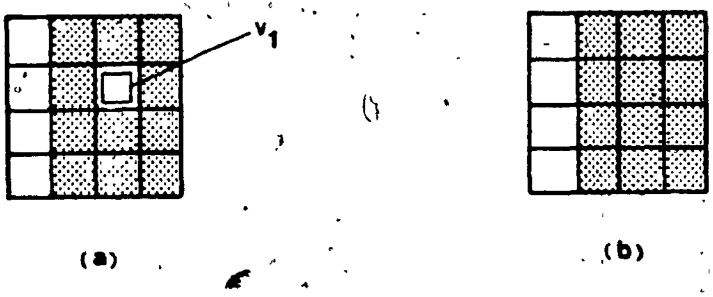


Figure 6.4. Algorithm 6.1 may be faster than Algorithm 4.1.

try to learn a 4-CNF, whereas Algorithm 6.1 after probably failing to learn a 1-CNF, it will probably succeed in learning the 2-CNF of Figure 6.4(b).

It has to be stated though, that if v_1 is a frequent NE, then Algorithm 6.1 will probably be more expensive than Algorithm 4.1, since it will probably learn a 4-CNF, after having unsuccessfully tried to learn a 1-CNF, a 2-CNF, and a 3-CNF. However, according to Theorems 4.3 and 6.2, the worst-case running time of Algorithm 6.1 will be asymptotically equal to that of Algorithm 4.1. ■

6.4. DNF Learning - Case I

Since no information about the form of the concept to be learned is available, it is equally reasonable to try to learn a DNF approximation of the concept, instead of a CNF one. The algorithm performing this task will be the dual of Algorithm 6.1. The way it works, on the same concept of Figure 6.3(a), is shown in Figure 6.5. For convenience, the 3-DNF concept to be learned has been redrawn in Figure 6.5(a). If the dual of Algorithm 6.1 (i.e. Algorithm 6.2) is applied, and if all PEs and NEs are seen, then the concepts learned after the 1st, 2nd, and 3rd run (i.e. after a 1-CNF, a 2-CNF, and a 3-CNF have been tried to be learned) are shown in Figures 6.5(b), 6.5(c), amd 6.5(d), respectively. ■

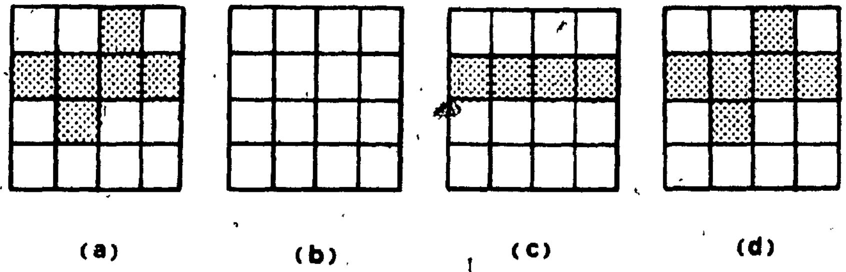


Figure 6.5. Attempts for k-DNF learning without knowledge of k.

The description of the function testing a n-DNF, learned from NEs, against PEs, is given first. It is followed by the algorithm's description and its theorems.

The duals of all comments and remarks made for Algorithm 6.1, hold for Algorithm 6.2 as well, and for this reason, they have been omitted here.

<u>Function</u> conflict-max-g-PE : boolean

Task : Return the value TRUE, if at least one of the seen error-free PEs of f makes g 0/*; otherwise, return FALSE.

Assumptions: g is in maximal n-DNF, for some $1 \le n \le t$.

Parameters : h_1, h_2; set up in calling routine.

```
repeat $\left\lceil \frac{\ln h_{\text{[illegible]}}}{\ln\left(\frac{h_1}{h_1-1}\right)} \right\rceil$ times
  begin
    v <-- EXAMPLC$^+$
    if  $p_i(v) \neq 1 \quad \forall p_i \in g$        [if $g(v) \neq 1$]
      then
        begin
          conflict-max-g-PE <-- TRUE
          STOP
        end
  end
conflict-max-g-PE <-- FALSE
```

<u>Algorithm 6.2.</u>

Task : Learn f from error-free PEs and NEs.

Assumptions: $x_1,\ldots,x_t$ are sufficient to express f.

Parameters : $h_1, h_2, h_3, h_4 \geq 1$.

```
1. $n \leftarrow 1$                             [set n to 1]
2. call Algorithm 4.2 with $k \leftarrow n$     [learn n-DNF from NEs]
3. if conflict-max-g-PE                         [if over-specialized]
4.   then
5.     begin
6.       $n \leftarrow n+1$                     [increment n]
7.       go to 2                                [and try again]
8.     end
```

■

Lemma 6.2. If f can be written in k-DNF, then the maximum number of runs of Algorithm 6.2 is k.

Proof: The dual of that of Lemma 6.1. ■

Theorem 6.3. The output g of Algorithm 6.2 has the properties:

(a) $P\left[\sum_{g(v)\neq 1} D^{+}(v) \geq h_1^{-1}\right] \leq h_2^{-1}$.

(b) $P\left[\sum_{g(v)\neq 0} D^{-}(v) \geq h_3^{-1}\right] \leq h_4^{-1}$.

(c) If f can be written in k-DNF, then g is in maximal n-DNF, for some $n \leq k$.

Proof: The proofs of (a), (b), and (c), are the duals of those of (b), (a), and (c), of Theorem 6.1. ■

<u>Theorem 6.4.</u> If f can be written in k-DNF, then the worst-case time complexity of Algorithm 6.2 is

$$T(t,h_1,h_2,h_3,h_4) = \Theta\left(t^{n+1}\left(h_3 t^n + h_1 \ln h_2 + h_3 \ln h_4\right)\right)$$

for some $n \leq k$.

Proof: As in Theorem 6.2. ■

In contrast to Algorithms 4.1 and 4.2, a comparison of Algorithms 6.1 and 6.2 does make sense. Suppose that the concept to be learned can be written in k_1-CNF and in k_2-DNF, and that neither of k_1, k_2 is known. Then, both Algorithms 6.1 and 6.2 are applicable. Unfortunately though, no comparison of their execution time can be made.

The reason for that is not (as it might be superficially thought) that it may not be known whether $k_1<k_2$ or $k_2<k_1$. Even if it is known, for example, that $k_2<k_1$, it cannot be decided that Algorithm 6.2 is faster than Algorithm 6.1. That is because Algorithm 6.2 may terminate after k_2 runs, but Algorithm 6.1 may terminate after only k_3 runs, for some $k_3<k_2<k_1$. Such an example was given in Figure 6.4.

From this brief discussion, it should be clear that it is the D^+ and D^- which determine which algorithm is faster. Since no information about these distributions is available, no prediction about the best algorithm (for a particular situation) can be made. ■

6.5. CNF/DNF Learning - Case I

Since it cannot be determined whether Algorithm 6.1, learning a CNF expression, or Algorithm 6.2, learning a DNF expression, is more appropriate to be applied in a particular situation, a hybrid algorithm, learning either a CNF or a DNF, has been designed. This algorithm tries to learn succesively a 1-CNF, 1-DNF, 2-CNF, 2-DNF, etc., until it succeeds. ■

The algorithm is precisely stated as follows:

Algorithm 6.3.

Task : Learn f from error-free PEs and NEs.

Assumptions: $x_1,\ldots,x_t$ are sufficient to express f.

Parameters : $h_1, h_2, h_3, h_4 > 1$.

```
n <-- 1                                [set n to 1]
call Algorithm 4.1 with k<--n          [learn n-CNF]
if conflict-max-g-NE                   [if over-general]
  then
    begin
      call Algorithm 4.2 with k<--n    [learn n-DNF]
      if conflict-max-g-PE             [if over-specific]
```

```
8.          then
9.            begin
10.             n <-- n+1                [increment n]
11.             go to 2                  [and try again]
12.           end
13.       end                                          ■
```

According to Definition 6.1, each execution of lines 2 to 13 is a run. Clearly, the i-th run of this algorithm takes twice as much time as the i-th run of either Algorithm 6.1 or 6.2, for i=1,2,3,...,t. ■

Lemma 6.3. If f can be written in k_1-CNF and in k_2-DNF, then the maximum number of runs of Algorithm 6.3 is $\min[k_1, k_2]$.

Proof: Immediate from Lemmas 6.1 and 6.2. ■

The two theorems, concerning the output and the time complexity of Algorithm 6.3, are stated next.

Theorem 6.5. The output g of Algorithm 6.3 has the properties:

(a) $P\left[\sum_{g(v)\neq 1} D^+(v) \geq h_1^{-1}\right] \leq h_2^{-1}$.

(b) $P\left[\sum_{g(v)\neq 0} D^-(v) \geq h_3^{-1}\right] \leq h_4^{-1}$.

(c) If f can be written in k_1-CNF and in k_2-DNF, then g is either in maximal n-CNF or in maximal n-DNF, for some $n \leq \min\{k_1, k_2\}$.

Proof: Immediate from Theorems 6.1, 6.3, and Lemma 6.3. ■

It has to be made clear here, that according to what was said at the end of Section 6.4, the statement "if $k_1 \leq k_2$ then g is in maximal n-CNF for some $n \leq k_1$, and if $k_1 > k_2$ then g is in maximal n-DNF for some $n \leq k_2$", with which one may attempt to replace part (c) of the last theorem, is wrong. ■

Theorem 6.6. If f can be written in k_1-CNF and in k_2-DNF, then the worst-case time complexity of Algorithm 6.3 is

$$T(t, h_1, h_2, h_3, h_4) = \Theta\left(t^{n+1}\left((h_1+h_3)t^n + h_1 \ln h_2 + h_3 \ln h_4\right)\right)$$

for some $n \leq \min\{k_1, k_2\}$

Proof: During the j-th run, execution of line 2 of Algorithm 6.3 takes at most $\Theta(h_1 t^{j+1}(t^j + \ln h_2))$ time, execution of line 3 takes at most $\Theta(t^{j+1} h_3 \ln h_4)$ time, execution of line 6 takes at most $\Theta(h_3 t^{j+1}(t^j + \ln h_4))$ time, and execution of line 7 takes at most $\Theta(t^{j+1} h_1 \ln h_2)$ time. According now to Lemma 6.3, the result follows. ■

6.6. CNF Learning - Case II

Let's go back for a moment to the example of Figure

6.4. As it has been said, if the stand-alone NE v_1 is very rare, then Algorithm 6.1 will probably require only 2 runs, since probably v_1 will not be seen as a NE, conflicting the 2-CNF concept of Figure 6.4(a) learned in the 2nd run.

Obviously, for this particular example, the probability that the algorithm will terminate after the 2nd run, can be increased, by starting a new run only when the frequency of the observed conflicts, between the so far learned concept and the NEs seen, is higher than a certain threshold d, rather than right after the first observed conflict. In general, the introduction of such a threshold d (i.e. the tolerance by the user of some discrepancies between the NEs and the learned concept) is hoped to reduce the number of runs required, and therefore the time complexity of algorithm as well. ■

The function performing this new kind of test against NEs, is given next.

<u>Function</u> d-conflict-max-g-NE : boolean

Task : Return the value TRUE if the frequency of the seen error-free NEs of f, making g 1/*, is greater than d; otherwise, return FALSE.

Assumptions: g is in maximal n-CNF, for some $1 \le n \le t$.

Parameters : h_3, h_4, and d; set up in calling routine.

```
1. COUNT <-- 0
2. repeat  $L = \left\lceil \frac{d+\ln h_4}{\ln\left(\frac{eh_3}{eh_3+1-e}\right)-d} \right\rceil$  times
3.    begin
4.       v <-- EXAMPLE
5.       if $s_i(v)\neq 0$   $\forall s_i\in g$          [if $g(v)\neq 0$]
6.          then   COUNT <-- COUNT+1
7.    end
8. d-conflict-max-g-NE <-- $\left(\frac{\text{COUNT}}{L} > d\right)$   ■
```

Notice once more that the test of line 5 of the above function is equivalent to $g(v)\neq 0$, because g is maximal. ■

The new CNF learning algorithm, using the last function as a detector of over-generalization, is stated next.

Algorithm 6.4.

Task   : Learn f from error-free PEs and NEs.

Assumptions: $x_1,\ldots,x_t$ are sufficient to express f.

Parameters : h_1, h_2, h_3, $h_4 > 1$, and $d>0$.

```
1. n <-- 1                              [set n to 1]
2. call Algorithm 4.1 with k<--n        [learn n-CNF from PEs]
3. if d-conflict-max-g-NE               [if over-generalized]
4.    then
5.       begin
6.          n <-- n+1                   [increment n]
```

```
7.      go to 2                    [and try again]
8.      end                                          ■
```

The next theorem is very important, since it establishes a necessary and sufficient condition in order for Algorithm 6.4 to be applicable.

<u>Theorem 6.7.</u> Algorithm 6.4 is applicable if and only if

$$0 < d < \ln\left(\frac{eh_3}{eh_3+1-e}\right)$$

holds for the user-specified parameter d.

Proof: Immediate from the definition of L in line 2 of function "d-conflict-max-g-NE" ■

The following lemma will be used in the proofs of Theorems 6.8 and 6.9.

<u>Lemma 6.4.</u> If f can be written in k-CNF, then the maximum number of runs of Algorithm 6.4 is k.

Proof: Same as in Lemma 6.1. ■

<u>Theorem 6.8.</u> The output g of Algorithm 6.4 has the properties:

(a) $P\left[\sum_{g(v)\neq 1} D^+(v) \geq h_1^{-1}\right] \leq h_2^{-1}$.

(b) $P\left[\sum_{g(v)\neq 0} D^-(v) \geq h_3{}^{-1}\right] \leq h_4{}^{-1}$.

(c) If f can be written in k-CNF, then g is in maximal n-CNF, for some $n \leq k$.

Proof:

(a) Same as in Theorem 6.1(a).

(b) Suppose that

$$\sum_{g(v)\neq 0} D^-(v) \geq h_3{}^{-1}$$

holds, after execution of the last run. If each NE, v, seen during the last run, is interpreted as a Bernoulli trial, with "success" being "v makes (all n-sums of) g 1/*", then

$$L = \left\lceil \frac{d + \ln h_4}{\ln\left(\frac{eh_3}{eh_3+1-e}\right) - d} \right\rceil$$

independent and equiprobable Bernoulli trials have been performed, each having probability of success at least $h_3{}^{-1}$, and at most dL successes have occurred (since this is the last run). Hence

$$P\left[\sum_{g(v)\neq 0} D^-(v) \geq h_3{}^{-1}\right] = b^*(\leq dL;\ L,\ \geq h_3{}^{-1}).$$

But, according to Theorem 6.7, Theorem 2.5 implies that

$$b^*(\leq dL;\ L,\ \geq h_3{}^{-1}) \leq h_4{}^{-1},$$

and the result follows.

(c) Same as in Theorem 6.1(c). ■

Theorem 6.8 implies that Algorithm 6.4 performs strong two-sided-error learning, and that its output g can determine fast whether any vector is a PE or a NE of the learned concept. One difference between the output of Algorithm 6.4 and that of Algorithm 6.1 is that, because of the introduced threshold d in Algorithm 6.4, some seen NEs may not be excluded from g. That will happen if only a few conflicts occur in the last run, between the so far learned CNF and the seen NEs. Despite this fact, the confidence the user wants to have to the output of the algorithm can always be fully determined by h_1, h_2, h_3, and h_4, which affect the time complexity of the algorithm only (sub)linearly, as the following theorem shows.

<u>Theorem 6.9.</u> If f can be written in k-CNF, then the worst-case time complexity of Algorithm 6.4 is

$$T(t,h_1,h_2,h_4) = \Theta\left(t^{n+1}\left(h_1t^n+h_1\ln h_2+\frac{d+\ln h_4}{\ln\left(\frac{eh_3}{eh_3+1-e}\right)-d}\right)\right)$$

for some $n \leq k$.

Proof: The proof is similar to the one of Theorem 6.2, and it is briefly sketched here.

During the j-th run, execution of line 2 of the algorithm takes at most

$$\Theta\left(h_1t^{j+1}(t^j+\ln h_2)\right)$$

time, and execution of line 3 takes at most

$$\theta\left(t^{j+1}\,\frac{d+\ln h_4}{\ln\left(\frac{eh_3}{eh_3+1-e}\right)-d}\right)$$

time. The result now follows from Lemma 6.4. ■

6.7. Comparison of Case I and Case II of CNF Learning

According to what was stated in the beginning of the last section, Algorithm 6.4 was introduced with the hope that it will be at least as fast as Algorithm 6.1 in all cases (i.e. if the same PEs and NEs are presented to both algorithms). Let's see now whether our hope became a reality or not.

First of all, both Algorithms 6.1 and 6.4 require the same number of PEs per run, and moreover, if they are shown the same PEs, they process them in the same amount of time.

However, if

$$L_2 = \left\lceil \frac{d+\ln h_4}{\ln\left(\frac{eh_3}{eh_3+1-e}\right)-d} \right\rceil \tag{6.9}$$

is the number of NEs required in each run of Algorithm 6.4, and

$$L_1 = \left\lceil \frac{\ln h_4}{\ln\left(\frac{h_3}{h_3-1}\right)} \right\rceil \tag{6.10}$$

is the number of NEs required in each run of Algorithm 6.1, then it can be shown that

$$L_1 < L_2 \qquad (6.11)$$

Indeed, in order to prove that, it suffices to prove that

$$\frac{\ln h_4}{\ln\left(\frac{eh_3}{eh_3+1-e}\right)} > \frac{\ln h_4}{\ln\left(\frac{h_3}{h_3-1}\right)},$$

or, equivalently,

$$\frac{eh_3}{eh_3+1-e} < \frac{h_3}{h_3-1},$$

which can be easily verified that is true.

Because of (6.11), the comparison of the process time of the same NEs, by Algorithms 6.1 and 6.4, makes no sense. Let's agree therefore, that the comparison of the two algorithms will be made under the assumption that Algorithm 6.1 encounters a proper subset of the NEs encountered by Algorithm 6.4. In this case, (6.11) implies that each run of Algorithm 6.4 takes more time than the corresponding run of Algorithm 6.1.

What is then the advantage of applying Algorithm 6.4 in lieu of Algorithm 6.1? A plausible answer could be that Algorithm 6.4 requires fewer runs than Algorithm 6.1. But unfortunately, this is not always true.

Clearly, if Algorithm 6.1 terminates after n runs, Algorithm 6.4 may require more than that, since the additional NEs it is going to examine may increase the frequency of conflicts beyond the threshold d. On the other hand, if Algorithm 6.4 terminates after n runs, Algorithm 6.1 may not terminate, since it may encounter one (or more) NEs causing conflicts. So, **any of those two algorithms may require** more runs than the other, and therefore, **more execution time than the other.**

Let's make one additional (but still reasonable) assumption, to see whether Algorithm 6.4 can be proven to be faster than Algorithm 6.1. Suppose that the two algorithms examine L_1 and L_2 NEs having the same frequency of NEs causing conflicts with the same g. In this case, Algorithm 6.4 will not require more runs than Algorithm 6.1. However, this is not of much help, because that implies that Algorithm 6.4 may require the same number of runs as Algorithm 6.1. In such a case, according to what was said before, the overall execution time of Algorithm 6.4 will be higher than that of Algorithm 6.1. This case may arise, for example, if, when learning the concept of Figure 6.4(a), d is chosen much lower than $D^{-}(v)$. Clearly, **it is the lack of knowledge about D^{-}, which makes the choice of the appropriate d not feasible.** ■

Can any conclusion, concerning the "usually" fastest algorithm, be drawn? Clearly, no, as long as we remain faithful to our initial choice of making no assumptions about the examples' distributions D^+ and D^-. ■

What about the worst-case time complexity of Algorithms 6.1 and 6.4? Since each NE is processed in at most $\Theta(t)$ time, (6.11) implies that, in the worst case, each run of Algorithm 6.4 takes more time than the corresponding run of Algorithm 6.1. Therefore, in the worst case, Algorithm 6.4 requires more execution time than Algorithm 6.1. On the other hand though, Theorems 4.3 and 6.9 imply that asymptotically both algorithms have the same worst-case time complexity. ■

To summarize, due to the lack of information about D^-, the threshold d was left as a user-specified parameter, destroying our hope for the superiority of Algorithm 6.4 over Algorithm 6.1. Actually, for certain choices of d, Algorithm 6.4 is slower than Algorithm 6.1.

Despite the last fact, both algorithms were stated and analyzed in this chapter, since none of them is worse than any other. ■

6.8. DNF Learning - Case II

The dual of Algorithm 6.4, learning DNF expressions, is briefly presented here. It is preceded by the function testing a DNF concept, learned from NEs, against a number of PEs, for possible over-specialization. Finally, the theorems referring to the algorithm's output and time complexity are stated. ■

Function d-conflict-max-g-PE : boolean

Task : Return the value TRUE if the frequency of the seen error-free PEs of f, making g 0/*, is greater than d; otherwise, return FALSE.

Assumptions: g is in maximal n-DNF, for some $1 \le n \le t$.

Parameters : h_1, h_2, and d; set up in calling routine.

```
COUNT <-- 0
repeat   $L = \left\lceil \frac{d+\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)-d} \right\rceil$   times
  begin
    v <-- EXAMPLE$^+$
    if $p_i(v) \neq 1 \quad \forall p_i \in g$        [if $g(v) \neq 1$]
      then   COUNT <-- COUNT+1
  end
d-conflict-max-g-PE <-- $\left(\frac{COUNT}{L} > d\right)$
```

■

Algorithm 6.5.

Task : Learn f from error-free PEs and NEs.

Assumptions: $x_1, \ldots, x_t$ are sufficient to express f.

Parameters : $h_1, h_2, h_3, h_4 > 1$, and $d>0$.

```
1. n <-- 1                                [set n to 1]
2. call Algorithm 4.2 with k<--n          [learn n-DNF from NEs]
3. if d-conflict-max-g-PE                 [if over-specialized]
4.   then
5.     begin
6.       n <-- n+1                        [increment n]
7.       go to 2                          [and try again]
8.     end
```
■

Theorem 6.10. Algorithm 6.5 is applicable if and only if

$$0 < d < \ln\left(\frac{eh_1}{eh_1+1-e}\right)$$

holds for the user-specified parameter d.

Proof: Immediate from the definition of L in line 2 of the function "d-conflict-max-g-PE". ■

Lemma 6.5. If f can be written in k-DNF, then the maximum number of runs of Algorithm 6.5 is k.

Proof: The dual of that of Lemma 6.4. ■

Theorem 6.11. The output g of Algorithm 6.5 has the properties:

(a) $P\left[\sum_{g(v)\neq 1} D^+(v) \geq h_1^{-1}\right] \leq h_2^{-1}$.

(b) $P\left[\sum_{g(v)\neq 0} D^-(v) \geq h_3^{-1}\right] \leq h_4^{-1}$.

(c) If f can be written in k-DNF, then g is in maximal n-DNF, for some $n \le k$.

Proof: The proofs of (a), (b), and (c), are the duals of those of (b), (a), and (c), of Theorem 6.8. ■

The only difference between the output of Algorithm 6.5 and that of Algorithm 6.2 (or that of Algorithm 6.1) is that some **seen** PEs may not be included in g. That will happen if only a few conflicts occur in the last run.

Theorem 6.12. If f can be written in k-DNF, then the worst-case time complexity of Algorithm 6.4 is

$$T(t,h_2,h_3,h_4) = \Theta\left(t^{n+1}\left(h_3t^n+h_3\ln h_4+\frac{d+\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)-d}\right)\right)$$

for some $n \le k$.

Proof: As in Theorem 6.9. ■

6.9. CNF/DNF Learning - Case II

In exactly the same way Algorithm 6.3 was a compromise between Algorithms 6.1 and 6.2, Algorithm 6.6, presented in this section, is a compromise between Algorithms 6.4 and 6.5. ■

Algorithm 6.6.

Task : Learn f from error-free PEs and NEs.

Assumptions: $x_1, \ldots, x_t$ are sufficient to express f.

Parameters : $h_1, h_2, h_3, h_4 > 1$, and $d>0$.

```
 1. n <-- 1                                [set n to 1]
 2. call Algorithm 4.1 with k<--n          [learn n-CNF]
 3. if d-conflict-max-g-NE                 [if over-general]
 4.   then
 5.     begin
 6.       call Algorithm 4.2 with k<--n    [learn n-DNF]
 7.       if d-conflict-max-g-PE           [if over-specific]
 8.         then
 9.           begin
10.             n <-- n+1                  [increment n]
11.             go to 2                    [and try again]
12.           end
13.     end
```

Theorem 6.13. Algorithm 6.6 is applicable if and only if

$$0 < d < \min\left\{\ln\left(\frac{eh_1}{eh_1+1-e}\right),\ \ln\left(\frac{eh_3}{eh_3+1-e}\right)\right\}$$

holds for the user-specified parameter d.

Proof: Immediate from the definitions of L in line 2 of each of the functions "d-conflict-max-g-NE" and "d-conflict-max-g-PE". ■

<u>Lemma 6.6.</u> If f can be written in k_1-CNF and in k_2-DNF, then the maximum number of runs of Algorithm 6.6 is $\min\{k_1,k_2\}$.

Proof: Immediate from Lemmas 6.4 and 6.5. ■

<u>Theorem 6.14.</u> The output g of Algorithm 6.6 has the properties:

(a) $P\left[\sum_{g(v)\neq 1} D^+(v) \geq h_1^{-1}\right] \leq h_2^{-1}$.

(b) $P\left[\sum_{g(v)\neq 0} D^-(v) \geq h_3^{-1}\right] \leq h_4^{-1}$.

(c) If f can be written in k_1-CNF and in k_2-DNF, then g is either in maximal n-CNF or in maximal n-DNF, for some $n \leq \min\{k_1,k_2\}$.

Proof: Immediate from Theorems 6.8, 6.11, and Lemma 6.6. ■

<u>Theorem 6.15.</u> If f can be written in k_1-CNF and in k_2-DNF, then the worst-case time complexity of Algorithm 6.6 is

$$T(t,\, h_2,\, h_4) =$$

$$= \Theta\left(t^{n+1}\left((h_1+h_3)t^n + h_1 \ln h_2 + h_3 \ln h_4 + \frac{d+\ln h_4}{\ln\left(\frac{eh_3}{eh_3+1-e}\right)-d} + \frac{d+\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)-d}\right)\right)$$

for some $n \leq \min\{k_1,k_2\}$.

Proof: During the j-th run, execution of line 2 of Algorithm 6.6 takes at most

$$\theta\left(h_1 t^{j+1}(t^j+\ln h_2)\right)$$

time, execution of line 3 takes at most

$$\theta\left(t^{j+1}\frac{d+\ln h_4}{\ln\left(\frac{eh_3}{eh_3+1-e}\right)-d}\right)$$

time, execution of line 6 takes at most

$$\theta\left(h_3 t^{j+1}(t^j+\ln h_4)\right)$$

time, and execution of line 7 takes at most

$$\theta\left(t^{j+1}\frac{d+\ln h_2}{\ln\left(\frac{eh_1}{eh_1+1-e}\right)-d}\right)$$

time. According now to Lemma 6.6, the result follows. ■

6.10. Summary

Six new algorithms for concept learning from error-free PEs and NEs were presented in this chapter. The main differences and similarities between those algorithms and the ones of Chapter 4 are summarized below.

(i) No information about the form of the concept to be learned is required by the algorithms of Chapter 6. In contrast to that, the knowledge of the maximum number of literals per conjunct/disjunct is required by the algorithms of Chapter 4.

(ii) Both PEs and NEs are required by the algorithms of Chapter 6. The examples have to be presented in groups of PEs and NEs of particular size, but the order of the PEs/NEs inside each group in immaterial. In contrast to that, only PEs or only NEs are required by the algorithms of Chapter 4.

(iii) The actual running time of Algorithms 6.1 and 6.4 (6.2 and 6.5) can be either lower or higher than that of Algorithm 4.1 (4.2). Their worst-case running time is non-asymptotically higher, but asymptotically (if considered as a function of t) is equal to that of Algorithm 4.1 (4.2).

(iv) The algorithms of Chapter 6 perform two-sided-error learning, whereas those of Chapter 4 perform one-sided-error learning. For both cases, the amount of discrepancy between the true concept (the one from which the PEs and NEs are drawn) and the learned one, is within a user-defined confidence range, (sub)linearly affecting the algorithm's running time.

(v) The outputs of the algorithms of Chapters 4 and 6 can determine fast (in at most $\Theta(t^{k+1})$ time) whether any vector is a PE or a NE.

The algorithms of this chapter greatly improve those of Chapter 4, because **they approximate a concept without knowing anything about its representability** (except for the number of variables involved), and according to what was said in (ii), (iii), and (iv) above, **the price which has to be paid for that is not high at all.**

What is a bit annoying though, is the fact that it cannot be determined which of those algorithms is the best/worst, or which one is more/less appropriate to be applied to a particular situation. That depends on the examples' distributions D^+ and D^-, about which no information is available, and on the choice of d, which, because of the lack of information about D^+ and D^-, is left as a user-specified parameter. ■

Evidently, the next topic to be examined, is that of k-concept learning in a noisy environment, always in the absence of knowledge of k. Chapter 7 examines the simplest case, in which only one of the example generators is noisy.■

CHAPTER 7

k-CONCEPT LEARNING; UNKNOWN K; ONE ERROR-FREE AND ONE NOISY EXAMPLE GENERATOR

7.1. Introduction

The k-concept learning problem, when k is unknown, was first considered in the last chapter. Having studied this problem in an error-free environment, we are now ready to study the same problem in a noisy environment.

The case in which only one of the example generators is noisy, is examined in this chapter; the more complex case in which both example generators are noisy, is examined in the next chapter. As was also the case with last chapter's problem, the problem addressed in this chapter has also not been considered elsewhere. ■

The first algorithm of this chapter is stated in Section 7.2. It learns a CNF concept from error-free PEs and noisy NEs. A modified version of this algorithm is presented in Section 7.4. The duals of those two algorithms, learning a DNF concept from noisy PEs and error-free NEs, are stated in Sections 7.3 and 7.5. Finally, the reason for which different approaches to this chapter's problem have not been further investigated, is explained in Section 7.6. As usual, the chapter closes with a summary in Section 7.7. ■

7.2. CNF Learning from Noisy NEs - Case I

The problem of k-CNF learning from error-free PEs and noisy NEs, when k is not known, will be studied in this section.

The main idea used in the design of the algorithms of Chapter 6, will be used here, too. First, try to learn a 1-CNF from PEs, by applying Algorithm 4.1 and by testing its output against NEs; if significant over-generalization is detected, try to learn a 2-CNF in the same way; in case of a new failure, try to learn a 3-CNF, etc. Terminate the process when no significant over-generalization is detected.

One important difference between this algorithm and the algorithms of Chapter 6 is that, because of the noise of the NEs generator, even after k runs, some conflicts may occur, when the learned k-CNF is tested against the NEs, and therefore, more than k runs may be required. In this section, no solution is given to this problem; the algorithm is left to run until either a t-CNF is learned, or only a few conflicts are detected. ■

For clarity, the description of the fuction testing for over-generalization, precedes the presentation of the algorithm.

<u>Function</u> d-conflict-max-g-r-NE-I : boolean

Task : Return the value TRUE, if the frequency of the seen noisy NEs of f, making g 1/*, is greater than d; otherwise return FALSE.

Assumptions: g is in maximal n-CNF, for some $1 \leq n \leq t$.

Parameters : h_3, h_4, and d; set up in calling routine.

```
COUNT <-- 0
repeat   $L = \left\lceil \frac{d + \ln h_4}{\ln\left(\frac{eh_3(1-r)^{-1}}{eh_3(1-r)^{-1}+1-e}\right) - d} \right\rceil$   times
  begin
    v <-- r-EXAMPLE$^-$
    if $s_i(v) \neq 0 \quad \forall s_i \in g$        [if $g(v) \neq 0$]
      then COUNT <-- COUNT+1
  end
d-conflict-max-g-r-NE-I <-- $\left(\frac{COUNT}{L} > d\right)$
```
■

<u>Algorithm 7.1.</u>

Task : Learn f from error-free PEs and noisy NEs.

Assumptions: (a) $x_1,\ldots,x_t$ are sufficient to express f.

(b) Error rate r of NEs generator is known.

Parameters : h_1, h_2, h_3, h_4 >-1, and d > 0.

```
1.  n <-- 1                               [set n to 1]
2.  call Algorithm 4.1 with k<--n         [learn n-CNF g from PEs]
3.  if n<t                                [if g not least general]
4.    then
5.      if d-conflict-max-g-r-NE-I        [if g is over-general]
6.        then
7.          begin
8.            n <-- n+1                   [increment n]
9.            go to 2                     [and try again]
10.         end
```
■

Before stating the theorem describing the output of Algorithm 7.1, the conditions under which the algorithm is applicable, are given.

<u>Theorem 7.1.</u> Algorithm 7.1 is applicable if and only if

$$0 < d < \ln\left(\frac{eh_3(1-r)^{-1}}{eh_3(1-r)^{-1}+1-e}\right)$$

holds for the user-specified parameter d.

Proof: Immediate from the definition of L in line 2 of the "d-conflict-max-g-r-NE-I" function. ■

Theorem 7.2. The output g of Algorithm 7.1 has the properties:

(a) $P\left[\sum_{g(v)\neq 1} D^{+}(v) \geq h_1^{-1}\right] \leq h_2^{-1}$.

(b) $P\left[\sum_{g(v)\neq 0} D^{-}(v) \geq h_3^{-1}\right] \leq h_4^{-1}$.

(c) g is in maximal n-CNF, for some $n \leq t$.

Proof:

(a) Since the final output g of Algorithm 7.1 is nothing but the g formed during execution of Algorithm 4.1, in the last run, i.e. g is not altered by the "d-conflict-max-g-r-NE-I" function, the proof is the same as that of Theorem 6.1(a).

(b) (i) Suppose that t runs are executed (i.e. the t-CNF learned is not tested against NEs).

According the Theorem 4.2,

$$\sum_{g(v)\neq 0} D^{-}(v) = 0$$

holds, which implies

$$P\left[\sum_{g(v)\neq 0} D^{-}(v) \geq h_3^{-1}\right] = 0,$$

and the result follows.

(ii) Suppose that less than t runs are executed (i.e. the k-CNF learned is successfully tested against NEs).

Let V be the set of all NEs making the final output g of Algorithm 7.1 1/*. Clearly, it suffices to prove that

$$P\left[\sum_{v\in V} D^{-}(v) \geq h_3^{-1}\right] \leq h_4^{-1}. \qquad (7.1)$$

If each output, v, of r-EXAMPLE$^-$, seen during the last run, is interpreted as a Bernoulli trial, with "success" being "v makes g 1/*", then

$$L = \left\lceil \frac{d + \ln h_4}{\ln\left(\frac{eh_3(1-r)^{-1}}{eh_3(1-r)^{-1}+1-e}\right) - d} \right\rceil \qquad (7.2)$$

independent and equiprobable Bernoulli trials have been performed, and at most dL successes have occurred (since this is the last run). If p (p') denotes the probability that r-EXAMPLE$^-$ will change a vector $v\in V$ ($v\notin V$) to a vector $v'\notin V$ ($v'\in V$), then the probability of success of each Bernoulli trial is

$$P[\text{success}] = P[v : \text{inside } V]*(1-p) + P[v : \text{outside } V]*p' \qquad (7.3)$$

and therefore, since $r\geq p$,

$$P[\text{success}] \geq \left(\sum_{v\in V} D^{-}(v)\right)(1-r).$$

Clearly now, the event

"$\sum_{v\in V} D^{-}(v) \geq h_3^{-1}$ holds after the last run"

is equivalent to

"the probability of success of each of the L trials (in the last run) is at least $h_3^{-1}(1-r)$, and at most dL successes have occurred".

Therefore,

$$P\left[\sum_{v\in V} D^{-}(v) \geq h_3^{-1}\right] = b^*\left(\leq dL; L, \geq h_3^{-1}(1-r)\right) \quad (7.4)$$

holds. But, according to Theorem 7.1, Theorem 2.5 implies

$$b^*\left(\leq dL;\ L,\ \geq h_3^{-1}(1-r)\right) \leq h_4^{-1},$$

which proves (7.1), and therefore the result, too.

(c) Same as that of Theorem 4.2(c), if we set k<--n. ■

Parts (a) and (b) of this theorem imply that Algorithm 7.1 performs strong two-sided-error learning. Part (c) also of the same theorem implies that if n is the number of runs executed by Algorithm 7.1, then it takes $\Theta(t^{n+1})$ time in the worst case to determine whether a vector is a PE or a NE of the learned concept. ■

Before studying the asymptotic worst-case time complexity of Algorithm 7.1, it would be useful to point out that **the number of NEs required per run is high** enough to assure that Theorem 7.2 holds even in the worst case. This will arise if all NEs conflicting with g (making g 1/*) are changed by the noise to NEs/PEs not conflicting with g (making g 0), and all NEs not conflicting with g, are either unchanged, or changed to other NEs/PEs not conflicting with g, too. To say it in other words, the worst case will arise if the noise is trying to hide, as much as possible, any information leading to the detection of an over-generalization. ■

The above discussion also explains the reason for which an increase of the error rate of the NEs generator, necessitates an increase of the number of NEs required per run (see definition of L in the "d-conflict-max-g-r-NE-I" function).

When r increases, the probability that a NE causing a conflict will be error-free (will not cause a conflict), decreases. If the algorithm is to produce an equally reliable output, it has to be fed with more NEs, so that it will catch a sufficient number of conflicts, detecting in this way the over-generalization occurred, and taking appropriate action.

But what about the NEs not conflicting with g? Doesn't the increased r imply that there is a increased probability that a NE not conflicting with g will be changed to a NE conflicting with g, and therefore, doesn't it imply that the algorithm can afford seeing fewer NEs? The answer to this question is "not necessarily"; the probability that a NE not conflicting with g will be noisy, is not related to the probability that a NE not conflicting with g will be changed to a NE conflicting with g. In the most unfavorable case, all NEs not conflicting with g will be changed to NEs also not conflicting with g. Therefore, no increase in the number of conflicts is guaranteed, which implies that no decrease of the number of NEs can be afforded by our algorithm. ■

<u>Theorem 7.3.</u> The worst-case time complexity of Algorithm 7.1 is

$$T(t,h_1,h_2,h_4) = \Theta\left(t^{n+1}\left(h_1t^n+h_1\ln h_2+\frac{d+\ln h_4}{\ln\left(\frac{eh_3(1-r)^{-1}}{eh_3(1-r)^{-1}+1-e}\right)-d}\right)\right),$$

where n is the number of runs; n can grow up to t.

Proof: According to Theorem 4.3, during the j-th run of Algorithm 7.1, execution of line 2 takes

$$\Theta\left(h_1t^{j+1}(t^j+\ln h_2)\right)$$

time in the worst case. On the other hand, execution of line 5 of the "d-conflict-max-g-r-NE-I" function takes at most $\Theta(t^{j+1})$ time, and therefore execution of line 5 of Algorithm 7.1 takes

$$\Theta\left(t^{j+1}\frac{d+\ln h_4}{\ln\left(\frac{eh_3(1-r)^{-1}}{eh_3(1-r)^{-1}+1-e}\right)-d}\right)$$

time in the worst case. The result now follows immediately. ■

As was also the case with the threshold d introduced in Chapter 6 (see Section 6.6), the d of Algorithm 7.1 is left as a user-specified parameter, simply because its optimal value is not known; when d increases, the execution time of each run increases (by a known amount, specified by L), whereas the number of runs either remains the same, or decreases (by an unknown amount, depending on D^-).

However, what is more annoying here, is the fact that Algorithm 7.1, approximating a k-CNF concept, ~~may~~ not terminate even after k runs. Although after k runs, g is more specific than f, the possibility that the errors made by the NEs generator will result in a frequency of conflicts greater than d, and therefore in the initiation of a new run, cannot be eliminated. ■

This is a serious problem, because it implies that the time complexity of Algorithm 7.1 can be exponential in t. It is therefore worth investigating whether the probability that more than k runs will be executed, can be computed, or, at least, upper-bounded.

If the k-th run is not the last one, and if each NE, v, seen during this run, is interpreted as a Bernoulli trial, with "success" being "v makes g 0", then L independent and equiprobable Bernoulli trials have been performed, at most L-dL successes have occurred (since this is not the last run), and each of them has probability of success at least equal to the probability that no error will be made by r-EXAMPLE$^-$, hence at least 1-r. Therefore,

$$P[\text{more than k runs executed}]=b^*(\leq L-dL;L,\geq 1-r) \quad (7.5)$$

However, according to Lemma 2.1, $b(\leq k;\ L,\ \geq 1-r)$ is upper-bounded by $b(k;\ L,\ 1-r)$ only if $k\leq L(1-r)$ holds;

otherwise, the upper bound is 1, which is of no use. That implies that if we are to find an upper bound of (7.5),

$$L-dL \leq L(1-r)$$

should hold, or equivalently

$$d \geq r \tag{7.6}$$

should hold. However, the restriction of Theorem 7.1, imposed on d, does not make (7.6) always true; if, for example, r=.1 and h=10, then Theorem 7.1 implies that d<.058 is required, which makes (7.6) false.

The above discussion suggests that under our assumptions, the probability that more than k runs will be executed, can be arbitrarily high. ■

7.3. DNF Learning from Noisy PEs - Case I

If the PEs generator, rather than the NEs generator, is noisy, then the dual of Algorithm 7.1 can be applied to learn a DNF expression: it will try to learn a 1-DNF, 2-DNF, 3-DNF, etc., from the error-free NEs, until no significant over-specialization is detected, when the learned DNF is tested against the noisy PEs. ■

The function performing the test against the PEs is given first, followed by the algorithm's description, and its theorems.

<u>Function</u> d-conflict-max-g-r-PE-I : boolean

Task : Return the value TRUE, if the frequency of the seen noisy PEs of f, making g 0/*, is greater than d; otherwise return FALSE.

Assumptions: g is in maximal n-DNF, for some $1 \le n \le t$.

Parameters : h_1, h_2, and d; set up in calling routine.

```
COUNT <-- 0
repeat  $L = \left\lceil \dfrac{d + \ln h_2}{\ln\left(\dfrac{eh_1(1-r)^{-1}}{eh_1(1-r)^{-1}+1-e}\right) - d} \right\rceil$  times
    begin
        v <-- r-EXAMPLE$^+$
        if $p_i(v) \neq 1 \quad \forall p_i \in g$        [if $g(v) \neq 1$]
            then COUNT <-- COUNT+1
    end
d-conflict-max-g-r-PE-I <-- $\left(\dfrac{COUNT}{L} > d\right)$
```

<u>Algorithm 7.2.</u>

Task : Learn f from noisy PEs and error-free NEs.

Assumptions: (a) $x_1, \ldots, x_t$ are sufficient to express f.

(b) Error rate r of PEs generator is known.

Parameters : h_1, h_2, h_3, $h_4 > 1$, and $d > 0$.

```
n <-- 1                                [set n to 1]
call Algorithm 4.2 with k<--n          [learn n-DNF g from NEs]
if n<t                                 [if g not least specific]
```

```
4.  then
5.    if d-conflict-max-g-r-PE-I [if g is over-specific]
6.      then
7.        begin
8.          n <-- n+1              [increment n]
9.          go to 2                [and try again]
10.       end
```
■

<u>Theorem 7.4.</u> Algorithm 7.2 is applicable if and only if

$$0 < d < \ln\left(\frac{eh_1(1-r)^{-1}}{eh_1(1-r)^{-1}+1-e}\right)$$

holds for the user-specified parameter d.

Proof: Immediate from the definition of L in line 2 of the "d-conflict-max-g-r-PE-I" function. ■

<u>Theorem 7.5.</u> The output g of Algorithm 7.2 has the properties:

(a) $P\left[\sum_{g(v)\neq 1} D^{+}(v) \geq h_1^{-1}\right] \leq h_2^{-1}$.

(b) $P\left[\sum_{g(v)\neq 0} D^{-}(v) \geq h_3^{-1}\right] \leq h_4^{-1}$.

(c) g is in maximal n-DNF, for some $n \leq t$.

Proof: The proofs of (a), (b), and (c) are the duals of those of (b), (a) and (c) of Theorem 7.2. ■

<u>Theorem 7.6.</u> The worst-case time complexity of Algorithm

7.2 is

$$T(t,h_2,h_3,h_4) = \theta\left(t^{n+1}\left(h_3t^n+h_3\ln h_4+\frac{d+\ln h_2}{\ln\left(\frac{eh_1(1-r)^{-1}}{eh_1(1-r)^{-1}+1-e}\right)-d}\right)\right),$$

where n is the number of runs; n can grow up to t.

Proof: As in Theorem 7.3. ■

It is worth mentioning that, in contrast to our dilemma concerning the choice of one of Algorithms 6.1 and 6.2, no such problem arises here. If the PEs generator is noisy, only Algorithm 7.2 is applicable, whereas if the NEs generator is noisy, then only Algorithm 7.1 is applicable. ■

7.4. CNF Learning from Noisy NEs - Case II

The problem of CNF learning from error-free PEs and noisy NEs, already examined in Section 7.2, will be examined here again.

According to what was stated at the end of Section 7.2, if the error rare r of the NEs generator is unrestricted, and only the restriction of Theorem 7.1 is imposed on the user-specified threshold d, then (7.6) may be false, and therefore the probability that more than k runs will be executed can grow as high as 1. Clearly, this is undesirable. What we will try to do now, is to see whether this probability can be upper-bounded, by imposing some additional restriction on r and/or d, or by fixing their values.

Suppose that d is unknown. Our aim is to find the minimum number of NEs required per run, so that

$$P\left[\sum_{g(v)\neq 0} D^{-}(v) \geq h_3^{-1}\right] \leq h_4^{-1}, \qquad (7.7)$$

i.e. property (b) of Theorem 7.2, as well as

$$P[\text{more than k runs executed}] \leq h_5^{-1}, \qquad (7.8)$$

for some user-specified parameter $h_5>1$, will hold.

As shown in Theorem 7.2(b),

$$L = \left\lceil \frac{d + \ln h_4}{\ln\left(\frac{eh_3(1-r)^{-1}}{eh_3(1-r)^{-1}+1-e}\right) - d} \right\rceil \qquad (7.9)$$

is the minimum number of NEs making (7.7) true. Also, (7.5) and Theorem 2.5 imply that

$$L' = \left\lceil \frac{(1-d) - \ln h_5}{\ln\left(\frac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}\right) - (1-d)} \right\rceil \qquad (7.10)$$

is the minimum number of NEs making (7.8) true. Hence, the minimum number of NEs required, so that both (7.7) and (7.8) are true, is max{L,L'}.

This is exactly the situation arised in the design of Algorithm 5.1 (see Subsection 5.5.1). L and L' are increasing and decreasing functions of d, respectively, and therefore, the d making $L=L'$ is the d minimizing the $\max\{L,L'\}$, i.e. the d yielding the minimum number of NEs making both (7.7) and (7.8) true. ■

The new function, testing a CNF against this minimum number of NEs, is given next.

Function d-conflict-max-g-r-NE-II : boolean

Task : Return the value TRUE, if the frequency of the seen noisy NEs of f, making g 1/*, is greater than d; otherwise return FALSE.

Assumptions: g is in maximal n-CNF, for some $1 \leq n \leq t$.

Parameters : h_3, h_4, h_5; set up in calling routine.

```
COUNT <-- 0
a <-- $\ln h_5$
b <-- $\ln h_4$
x <-- $\ln\left(\frac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}\right)$
y <-- $\ln\left(\frac{eh_3(1-r)^{-1}}{eh_3(1-r)^{-1}+1-e}\right)$
d <-- $\frac{y(1+a)+b(1-x)}{a+b+x+y}$
M <-- $\left\lceil\frac{a+b+1}{x+y-1}\right\rceil$
repeat M times
    begin
```

```
10.     v <-- r-EXAMPLE$^-$
11.     if $s_i(v)\neq 0$   $\forall s_i \in g$          [if $g(v)\neq 0$]
12.        then   COUNT <-- COUNT+1
13.   end
14. d-conflict-max-g-r-NE-II <-- $\left(\frac{\text{COUNT}}{M} > d\right)$          ■
```

If this function is used, instead of the "d-conflict-max-g-r-NE-I" used in Algorithm 7.1, the following algorithm is obtained.

Algorithm 7.3.

Task : Learn f from error-free PEs and noisy NEs.

Assumptions: (a) $x_1,\ldots,x_t$ are sufficient to express f.

(b) Error rate r of NEs generator is known.

Parameters : h_1, h_2, h_3, h_4, $h_5 > 1$.

```
 1. n <-- 1                              [set n to 1]
 2. call Algorithm 4.1 with k<--n        [learn n-CNF g from PEs]
 3. if n<t                               [if g not least general]
 4.   then
 5.     if d-conflict-max-g-r-NE-II      [if g is over-general]
 6.       then
 7.         begin
 8.           n <-- n+1                  [increment n]
 9.           go to 2                    [and try again]
10.         end                                                   ■
```

Unavoidably, since Theorem 2.5 was used in the derivations of both L and L', in (7.9) and (7.10), some restrictions, stemming from (2.34), will be imposed on the problem's parameters.

Theorem 7.7. Algorithm 7.3 is applicable if and only if

$$0 < r < r_{max} = \frac{\sqrt{e}-1}{e-1} \approx .3775 \qquad (7.11)$$

holds for the error rate of the NEs generator, and also

$$1 < h_3 < h_{max} = \frac{(1-r)(er+1-r)}{er} \qquad (7.12)$$

holds for the user-specified parameter h_3. Moreover,

$$r \dashrightarrow r_{max} \quad \text{if and only if} \quad h_{max} \dashrightarrow 1. \qquad (7.13)$$

Proof: The proof is similar to that of Theorem 5.1, and it is briefly stated here.

Clearly, the number M of NEs seen in each run of Algorithm 7.3 has to be a positive integer. Hence

$$1 - \ln\left(\frac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}\right) < \ln\left(\frac{eh_3(1-r)^{-1}}{eh_3(1-r)^{-1}+1-e}\right) \qquad (7.14)$$

must be true, or equivalently,

$$\frac{e}{\dfrac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}} < \frac{eh_3(1-r)^{-1}}{eh_3(1-r)^{-1}+1-e}$$

must be true. The solution of this inequality yields

$$h_3 < \frac{(1-r)(er+1-r)}{er},$$

which proves the necessity of (7.12).

From (7.12) now, it is clear that

$$1 < \frac{(1-r)(er+1-r)}{er}$$

must hold, or equivalently,

$$(1-e)r^2-2r+1 > 0$$

must hold. According to what was said in Subsection 2.2.3, the solution of the last inequlity is

$$-1.5415 \simeq \frac{1+\sqrt{e}}{1-e} < r < \frac{1-\sqrt{e}}{1-e} \simeq .3775,$$

which proves the necessity of (7.11).
Clearly also, (7.13) is true, since (7.11) was obtained from the solution of $1<h_{max}$, from (7.12). ■

A comparison of Theorems 7.1 and 7.7 reveals that, in contrast to Algorithm 7.1, Algorithm 7.3 works only for a restricted range of error rates of the NEs generator. This can be explained as follows.

First, if the output of Algorithm 7.3 has a fixed reliability (i.e. if h_3 and h_4 are not changed), then

when r increases, d's lower bound decreases. (7.15)

That is so, because when the number of noisy NEs increases, on the one hand, the number of conflicting NEs which are error-free (i.e. which remain conflicting) decreases, and on the other hand, the number of non-conflicting NEs which are changed to conflicting ones can either decrease or increase (see also discussion after Theorem 7.2). Therefore, in the worst case, the number of observed conflicts decreases. Hence, d has to decrease, in order for the algorithm to catch the few conflicts observed, and determine correctly whether the amount of over-generalization actually occurred is acceptable or not.

Second, if the probability that k runs will be executed is fixed (i.e. if h_5 is not changed), then

when r increases, d's upper bound increases. (7.16)

That is so because after k runs, the learned concept is more specific than the concept to be learned. That implies that when the number of noisy NEs increases, the number of conflicts increases, too. Therefore, d has to increase, in order for the algorithm not to be fooled and ask for additional runs.

From (7.15) and (7.16) it is clear that when r increases, the range of permissible values for the threshold d is shrinking down. Since d should always have a value, r should not be allowed to grow arbitrarily. ■

The next theorem shows that the properties of the output of Algorithm 7.3 are identical to those of Algorithm 7.1, although the two algorithm process a different number of NEs.

<u>Theorem 7.8.</u> The output g of Algorithm 7.3 has the properties:

(a) $P\left[\sum_{g(v)\neq 1} D^{+}(v) \geq h_1^{-1}\right] \leq h_2^{-1}$.

(b) $P\left[\sum_{g(v)\neq 0} D^{-}(v) \geq h_3^{-1}\right] \leq h_4^{-1}$.

(c) g is in maximal n-CNF, for some $n \leq t$.

Proof:

(a) Same as in Theorem 7.2(a).

(b) Initially, the proof proceeds as in Theorem 7.2(b), the only difference being that the number of NEs required per run is M, instead of L. That implies that (7.4) will be changed to

$$P\left[\sum_{g(v)\neq 0} D^{-}(v) \geq h_3^{-1}\right] \leq b^{*}\left(\leq dM; M, \geq h_3^{-1}(1-r)\right). \quad (7.17)$$

Theorem 2.5 now implies that

$$\min\left\{n \in N \mid b^{*}\left(\leq dn; n, \geq h_3^{-1}(1-r)\right) \leq h_4^{-1}\right\} \leq M_1 \quad (7.18)$$

where

$$M_1 = \left\lceil \frac{d + \ln h_4}{\ln\left(\frac{eh_3(1-r)^{-1}}{eh_3(1-r)^{-1}+1-e}\right) - d} \right\rceil, \quad (7.19)$$

provided of course that (2.34) holds, or, according to the definitions of d and y, provided that

$$\frac{y(1+a)+b(1-x)}{a+b+x+y} < y \quad (7.20)$$

holds. But (7.20) is equivalent to

$$(b+x)(1-x-y) < 0,$$

which is always true, since Theorem 7.7 implies that $M>0$, and therefore, $1-x-y<0$. That establishes (7.20), which, in turn, establishes (7.18). According now to the definition of d, it can be verified that the M_1 of (7.19) is equal to the M of Algorithm 7.3. Therefore, (7.18) implies

$$b^*\left(\leq dM;\ M,\ \geq h_3^{-1}(1-r)\right) \leq h_4^{-1},$$

which, according to (7.17), proves the result.

(c) Same as in Theorem 7.2(c). ■

The formal results, concerning the role of the user-specified paremeter h_5, as well as the time complexity of Algorithm 7.3, conclude the algorithm's analysis.

Theorem 7.9. If Algorithm 7.3 is applied to learn a k-CNF concept, then

(a) P[more than k runs executed] $\leq h_5^{-1}$.

(b) The algorithm's worst-case time complexity is

$$T(t, h_4, h_2, h_4, h_5) =$$
$$=\Theta\left(t^{n+1}\left(h_1t^n+h_1\ln h_2+\frac{\ln(h_4h_5)}{\ln\left(\frac{eh_3}{(er-r+1)(er+eh_3-r+1)}\right)}\right)\right)$$

where n is the number of runs executed.

Proof:

(a) Suppose that more than k runs are executed. Clearly, after the k-th run, the learned concept g will be more specific than the concept f to be learned. If each output, v, of r-EXAMPLE$^-$, seen during the last run, is interpreted as a Bernoulli trial, with "success" being "v makes g 0", then M independent and equiprobable

Bernoulli trials have been performed, and at most M-dM successes have occurred (since this is not the last run). If p denotes the probability that the noisy output v of r-EXAMPLE^{-} makes g 0, then the probability of success of each Bernoulli trial is

$$P[\text{success}] = P[\text{no error by r-EXAMPLE}^{-}] + P[\text{error by r-EXAMPLE}^{-}] * p,$$

which implies that

$$P[\text{success}] \geq 1-r.$$

The above imply that the event

"more than k runs are executed"

is equivalent to the event

"the probability of success of each of the M trials (in the k-th run) is at least 1-r, and at most M-dM successes have occurred".

Therefore,

$$P[\text{more than k runs are executed}] = b^{*}(\leq M-dM; M, \geq 1-r). \tag{7.21}$$

Theorem 2.5 now implies that

$$\min\left\{n\in N \,\middle|\, b^{*}\left(\leq(1-d)n;\ n,\ \geq 1-r\right) \leq h_5^{-1}\right\} \leq M_2, \tag{7.22}$$

where

$$M_2 = \left\lceil \frac{(1-d) + \ln h_5}{\ln\left(\frac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}\right) - (1-d)} \right\rceil, \tag{7.23}$$

provided of course that (2.34), or that

$$1 - \frac{y(1+a)+b(1-x)}{a+b+x+y} < x \tag{7.24}$$

holds. But (7.20) is equivalent to

$$(a+x)(1-x-y) < 0,$$

which is always true, since Theorem 7.7 implies that $M>0$. Therefore, (7.24) is true, which implies that (7.22) is true. It can now be verified that $M_2=M$, and therefore (7.22) implies that

$$b^*\left(\le(1-r)M;\ M,\ \ge 1-r\right) \le h_5^{-1},$$

which, according to (7.21), proves the result.

(b) It can be easily verified that

$$\ln\left(\frac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}\right) + \ln\left(\frac{eh_3(1-r)^{-1}}{eh_3(1-r)^{-1}+1-e}\right) - 1$$

$$= \ln\left(\frac{eh_3}{(er-r+1)(er+eh_3-r+1)}\right). \qquad (7.25)$$

Hence

$$M = \Theta\left(\frac{\ln(h_4h_5)}{\ln\left(\frac{eh_3}{(er-r+1)(er+eh_3-r+1)}\right)}\right). \qquad (7.26)$$

Similar reasoning, now, to that of Theorem 7.3, proves the result. ■

Although Algorithm 7.3, as well as Algorithm 5.1, have both r and h_3 upper-bounded, the time complexity analysis of Algorithm 7.3 was much easier than that of Algorithm 5.1. That was so, because r_{max} of Algorithm 7.3 does not depend on t, as was the case with the r_{max} of Algorithm 5.1. Therefore, r can appear as a parameter in the asymptotic time complexity expression of Algorithm 7.3 (see also discussion in the beginning of Section 5.6). ■

It is also worth mentioning that no simplification of the time complexity in Theorem 7.9 is possible, since Theorem 7.7 and (7.25) imply that it is the very denominator of (7.26) which tends to 0, when $h_3 \dashrightarrow h_{max}$, or $r \dashrightarrow r_{max}$; this information should not be lost. ■

Since both Algorithms 7.1 and 7.3 learn a CNF concept from error-free PEs and noisy NEs, a comparison of them is in place. Clearly, the advantage of **Algorithm 7.3** is that **it has a user-specified probability to stop after k runs.** However, a price has to be paid for that: **Algorithm 7.1 is applicable for any error rate of the NEs generator, whereas Algorithm 7.3 is applicable for error rates of only up to .37.** ■

7.5. DNF Learning from Noisy PEs - Case II

The dual of Algorithm 7.3, performing DNF learning from noisy PEs and error-free NEs, is presented in this section. This algorithm can also be viewed as an extension of Algorithm 7.2, in exactly the same way Algorithm 7.3 was viewed as an extension of Algorithm 7.1. To be more specific, an additional user-specified parameter h_5 is introduced in Algorithm 7.2, bounding the probability that more than k runs will be executed. This parameter uniquely defines the best choice for the threshold d, which is now assigned a particular value, instead of being left as a user-specified parameter, as was the case with Algorithm 7.2. ■

The function performing the test of the learned DNF concept against the PEs, is given first, followed by the algorithm's description and its theorems.

Function d-conflict-max-g-r-PE-II : boolean

Task : Return the value TRUE, if the frequency of the seen noisy PEs of f, making g 0/*, is greater than d; otherwise return FALSE.

Assumptions: g is in maximal n-DNF, for some $1 \le n \le t$.

Parameters : h_1, h_2, h_5; set up in calling routine.

```
1. COUNT <-- 0
2. a <-- $\ln h_5$
3. b <-- $\ln h_2$
4. x <-- $\ln\left(\frac{e(1-r)^{-1}}{e(1-r)^{-1}+1-e}\right)$
5. y <-- $\ln\left(\frac{eh_1(1-r)^{-1}}{eh_1(1-r)^{-1}+1-e}\right)$
6. d <-- $\frac{y(1+a)+b(1-x)}{a+b+x+y}$
7. M <-- $\left\lceil\frac{a+b+1}{x+y-1}\right\rceil$
8. repeat M times
9.    begin
```

```
10.     v <-- r-EXAMPLE$^+$
11.     if $p_i(v) \neq 1$   $\forall p_i \in g$          [if $g(v) \neq 1$]
12.        then  COUNT <-- COUNT+1
13.   end
14. d-conflict-max-g-r-PE-II <-- $\left(\frac{COUNT}{M} > d\right)$
```
■

<u>Algorithm 7.4.</u>

Task : Learn f from noisy PEs and error-free NEs.

Assumptions: (a) $x_1, \ldots, x_t$ are sufficient to express f.

(b) Error rate r of PEs generator is known.

Parameters : $h_1, h_2, h_3, h_4, h_5 > 1$.

```
 1. n <-- 1                              [set n to 1]
 2. call Algorithm 4.2 with k<--n        [learn n-DNF g from NEs]
 3. if n<t                               [if g not least specific]
 4.    then
 5.       if d-conflict-max-g-r-PE-II[if g is over-specific]
 6.          then
 7.            begin
 8.               n <-- n+1              [increment n]
 9.               go to 2                [and try again]
10.            end
```
■

<u>Theorem 7.10.</u> Algorithm 7.4 is applicable if and only if

$$0 < r < r_{max} = \frac{\sqrt{e} - 1}{e - 1} \simeq .3775 \qquad (7.27)$$

holds for the error rate of the PEs generator, and also

$$1 < h_1 < h_{max} = \frac{(1-r)(er+1-r)}{er} \qquad (7.28)$$

holds for the user-specified parameter h_1. Moreover,

$$r \dashrightarrow r_{max} \text{ if and only if } h_{max} \dashrightarrow 1\,. \qquad (7.29)$$

Proof: Same as in Theorem 7.7. ■

<u>Theorem 7.11.</u> The output g of Algorithm 7.4 has the properties:

(a) $P\left[\sum_{g(v)\neq 1} D^{+}(v) \geq h_1^{-1}\right] \leq h_2^{-1}$.

(b) $P\left[\sum_{g(v)\neq 0} D^{-}(v) \geq h_3^{-1}\right] \leq h_4^{-1}$.

(c) g is in maximal n-DNF, for some $n \leq t$.

Proof: The proofs of (a), (b), and (c), are the duals of those of (b), (a), and (c) of Theorem 7.8. ■

<u>Theorem 7.12.</u> If Algorithm 7.4 is applied to learn a k-DNF concept, then

(a) P[more than k runs executed] $\leq h_5^{-1}$.

(b) The algorithm's worst-case time complexity is

$$T(t,\ h_2,\ h_3,\ h_4,\ h_5) =$$
$$=\Theta\left(t^{n+1}\left(h_3 t^n + h_3 \ln h_4 + \frac{\ln(h_2 h_5)}{\ln\left(\frac{eh_1}{(er-r+1)(er+eh_1-r+1)}\right)}\right)\right)$$

where n is the number of runs executed.

Proof: Same as in Theorem 7.9. ■

7.6. Discussion

A different method for concept learning from error-free PEs and noisy NEs could have been followed. Instead of applying Algorithm 7.1 or 7.2, i.e. instead of learning a CNF from error-free PEs (by applying Algorithm 4.1) and testing it against noisy NEs, learn a DNF concept from noisy NEs (by applying Algorithm 5.2) and test it against error-free PEs.

However, this approach has not been considered further, because it has two disadvantages. First, according to Theorem 5.4, it restricts the range of permissible error rates for the NEs generator. Second, according to Theorem 5.5(c), it is NP-hard to determine whether a partial vector is a PE of the learned concept.

Similarly, the method adopted by Algorithms 7.2 and 7.4, namely, that of learning a DNF from error-free NEs (by applying Algorithm 4.2) and testing it against noisy PEs, is better than that of learning a CNF from noisy PEs, (by applying Algorithm 5.1) and testing it against error-free NEs. ■

7.7. Summary

Four algorithms, performing strong two-sided-error learning from error-free PEs (NEs) and noisy NEs (PEs), without having any knowledge about the representability of the concept to be learned, were presented in this chapter.

Algorithm 7.1 (7.2) is applicable for any error rate of the NEs (PEs) generator, but its worst-case time complexity, considered as a function of t, can grow up to $\Theta(t^{2t+1})$, depending on D^- (D^+) and d. Also, the time required by its output to determine whether a vector is a PE or a NE, can grow up to $\Theta(t^{t+1})$ as well; that also depends on D^- (D^+) and d.

On the other hand, Algorithm 7.3 (7.4) is applicable for error rates of the NEs (PEs) generator, of only up to .37, and its worst-case time complexity can also grow up to $\Theta(t^{2t+1})$. However, if the concept to be learned can be written in k-CNF/k-DNF (with k itself being unknown), then the probability that the worst-case time complexity of Algorithm 7.3 (7.4) will be $\Theta(t^{2k+1})$, as well as the probability that the output of the algorithm can determine in $\Theta(t^{k+1})$ time whether a vector is a PE or a NE, can be bounded by a user-specified parameter, logarithmically affecting the algorithm's time complexity. ■

Clearly, the problem of concept learning from one error-free and **one noisy** example generator, studied in this chapter, is more realistic than that of concept learning from two error-free example generators, studied in the last chapter. In addition, this chapter's results can also be viewed as a preparation towards the solution of an even more realistic problem: that of concept learning from **two noisy** example generators. This is the subject of the next chapter, and is also the last problem addressed in our thesis. ■

CHAPTER 8

k-CONCEPT LEARNING; UNKNOWN k; TWO NOISY EXAMPLE GENERATORS

8.1. Introduction

Having studied the problem of concept learning from PEs and NEs, without having any information about the concept's representability, when both example generators are error-free (Chapter 6), as well as when only one of them is noisy (Chapter 7), we will now study the same problem, when both example generators are noisy.

The main idea used in the design of the algorithms of Chapters 6 and 7, has also been used in the design of the algorithms of this chapter. Once more, the problem addressed here, has not been considered elsewhere. ■

A CNF learning algorithm is presented is Section 8.2. Its dual algorithm, performing DNF learning, is presented in Section 8.3. A second version of each of those two algorithms is presented in Sections 8.4 and 8.5. The chapter closes with a summary, in Section 8.6. ■

8.2. CNF Learning - Case I

If both example generators are noisy, and no information about the representability of the concept to be learned is available, then Algorithm 7.1, learning a CNF concept from error-free PEs and noisy NEs, can be modified to cope with the additional problem of the noisy PEs: instead of applying Algorithm 4.1 to learn a 1-CNF, 2-CNF, etc., from error-free PEs, apply Algorithm 5.1 to learn a 1-CNF, 2-CNF, etc., from noisy PEs. In both cases, a new run is going to be initiated if significant over-generalization is detected.

Here is where the similarities between Algorithm 7.1 and the new algorithm stop. First of all, according to Theorem 5.2(c), the output g of Algorithm 5.1 is not maximal, fact which, according to our very useful Theorem 3.3, implies that it cannot be tested whether a NE makes the learned g 1/*, i.e. whether a conflict (the way it has been considered so far) has occurred. The only solution to this problem is to test whether a NE makes g 1, instead of 1/*. The implication of such a test is that the new algorithm is going to perform weak, rather than strong, two-sided-error learning (see Definition 3.20).

Obviously, more differences will exist between the new algorithm and Algorithm 7.1. They are inherited from

Algorithm 5.1, and they will all be stated in the theorems accompanying the new algorithm. ■

The function performing the test of the learned CNF against NEs, is given first. Although its only actual difference from the "d-conflict-max-g-r-NE-I" function is its test for g(v)=1, instead of g(v)≠0, it is fully stated here, for the sake of completeness.

<u>Function</u> d'-conflict-g-r^--NE-I : boolean

Task : Return the value TRUE, if the frequency of the seen noisy NEs of f, making g 1, is greater than d'; otherwise return FALSE.

Assumptions: g is in (not necessarily maximal) n-CNF, for some $1 \leq n \leq t$.

Parameters : h_3, h_4, and d'; set up in calling routine.

```
COUNT <-- 0
repeat $L = \left\lceil \dfrac{d' + \ln h_4}{\ln\left(\dfrac{eh_3(1-r^-)^{-1}}{eh_3(1-r^-)^{-1}+1-e}\right) - d'} \right\rceil$ times
  begin
    v <-- $r^-$-EXAMPLE$^-$
    if $s_i(v)=1 \quad \forall s_i \in g$          [if g(v)=1]
      then COUNT <-- COUNT+1
  end
d'-conflict-g-$r^-$-NE-I <-- $\left(\dfrac{COUNT}{L} > d'\right)$
```

■

Notice that the name d', instead of the usual d, is used for the threshold of this function. This is done so, because d is the threshold used by Algorithm 5.1, which is called by Algorithm 8.1, described below. Keep in mind though, that d is not stated explicitely in the algorithm's description. ■

<u>Algorithm 8.1.</u>

Task : Learn f from noisy PEs and noisy NEs.

Assumptions: (a) $x_1,\ldots,x_t$ are sufficient to express f.

(b) Error rates r^+ and r^- of PEs and NEs generators are known.

Parameters : h_1, h_2, h_3, h_4 > 1, and d' > 0.

```
 1. n <-- 1                              [set n to 1]
 2. call Algorithm 5.1
    with k<--n and r<--$r^+$             [learn n-CNF g from PEs]
 3. if n<t                               [if g not least general]
 4.   then
 5.     if d'-conflict-g-$r^-$-NE-I      [if g is over-general]
 6.       then
 7.         begin
 8.           n <-- n+1                  [increment n]
 9.           go to 2                    [and try again]
10.         end                                              ■
```

First, the conditions under which this algorithm is applicable, are given.

Theorem 8.1. Algorithm 8.1 is applicable if and only if

$$0 < r^+ < r^+_{max} = \frac{\sqrt{(eA_t+2-e)^2+4(e-1)}-(eA_t+2-e)}{2(e-1)} \qquad (8.1)$$

holds for the error rate r^+ of the PEs generator,

$$1 < h_1 < h_{max} = \frac{(1-r^+)(er^++1-r^+)}{eA_t r^+} \qquad (8.2)$$

holds for the user-specified parameter h_1, and also

$$0 < d' < \ln\left(\frac{eh_3(1-r^-)^{-1}}{eh_3(1-r^-)^{-1}+1-e}\right) \qquad (8.3)$$

holds for the user-specified parameter d'.

Proof: The call of Algorithm 5.1 necessitates (8.1) and (8.2) (see Theorem 5.1); since the concept to be learned may have up to t literals per disjunct, the A_k of Theorem 5.1 has to be changed to A_t. Also, the definition of L in the "d'-conflict-g-r^--NE-I" function, necessitates (8.3).

The fact that r^+_{max} is a function of A_t, rather than a function of A_k (k is not known), has serious consequences, which should not be overlooked. According to Lemma 5.2,

$$\frac{.3679}{A_t} \le r^+_{max} \le \frac{.3954}{A_t}$$

holds. The problem is that A_t grows very fast (for example, $A_2=2$, $A_3=26$, $A_4=80$, etc.), and therefore, in practical applications, the maximum error rate of the PEs generator, acceptable by Algorithm 8.1, has to be very small. ■

The next theorem shows that, as already claimed in the beginning of this section, Algorithm 8.1 performs weak two-sided-error learning.

<u>Theorem 8.2.</u> The output g of Algorithm 8.1 has the properties:

(a) $P\left[\sum_{g(v)\neq 1} D^{+}(v) \geq h_1^{-1}\right] \leq h_2^{-1}$.

(b) $P\left[\sum_{g(v)=1} D^{-}(v) \geq h_3^{-1}\right] \leq h_4^{-1}$.

(c) g is in n-CNF, for some $n \leq t$; g is not necessarily in maximal n-CNF.

Proof:

(a) Since the output of Algorithm 8.1 is nothing but the output of Algorithm 5.1 (called in line 2), the proof is the same as that of Theorem 5.2(a), if we set k<--n and consider only the last run.

(b) (i) Suppose that t runs are executed (i.e. the t-CNF learned is not tested against NEs).

According to Theorem 5.2(b)

$$P\left[\sum_{g(v)\neq 0} D^{-}(v) > 0\right] \leq h_4^{-1} \qquad (8.4)$$

holds. But

$$P\left[\sum_{g(v)=1} D^{-}(v) \geq h_3^{-1}\right] \leq P\left[\sum_{g(v)\neq 0} D^{-}(v) \geq h_3^{-1}\right] (8.5)$$

is always true, as well as

$$P\left[\sum_{g(v)\neq 0} D^{-}(v) \geq h_3^{-1}\right] \leq P\left[\sum_{g(v)\neq 0} D^{-}(v) > 0\right] (8.6)$$

The result now follows immediately from (8.4), (8.5), and (8.6).

(ii) Suppose that less than t runs are executed (i.e. the k-CNF learned is successfully tested against NEs).

The proof proceeds in exactly the same way as that of Theorem 7.2(b)(ii), if V is defined as the set of all NEs making the final output g of Algorithm 8.1 1 (instead of 1/*), and if d' is substituted for d. ■

The asymptotic time complexity of Algorithm 8.1 is given next. Keep in mind that, according to Lemmas 5.2 and 5.3, this is derived for the **entire** range of permissible values of h_1 and r^+.

<u>Theorem 8.3.</u> If c is a constant, with

$$c < \frac{1}{e}, \tag{8.7}$$

and

$$r^+ = \frac{c}{A_t} \quad \text{for all } A_t \tag{8.8}$$

is the error rate of the PEs generator acceptable by Algorithm 8.1, and also

$$h_1 < \frac{1}{ec} \tag{8.9}$$

holds for the user-specified parameter h_1, then the worst-case time complexity of Algorithm 8.1 is

$T(t, h_2, h_3, h_4) =$

$$= O\left(t^{n+1}\left\{\frac{h_1t^n\ln(h_2h_4)}{1-ech_1}+\frac{d'+\ln h_4}{\ln\left(\frac{eh_3(1-r^-)^{-1}}{eh_3(1-r^-)^{-1}+1-e}\right)-d'}\right\}\right) \quad (8.10)$$

and

$$T(t,\ h_2,\ h_3,\ h_4) =$$
$$= \Omega\left(t^{n+1}\left\{\frac{h_1t^{2n+1}\ln(h_2h_4)}{(1-ch_1)t^n-c}+\frac{d'+\ln h_4}{\ln\left(\frac{eh_3(1-r^-)^{-1}}{eh_3(1-r^-)^{-1}+1-e}\right)-d'}\right\}\right) \quad (8.11)$$

where n is the number of runs executed; n can grow up to t.

Proof: According to Theorem 5.3, during the j-th run of Algorithm 8.1, execution of line 2 takes

$$O\left(\frac{h_1t^{2j+1}\ln(h_2h_4)}{1-ech_1}\right)$$

and

$$\Omega\left(\frac{h_1t^{3j+1}\ln(h_2h_4)}{(1-ch_1)t^j-c}\right)$$

time in the worst case. On the other hand, execution of line 5 of the "d'-conflict-g-r$^-$-NE-I" function takes at most $\Theta(t^{j+1})$ time, and therefore, execution of line 5 of Algorithm 8.1 takes

$$\Theta\left(t^{j+1}\frac{d'+\ln h_4}{\ln\left(\frac{eh_3(1-r^-)^{-1}}{eh_3(1-r^-)^{-1}+1-e}\right)-d'}\right)$$

time in the worst case. The result now follows immediately. ■

8.3. DNF Learning - Case I

Algorithm 8.1, presented in the last section, is applicable for any error rate of the NEs generetor, but only for small error rates of the PEs generator. If it happens that the error rate of PEs generator is high, but that of the NEs generator is small, then the dual of Algorithm 8.1, learning a DNF concept, may be applicable.

Notice a difference here, between this pair of dual algorithms on the one hand, and the pairs of dual algorithms of Chapter 6 on the other hand. For the Chapter 6 algorithms, the user was supposed to choose arbitrarily the most suitable algorithm; for the Chapter 8 algorithms, the error rates of the example generators will most probably be the decisive factor for this choise. ■

The function performing the test of the learned DNF against the PEs, is given first. This is the dual of the "d'-conflict-g-r$^-$-NE-I" function. Also, its only actual difference from the "d-conflict-max-g-r-PE-I" function, is that it tests for g(v)=0, instead of g(v)≠1.

Function d'-conflict-g-r$^+$-PE-I : boolean

Task : Return the value TRUE, if the frequency of the seen noisy PEs of f, making g 0, is greater than d'; otherwise return FALSE.

Assumptions: g is in (not necessarily maximal) n-DNF, for some $1 \le n \le t$.

Parameters : h_1, h_2, and d'; set up in calling routine.

```
COUNT <-- 0
repeat  $L = \left\lceil \dfrac{d' + \ln h_2}{\ln\left(\dfrac{e h_1 (1-r^+)^{-1}}{e h_1 (1-r^+)^{-1} + 1 - e}\right) - d'} \right\rceil$  times
  begin
    v <-- $r^+$-EXAMPDE$^+$
    if $p_i(v)=0 \quad \forall p_i \in g$          [if $g(v)=0$]
      then COUNT <-- COUNT+1
  end
d'-conflict-g-$r^+$-PE-I <-- $\left(\dfrac{COUNT}{L} > d'\right)$   ■
```

Keep in mind again, that **two** thresholds are used in Algorithm 8.2, but only one of them is stated explicitely in the algorithm's description below. The threshold d is set up and used in Algorithm 5.2, when a DNF expression is learned from NEs; the threshold d' is user-defined, and it is used in the "d'-conflict-g-r^+-PE-I" function, when the learned DNF is tested against PEs. ■

The DNF learning algorithm is stated next, followed by its theorems.

Algorithm 8.2.

Task : Learn f from noisy PEs and noisy NEs.

Assumptions: (a) $x_1, \ldots, x_t$ are sufficient to express f.

(b) Error rates r^+ and r^- of PEs and NEs generators are known.

Parameters : $h_1, h_2, h_3, h_4 > 1$, and $d' > 0$.

```
 1. n <-- 1                              [set n to 1]
 2. call Algorithm 5.2
    with k<--n and r<--r^-               [learn n-DNF g from NEs]
 3. if n<t                               [if g not least specific]
 4.   then
 5.     if d'-conflict-g-r^+-PE-I        [if g is over-specific]
 6.       then
 7.         begin
 8.           n <-- n+1                  [increment n]
 9.           go to 2                    [and try again]
10.         end
```

■

Theorem 8.4. Algorithm 8.2 is applicable if and only if

$$0 < r^- < r^-_{max} = \frac{\sqrt{(eA_t+2-e)^2+4(e-1)}-(eA_t+2-e)}{2(e-1)} \qquad (8.12)$$

holds for the error rate r^- of the NEs generator,

$$1 < h_3 < h_{max} = \frac{(1-r^-)(er^-+1-r^-)}{eA_t r^-} \qquad (8.13)$$

holds for the user-specified parameter h_3, and also

$$0 < d' < \ln\left(\frac{eh_1(1-r^+)^{-1}}{eh_1(1-r^+)^{-1}+1-e}\right) \tag{8.14}$$

holds for the user-specified parameter d'.

Proof: Theorem 5.4 necessitates (8.12) and (8.13). The number, L, of PEs required per run, necessitates (8.14). ■

Theorem 8.5. The output g of Algorithm 8.2 has the properties:

(a) $P\left[\sum_{g(v)=0} D^+(v) \geq h_1^{-1}\right] \leq h_2^{-1}$.

(b) $P\left[\sum_{g(v)\neq 0} D^-(v) \geq h_3^{-1}\right] \leq h_4^{-1}$.

(c) g is in n-DNF, for some $n \leq t$; g is not necessarily in maximal n-DNF.

Proof: The proofs of (a), (b), and (c) are the duals of those of (b), (a), and (c) of Theorem 8.2. ■

Theorem 8.6. If c is a constant, with

$$c < \frac{1}{e}, \tag{8.15}$$

and

$$r^- = \frac{c}{A_t} \quad \text{for all } A_t \tag{8.16}$$

is the error rate of the NEs generator acceptable by Algorithm 8.2, and also

$$h_3 < \frac{1}{ec} \tag{8.17}$$

holds for the user-specified parameter h_3, then the worst-case time complexity of Algorithm 8.3 is

$$T(t,\ h_1,\ h_2,\ h_4) =$$
$$= O\left(t^{n+1}\left\{\frac{h_3 t^n \ln(h_2 h_4)}{1-ech_3} + \frac{d'+\ln h_2}{\ln\left(\frac{eh_1(1-r^+)^{-1}}{eh_1(1-r^+)^{-1}+1-e}\right)-d'}\right\}\right) \tag{8.18}$$

and

$$T(t,\ h_1,\ h_2,\ h_4) =$$
$$= \Omega\left(t^{n+1}\left\{\frac{h_3 t^{2n+1} \ln(h_2 h_4)}{(1-ch_3)t^n-c} + \frac{d'+\ln h_2}{\ln\left(\frac{eh_1(1-r^+)^{-1}}{eh_1(1-r^+)^{-1}+1-e}\right)-d'}\right\}\right) \tag{8.19}$$

where n is the number of runs executed; n can grow up to t.

Proof: As in Theorem 8.3. ■

8.4. CNF Learning - Case II

Algorithm 8.1. has the same drawback which Algorithm 7.1 had: it may require up to t runs, and, moreover, if a k-CNF concept is to be learned (k itself being unknown), then the probability that more than k runs will be executed can grow up to 1. That is so, because d' has been left as a user-specified parameter; it depends on the unknown D^-, and therefore its optimal value cannot be found.

The problem just described, will be solved by using the method introduced in Chapter 7. That is, the minimum number of NEs which, in addition of satisfying

$$P\left[\sum_{g(v)=1} D^{-}(v) \geq h_3^{-1}\right] \leq h_4^{-1}, \qquad (8.20)$$

i.e. protery (b) of Theorem 8.2, will also satisfy

$$P[\text{more than k runs executed}] \leq h_5^{-1}, \qquad (8.21)$$

for some $h_5>1$, will be determined.

It has been shown that

$$L = \left\lceil \frac{d' + \ln h_4}{\ln\left(\frac{eh_3(1-r^{-})^{-1}}{eh_3(1-r^{-})^{-1}+1-e}\right) - d'} \right\rceil$$

is the minimum number of NEs, satisfying (8.20). It can also been shown (for a complete proof see Theorem 8.9(a)) that

$$L' = \left\lceil \frac{(1-d') + \ln\left(\frac{h_5(h_4-1)}{h_4-h_5}\right)}{\ln\left(\frac{e(1-r^{-})^{-1}}{e(1-r^{-})^{-1}+1-e}\right) - (1-d')} \right\rceil$$

is the minimum number of NEs satisfying (8.21). The minimum number of NEs satisfying both (8.20) and (8.21), can now be easily found (see derivation of M in Subsection 5.5.1, or derivation of M is Section 7.4): find the d' satisfying the L=L', and substitute it to L, to obtain the answer. ■

The new function, testing the learned CNF against a sufficient number of NEs, so that both (8.20) and (8.21) are satisfied, is given next.

<u>Function</u> d'-conflict-g-r$^-$-NE-II : boolean

Task : Return the value TRUE, if the frequency of the seen noisy NEs of f, making g 1, is greater than d'; otherwise return FALSE.

Assumptions: g is in (not necessarily maximal) n-CNF, for some $1 \le n \le t$.

Parameters : h_3, h_4, h_5; set up in calling routine.

```
COUNT <-- 0
a <-- $\ln\left(\frac{h_5(h_4-1)}{h_4-h_5}\right)$
b <-- $\ln h_4$
x <-- $\ln\left(\frac{e(1-r^-)^{-1}}{e(1-r^-)^{-1}+1-e}\right)$
y <-- $\ln\left(\frac{eh_3(1-r^-)^{-1}}{eh_3(1-r^-)^{-1}+1-e}\right)$
d' <-- $\frac{y(1+a)+b(1-\text{[illegible]}}{a+b+x+y}$
M <-- $\left\lceil\frac{a+b+1}{x+y-1}\right\rceil$
repeat M times
  begin
    v <-- r$^-$-EXAMPLE$^-$
    if $s_i(v)=1 \quad \forall s_i \in g$          [if g(v)=1]
      then  COUNT <-- COUNT+1
  end
d'-conflict-g-r$^-$-NE-II <-- $\left(\frac{\text{COUNT}}{M} > d'\right)$
```

■

If the above function is used to test the learned CNF concept for possible over-generalization, then the following algorithm is obtained.

Algorithm 8.3.

Task : Learn f from noisy PEs and noisy NEs.

Assumptions: (a) $x_1, \ldots, x_t$ are sufficient to express f.

(b) Error rates r^+ and r^- of PEs and NEs generators are known.

Parameters : $h_1, h_2, h_3, h_4, h_5 > 1$.

```
 1. n <-- 1                                [set n to 1]
 2. call Algorithm 5.1
    with k<--n and r<--$r^+$                  [learn n-CNF g from PEs]
 3. if n<t                                 [if g not least general]
 4.    then
 5.      if d'-conflict-g-$r^-$-NE-II         [if g is over-general]
 6.        then
 7.          begin
 8.            n <-- n+1                   [increment n]
 9.            go to 2                     [and try again]
10.          end
```
■

Although the description of Algorithm 8.3 may look the same as that of Algorithm 8.1, except for line 5, the properties of the two algorithms are different.

First of all, the next theorem shows that the error rates of both example generators are upper-bounded. As it is also the case with Algorithm 8.1, the error rate of the PEs generator in inversely proportional to A_t; that means that for practical applications, it is too small. On the other hand, the error rate of the NEs generator can be considerably high. ■

<u>Theorem 8.7.</u> Algorithm 8.3 is applicable if and only if

$$0 < r^+ < r^+_{max} = \frac{\sqrt{(eA_t+2-e)^2+4(e-1)}-(eA_t+2-e)}{2(e-1)} \quad (8.22)$$

and

$$0 < r^- < r^-_{max} = \frac{\sqrt{e}-1}{e-1} \simeq .3775 \quad (8.23)$$

hold for the error rates r^+ and r^- of the PEs and NEs generators, respectively, and also

$$1 < h_1 < h_{1,max} = \frac{(1-r^+)(er^++1-r^+)}{eA_tr^+}, \quad (8.24)$$

as well as

$$1 < h_3 < h_{3,max} = \frac{(1-r^-)(er^-+1-r^-)}{er^-}, \quad (8.25)$$

and

$$h_5 < h_4 \quad (8.26)$$

hold for the user-specified parameters h_1, h_3, h_4, and h_5.

Proof: (8.22) and (8.24) are necessary preconditions for the applicability of Algorithm 5.1 (see Theorem 5.1). Also, the definition of M, in the "d'-conflict-g-r$^-$-NE-II" function, implies, on the one hand, that the denominator of M has to be positive, which, in turn implies that (8.23) and (8.25) must hold (see derivation of (7.11) and (7.12) from (7.14)), and on the other hand, that

$$\frac{h_5(h_4-1)}{h_4-h_5} > 0$$

must be true, which necessitates (8.26).

The definition of M has one more implication, namely that

$$\ln h_4 + \ln\left(\frac{h_5(h_4-1)}{h_4-h_5}\right) + 1 > 0$$

must hold, or equivalently, that

$$\frac{eh_4h_5(h_4-1)}{h_4-h_5} > 1$$

must hold. But the last inequality is equivalent to

$$h_5 > \frac{h_4}{eh_4(h_4-1)+1},$$

which is always true, since it is covered by the condition $h_5>1$. That is so because

$$\frac{h_4}{eh_4(h_4-1)+1} < 1$$

is equivalent to

$$eh_4^2-(e+1)h_4+1 > 0,$$

which, according to what was stated in Subsection 2.2.3, is true for all $h_4>1$. Therefore, no additional restrictions on h_4 and h_5 should be imposed.

Since the numerator of M is always positive, it follows that the ranges of the permissible values of the problem's parameters cannot be augmented by considering the case in which both the numerator and the denominator are negative. The proof now of the theorem is complete. ■

The next theorem shows that Algorithm 8.3 performs weak two-sided-error learning.

<u>Theorem 8.8.</u> The output g of Algorithm 8.3 has the properties:

(a) $P\left[\sum_{g(v)\neq 1} D^{+}(v) \geq h_1^{-1}\right] \leq h_2^{-1}$.

(b) $P\left[\sum_{g(v)=1} D^{-}(v) \geq h_3^{-1}\right] \leq h_4^{-1}$.

(c) g is in n-CNF, for some $n \leq t$; g is not necessarily in maximal n-CNF.

Proof:

(a) Same as in Theorem 8.2(a).

(b) (i) Suppose that t runs are executed (i.e. the t-CNF learned is not tested against NEs).

The proof is identical to that of Theorem 8.2(b)(i).

(ii) Suppose that less than t runs are executed (i.e. the k-CNF learned is succesfully tested against NEs).

If each output, v, of r^--EXAMPLE$^-$, seen during the last run, is interpreted as a Bernoulli trial, with "success" being " v makes g 1", then M (defined in function "d'-conflict-g-r$^-$-NE-II") independent and equiprobable Bernoulli trials have been performed, each with probability of success at least $h_3^{-1}(1-r^-)$ (see (7.3)), and at most d'M successes have occurred. Therefore,

$$P\left[\sum_{g(v)=1} D^-(v) \geq h_3^{-1}\right] \leq$$

$$\leq b^*\left(\leq d'M;\ M,\ \geq h_3^{-1}(1-r^-)\right) \qquad (8.27)$$

is true. Theorem 2.5, now, implies that

$$\min\left\{n \in N \mid b^*\left(\leq d'n;\ n,\ \geq h_3^{-1}(1-r^-)\right) \leq h_4^{-1}\right\} \leq M_1 \quad (8.28)$$

where

$$M_1 = \left\lceil \frac{d' + \ln h_4}{\ln\left(\dfrac{eh_3(1-r^-)^{-1}}{eh_3(1-r^-)^{-1}+1-e}\right) - d'} \right\rceil, \qquad (8.29)$$

provided of course that (2.34), or that

$$\frac{y(1+a)+b(1-x)}{a+b+x+y} < y \qquad (8.30)$$

holds. But Theorem 8.7 implies that (8.30) is true (see derivation of (7.20)), and therefore (8.28) is true, as well. According now to the definition of d', (8.29) implies that M_1=M, and hence (8.28) implies

$$b^*\left(\leq d'M;\ M,\ \geq h_3^{-1}(1-r^-)\right) \leq h_4^{-1},$$

which, together with (8.27), prove that result.

(c) Same as in Theorem 8.2(c). ■

The role of the user-specified parameter h_5, as well as the time complexity of Algorithm 8.3, are given next. Keep in mind, once more, that in spite of the conditions (8.31), (8.32), and (8.33), the asymptotic time complexity expression for Algorithm 8.3, is given for the entire range of the permissible values of r^+ and h_1.

Theorem 8.9. If Algorithm 8.3 is applied to learn a k-CNF concept, then

(a) P[more than k runs executed] $\leq h_5^{-1}$.

(b) If c is a constant, with

$$c < \frac{1}{e}\,, \tag{8.31'}$$

and

$$r^+ = \frac{c}{A_t} \quad \text{for all } A_t \tag{8.32}$$

is an error rate of the PEs generator, acceptable by Algorithm 8.3, and also

$$h_1 < \frac{1}{ec} \tag{8.33}$$

holds for the user-specified parameter h_1, then the worst-case time complexity of the algorithm is

$$T(t,\, h_2,\, h_4) = O\left(t^{n+1}\left\{\frac{h_1 t^n + \ln(h_2 h_4)}{1-ech_1} + \frac{\ln\left(\frac{h_4 h_5}{h_4-h_5}\right)}{\ln\left(\frac{eh_3}{(er^- - r^- +1)(er^- + eh_3 - r^- +1)}\right)}\right\}\right)$$

and

$T(t,\, h_2,\, h_4) =$

$$= \Omega\left(t^{n+1}\left\{\frac{h_1 t^{2n+1}+\ln(h_2 h_4)}{(1-ch_1)t-c} + \frac{\ln\left(\frac{h_4 h_5}{h_4-h_5}\right)}{\ln\left(\frac{eh_3}{(er^- - r^- + 1)(er^- + eh_3 - r^- + 1)}\right)}\right\}\right)$$

where n is the number of runs executed.

Proof:

(a) Suppose that exactly k runs are executed, and let V be the set of NEs making the final output of Algorithm 8.3 1. If each output, v, of r^--EXAMPLE$^-$, seen during the last (k-th) run, is interpreted as a Bernoulli trial, with "success" being "v makes g 1", then M independent and equiprobable Bernoulli trials have been performed, and at most d'M successes have occurred (since this is the last run). Hence

$$P[\text{k runs executed}] =$$
$$= P[\le d'M \text{ successes} \mid V=\emptyset] * P[V=\emptyset]$$
$$+ P[\le d'M \text{ successes} \mid V\ne\emptyset] * P[V\ne\emptyset]$$

holds, which implies that

$$P[\text{k runs executed}] \ge$$
$$\ge P[\le d'M \text{ successes} \mid V=\emptyset] * P[V=\emptyset] \qquad (8.34)$$

holds. Since

$$P\left[\sum_{g(v)=1} D^-(v) > 0\right] \le P\left[\sum_{g(v)\ne 0} D^-(v) > 0\right]$$

is true, Theorem 5.2(b) implies that

$$P\left[\sum_{g(v)=1} D^-(v) > 0\right] \le h_4^{-1}$$

or, equivalently,

$$P[V=\emptyset] \geq 1-h_4^{-1}. \qquad (8.35)$$

Clearly, also, when $V=\emptyset$ (i.e. when g is more specific than f), the probability of success of each Bernoulli trial is at most r^-. Therefore, according to (8.35), (8.34) can be rewritten as

$$P[\text{k runs executed}] \geq b^*(\leq d'M;\ M,\ <r^-)*(1-h_4^{-1})$$

or, equivalently,

$$P[\text{k runs executed}] \geq b^*(>M-d'M; M, \geq 1-r^-)*(1-h_4^{-1})$$

which is equivalent to

$$P[\text{k runs executed}] \geq$$
$$\geq \left(1 - b^*\left(\leq M-d'M;\ M,\ \geq(1-r^-)\right)\right)*\left(1-h_4^{-1}\right) \qquad (8.36)$$

Theorem 2.5, now, implies that

$$\min\left\{n\in N \,\middle|\, b^*\left(\leq n-d'n; n, \geq 1-r^-\right) \leq 1-\frac{1-h_5^{-1}}{1-h_4^{-1}}\right\} \leq M_2 \qquad (8.37)$$

where

$$M_2 = \left\lceil \frac{(1-d') + \ln\left(\frac{h_5(h_4-1)}{h_4-h_5}\right)}{\ln\left(\frac{e(1-r^-)^{-1}}{e(1-r^-)^{-1}+1-e}\right) - (1-d')} \right\rceil, \qquad (8.38)$$

provided of course that (2.34), or that

$$1 - \frac{y(1+a)+b(1-x)}{a+b+x+y} < x$$

holds. But this is equivalent to

$$(a+x)*(1-x-y) < 0,$$

which, according to Theorem 8.7, is always true. Therefore, (8.37) is true, and since it can be verified that M_2 (in (8.38)) is equal to M (the number of NEs required by the algorithm), (8.37) implies that.

$$b^*\left(\le M-d'M;\ M,\ \ge 1-r^-\right) \le 1-\frac{1-h_5^{-1}}{1-h_4^{-1}}. \qquad (8.39)$$

From (8.36), now, and from (8.39), it can be concluded that

$$P[k\text{ runs executed}] \ge 1-h_5^{-1},$$

which implies that

$$P[k\text{ or more runs executed}] \ge 1-h_5^{-1},$$

and the result follows.

(b) According to Theorem 5.3, during the j-th run of Algorithm 8.3, execution of line 2 takes

$$O\left(\frac{h_1t^{2j+1}\ln(h_2h_4)}{1-ech_1}\right)$$

and

$$\Omega\left(\frac{h_1t^{3j+1}\ln(h_2h_4)}{(1-ch_1)t^j-c}\right)$$

time in the worst case. Also, the established (7.25) implies that the "d'-conflict-g-r^--NE-II" function takes

$$\Theta\left(t^{j+1}\frac{\ln\left(\frac{h_4h_5}{h_4-h_5}\right)}{\ln\left(\frac{eh_3}{(er^- - r^- +1)(er^- +eh_3-r^- +1)}\right)}\right)$$

time in the worst case. The result now follows immediately. ■

8.5. DNF Learning - Case II

If the error rate of the NEs generator is very small, whereas that of the PEs generator is higher (up to .37), then the dual of Algorithm 8.3 may be applicable. This algorithm has the advantage, over Algorithm 8.2, that, if a k-DNF concept is to be learned (k itself being unknown), then the probability that more than k runs will be executed, is upper-bounded by a user-specified parameter. ■

The function performing the test of the DNF, learned from NEs, against the PEs, is given first.

Function d'-conflict-g-r^+-PE-II : boolean

Task : Return the value TRUE, if the frequency of the seen noisy PEs of f, making g 0, is greater than d'; otherwise return FALSE.

Assumptions: g is in (not necessarily maximal) n-DNF, for some $1\leq n\leq t$.

Parameters : h_1, h_2, h_5; set up in calling routine

```
1. COUNT <-- 0
2. a <-- $\ln\left(\frac{h_5(h_2-1)}{h_2-h_5}\right)$
3. b <-- $\ln h_2$
4. x <-- $\ln\left(\frac{e(1-r^+)^{-1}}{e(1-r^+)^{-1}+1-e}\right)$
```

```
$y \leftarrow \ln\left(\frac{eh_1(1-r^+)^{-1}}{eh_1(1-r^+)^{-1}+1-e}\right)$
$d' \leftarrow \frac{y(1+a)+b(1-x)}{a+b+x+y}$
$M \leftarrow \left\lceil\frac{a+b+1}{x+y-1}\right\rceil$

repeat  M  times
  begin
    v <-- $r^+$-EXAMPLE$^+$
    if $p_i(v)=0 \quad \forall p_i \in g$            [if g(v)=0]
      then   COUNT <-- COUNT+1
  end
d'-conflict-g-$r^+$-PE-II <-- $\left(\frac{\text{COUNT}}{M} > d'\right)$   ■
```

The dual of Algorithm 8.3, followed by its theorems, is stated next.

Algorithm 8.4.

Task : Learn f from noisy PEs and noisy NEs.

Assumptions: (a) $x_1,\ldots,x_t$ are sufficient to express f.

(b) Error rates r^+ and r^- of PEs and NEs generators are known.

Parameters : h_1, h_2, h_3, h_4, $h_5 > 1$.

```
n <-- 1                              [set n to 1]
call Algorithm 5.2
 with k<--n and r<--$r^-$            [learn n-DNF g from NEs]
if n<t                               [if g not least specific]
```

```
4.  then
5.     if d'-conflict-g-$r^+$-PE-II   [if g is over-specific]
6.       then
7.         begin
8.           n <-- n+1                [increment n]
9.           go to 2                  [and try again]
10.        end
```

■

<u>Theorem 8.10.</u> Algorithm 8.4 is applicable if and only if

$$0 < r^- < r^-_{max} = \frac{\sqrt{(eA_t+2-e)^2+4(e-1)}-(eA_t+2-e)}{2(e-1)} \quad (8.40)$$

and

$$0 < r^+ < r^+_{max} = \frac{\sqrt{e}-1}{e-1} \simeq .3775 \quad (8.41)$$

hold for the error rates r^- and r^+ of the NEs and PEs generators, respectively, and also

$$1 < h_3 < h_{3,max} = \frac{(1-r^-)(er^-+1-r^-)}{eA_t r^-}, \quad (8.42)$$

as well as

$$1 < h_1 < h_{1,max} = \frac{(1-r^+)(er^++1-r^+)}{er^+}, \quad (8.43)$$

and

$$1 < h_5 < h_2 \quad (8.44)$$

hold for the user-specified parameters h_1, h_2, h_3, and h_5.

Proof: Theorem 5.4 necessitates (8.40) and (8.42). The number, M, of PEs required per run, necessitates (8.41), (8.43), and (8.44). ■

Theorem 8.11. The output g of Algorithm 8.4 has the properties:

(a) $P\left[\sum_{g(v)=0} D^{+}(v) \geq h_1^{-1}\right] \leq h_2^{-1}$.

(b) $P\left[\sum_{g(v)\neq 0} D^{-}(v) \geq h_3^{-1}\right] \leq h_4^{-1}$.

(c) g is in n-DNF, for some $n \leq t$; g is not necessarily in maximal n-DNF.

Proof: The proofs of (a), (b), and (c), are the duals of those of (b), (a), and (c), of Theorem 8.8. ■

Thoerem 8.12. If Algorithm 8.4 is applied to learn a k-DNF concept, then

(a) P[more than k runs executed] $\leq h_5^{-1}$.

(b) If c is a constant, with

$$c < \frac{1}{e}, \tag{8.45}$$

and

$$r^{-} = \frac{c}{A_t} \quad \text{for all } A_t \tag{8.46}$$

is an error rate of the NEs generator, acceptable by Algorithm 8.4, and also

$$h_3 < \frac{1}{ec} \tag{8.47}$$

holds for the user-specified parameter h_3, then the worst-case time complexity of the algorithm is

$T(t, h_2, h_4) =$

$$= O\left(t^{n+1}\left\{\frac{h_3t^n+\ln(h_2h_4)}{1-ech_3}+\frac{\ln\left(\frac{h_2h_5}{h_2-h_5}\right)}{\ln\left(\frac{eh_1}{(er^+-r^++1)(er^++eh_1-r^++1)}\right)}\right\}\right)$$

and

$T(t, h_2, h_4) =$

$$= \Omega\left(t^{n+1}\left\{\frac{h_3t^{2n+1}+\ln(h_2h_4)}{(1-ch_3)t+c}+\frac{\ln\left(\frac{h_2h_5}{h_2-h_5}\right)}{\ln\left(\frac{eh_1}{(er^+-r^++1)(er^++eh_1-r^++1)}\right)}\right\}\right)$$

where n is the number of runs executed.

Proof:

(a) The dual of that of Theorem 8.9(a).

(b) Same as in Theorem 8.9(b). ■

Before summarizing this chapter's results, let's make one remark. Two other algorithms, learning either a CNF or a DNF, could have been easily constructed; one of them would be the combination of Algorithms 8.1 and 8.2, and the other one the combination of Algorithms 8.3 and 8.4. However, in order for those two algorithms to be applicable, **both** example genarators should have very small (inverserly proportional to A_t) error rates. That would make the algorithms not practical, and that's why they were not stated in this chapter. ■

8.6. Summary

Four algorithms, performing weak two-sided-error learning, from noisy PEs and NEs, were presented in this chapter.

Algorithm 8.1 (8.2) is applicable for any error rate of the NEs (PEs) generator, and for very small error rates of the PEs (NEs) generator. Its running time, considered as a function of t, is at most $\Theta(t^{2t+1})$. The time, also, required by its output to determine whether a vector is a PE (NE) is at most $\Theta(t^{t+1})$; determining whether a vector is a NE (PE), is NP-hard.

On the other hand, Algorithm 8.3 (8.4) is applicable when the error rate of the NEs (PEs) generator is less than .37, and when the error rates of the PEs (NEs) generator is very small. However, if the concept to be learned can be written in k-CNF (k-DNF), with k itself being unknown, then the probability that the algorithm's running time will be at most $\Theta(t^{2k+1})$, as well as the probability that the algorithm's output can determine in at most $\Theta(t^{k+1})$ time whether a vector is a PE (NE), can be bounded by user-specified parameter; determining whether a vector is a NE (PE) is still NP-hard. ■

The problem solved in this chapter completed the study, started in Chapter 6, of concept learning from PEs and NEs, in the absence of information about the concept's representability. That was the most realistic problem addressed in our thesis. Problems to be considered for future research are given in the next chapter. ■

CHAPTER 9

CONCLUSION

9.1. Introduction

The most important results of our work, derived in Chapters 2 to 8 of this thesis, are briefly sketched in Section 9.2. The places in which improvements can be made, as well as the weaknesses of some of our results are stated in Section 9.3. A few proposals for further research are also given in that section. ■

9.2. Thesis Results

L. G. Valiant's approach for approximate, instead of exact, learning was adopted in our thesis, where we studied several variations of the problem of boolean concept learning from examples.

Valiant's idea is important not because it led to algorithms which can learn an approximation of a concept (any algorithm not shown all PEs/NEs cannot be guaranteed but to learn an approximation), but because it allowed the **estimation of the closeness** of the learned approximation. Notice that this closeness cannot be measured except in terms of a "true" concept (denoted by f in our thesis), a notion which had not received much attention before Valiant's work.

Valiant was also able to find that the classes of k-CNF and k-DNF expressions are approximately learnable from error-free examples in polynomial time, provided that the value of k is known a priori. His algorithms have been made more flexible in our thesis (Chapter 4), and their time complexities have been improved by at least 20%.

Valiant was also able to show that even if the examples are noisy, the k-CNF and k-DNF expressions are still approximately learnable in polynomial time, provided that the value of k is known, and that the maximum noise, which cannot be too high, is known, as well. His learning algorithms have been modified here (Chapter 5), so that they are faster (their worst-case time complexity is $\Theta(t^{2k+1})$, compared to the $\Theta(t^{2k+1}\ln(t^k))$ of Valiant's), they are more flexible, and they are also applicable in more noisy environments.

Next, the same problem of concept learning from examples was examined, but in the absence of the knowledge of k. Our algorithms (Chapter 6), performing approximate concept learning from error-free examples, when k is not known, are in the worst case asymptotically as good as those performing the same task when k is known.

On the other hand, our algorithms (Chapters 7 and 8) performing approximate concept learning from noisy examples, when k is not known, may be in the worst case asymptotically worse than those performing the same task when k is known, but the probability of that happening can be bounded by a user-specified parameter. ■

9.3. Further Research

Several suggestions towards either the improvement of our thesis' results, or the consideration of new problems are given in this section.

First of all, it should be clear that the value of $L(h_1, h_2, k)$ is of tremendous importance for the time complexity of our algorithms. Therefore, the question of whether the upper bound of $L_0(h_1, h_2, k)$ can be tightened deserves further consideration.

An unpleasant situation was encountered towards the end of our thesis (Chapter 8). If the value of k in not known, and both example generators are noisy, then their noise has to be very low in order for our algorithms to be applicable. This is a serious problem, and it needs further investigation.

A satisfactory solution of this problem will also allow the design of practical algorithms performing <u>multiple concept learning</u> (or <u>concept discrimination</u>), that is, learning several concepts, given a set of examples for each of them. If concepts $f_1,f_2,\ldots,f_n$ are to be learned, then each concept f_i can be learned by applying our thesis' algorithms, if the examples of f_i are considered as PEs, and the examples of $f_1,\ldots,f_{i-1},f_{i+1},\ldots,f_n$ as NEs. Definitely, if all example generators are error-free, then the algorithms of Chapter 6 can be applied to perform concept discrimination. However, if more than one of the example generators are noisy, then their error rates have to be very low.

Another problem to be considered is that of boolean concept learning, when the variables expressing the concept are not known. For example, if the first PE is $E_1=xz$, the second PE is $e_2=tx'w$, etc., is it possible, and how, to learn an approximation?

Finally, it will be nice if our thesis' results can be (partially) extended to include first order predicate logic;

as already mentioned in Chapter 1, the propositional logic used is rather a weak knowledge representation. L. G. Valiant ([VALIA85]) made some attempts in this direction. ■

9.4. Summary

The first part (Chapters 4 and 5) of our thesis improved considerably Valiant's answer to the problem of concept learning from examples. The second part (Chapters 6, 7, and 8) of our thesis led to very satisfactory results, although the learning problems examined there had not been considered before. As it is always the case, there is still plenty of room for improvements. ■

REFERENCES

[BAASE78] Baase, S. Computer Algorithms: Introduction to Design and Analysis. Addison-Wesley, Reading, MA, 1978.

[BAFEI81] Barr, A., and Feigenbaum, E. A., eds. The Handbook of Artificial Intelligence, vol. I. W. Kaufmann, Los Altos, CA, 1981.

[CAMIM83] Carbonell, J. G., Michalski, R. S., and Mitchell, T. M. "An Overview of Machine Learning". In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds., Machine Learning: An Artificial Intelligence Approach. Tioga, Palo Alto, CA, 1983.

[COCKB85] Cockburn, B. F. "Inductive Concept Learning Using the Artificial Intelligence Approach". Master's Essay. Research Report CS-85-12, Department of Computer Science, University of Waterloo, May 1985.

[COFEI82] Cohen, P. R., and Feigenbaum, E. A., eds. The Handbook of Artificial Intelligence, vol. III. W. Kaufmann, Los Altos, CA, 1982.

[DEJON83] DeJong, G. "Acquiring Schemata Through Understanding and Generalizing Plans". Proceedings of the Eighth International Joint Conference on Artificial

Intelligence, Karlsruhe, W. Germany, August 1983, pp. 462-464.

[DIMIC83] Dietterich, T. G., and Michalski, R. S. "A Comparative Review of Selected Methods for Learning from Examples". In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds., Machine Learning: An Artificial Intelligence Approach. Tioga, Palo Alto, CA, 1983.

[DREYF72] Dreyfus, H. What Computers Can't Do: A Critique of Artificial Reason. Harper & Row, New York, 1972.

[FORSY84] Forsyth, R. "Machine Learning Systems". Aslib Proceedings, vol. 36, no. 5, May 1984, pp. 219-227.

[GAJOH79] Garey, M. R., and Johnson, D. S. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, San Fransisco, CA, 1979.

[HAYES84] Hayes-Roth, F. "Knowledge-Based Expert Systems". Computer, vol. 17, no. 10, October 1984, pp. 263-273.

[KARP77] Karp, R. M. "The Probabilistic Analysis of Some Combinatorial Search Problems". In J. F. Traub, ed., Algorithms and Complexity: New Directions. Academic Press, New York, 1977.

[KNUTH76] Knuth, D. E. "Big Omicron and Big Omega and Big Theta". SIGACT News, vol. 8, no. 2, June 1976, pp. 18-24.

[KOORP81] Konrad, E., Orlowska, E., and Pawlak, Z. "On Approximate Concept Learning". Technical Report 81-7, University of Berlin, October 1981.

[LACAR84] Langley, P., and Carbonell, J. G. "Approaches to Machine Learning". Journal of the American Society for Information Science, vol. 35, no. 5, September 1984, pp.306-316.

[LENAT82] Lenat, D. B. "AM: Discovery in Mathematics as Heuristic Search". In R. Davis, and D. B. Lenat, Knowledge-Based Systems in Artificial Intelligence. McGraw-Hill, New York, 1982.

[MCCOR79] McCorduck, P. Machines Who Think. W. H. Freeman, New York, 1979.

[MICHA83] Michalski, R. S. "A Theory and Methodology of Inductive Learning". In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds., Machine Learning: An Artificial Intelligence Approach. Tioga, Palo Alto, CA, 1983.

[MICAM83] Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., eds., Machine Learning: An Artificial Intelligence Approach. Tioga, Palo Alto, CA, 1983.
(Based on the First Machine Learning Workshop, Carnegie-Mellon University, July 1980).

[MICAM86] Michalski, R. S., Carbonell, J. G., and Mitchell,

T. M., eds., Machine Learning: An Artificial Intelligence Approach, vol. II. Morgan Kaufmann, Palo Alto, CA, 1986. (Based on the Second Machine Learning Workshop, University of Illinois at Urbana-Champain, June 1983).

[MICHI82] Michie, D. "The State of the Art in Machine Learning". In D. Michie, ed., Introductory Readings in Expert Systems. Gordon and Breach, New York, 1982.

[MICHI84] Michie, D. "Automating the Synthesis of Expert Knowledge". Aslib Proceedings, vol. 36, no. 9, September 1984, pp. 337-343.

[MINTO85] Minton, S. "A Game-Playing Program that Learns by Analyzing Examples". Technical Report CMU-CS-85-130, Department of Computer Science, Carnegie-Mellon University, May 1985.

[MITCH77] Mitchell, T. M. "Version Spaces: A Candidate Elimination Approach to Rule Learning". Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Cambridge, MA, August 1977, pp. 305-310.

[MITCH79] Mitchell, T. M. "An Analysis of Generalization as a Search Problem". Proceedings of the Sixth International Joint Conference on Artificial Intelligence, Tokyo, Japan, August 1979, pp. 577-582.

[MITCH80] Mitchell, T. M. "The Need for Biases in Learning Generalizations". Technical Report CBM-TR-117, Computer

Science Department, Rutgers University, May 1980.

[MITCH82] Mitchell, T. M. "Generalization as Search". Artificial Intelligence, vol. 18, no. 2, March 1982, pp. 203-226.

[MOSTO83] Mostow, D. J. "Machine Tranformation of Advice into a Heuristic Search Procedure". In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds., Machine Learning: An Artificial Intelligence Approach. Tioga, Palo Alto, CA, 1983.

[PAPOU84] Papoulis, A. Probabilities, Random Variables, and Stochastic Processes, 2nd edition, McGraw-Hill, New York, 1984.

[QUINL83] Quinlan, J. R. "Learning from Noisy Data". Proceedings of the Second International Machine Learning Workshop, University of Illinois at Urbana-Champain, June 1983, pp. 58-64.

[RABIN77] Rabin, M. O. "Probabilistic Algorithms". In J. F. Traub, ed., Algorithms and Complexity: New Directions. Academic Press, New York, 1977.

[RALST83] Ralston, A., ed. Encyclopedia of Computer Science ans Engineering, 2nd edition. Von Nostrand Reinhold, New York, 1983.

[RISSL83] Rissland, E. L. "Examples and Learning Systems". Technical Report 83-16, Department of Computer and

Information Science, University of Massachusetts at Amherst, 1983.

[ROSEN58] Rosenblatt, F. "The Perceptron, a Probabilistic Model for Information Organization and Storage in the Brain". Psychological Review, vol. 65, 1958, pp. 36[illegible]-408.

[SAMUE59] Samuel, A. L. "Some Studies in Machine Learning Using the Game of Checkers". IBM Journal of Research and Development, vol. 3, no. 3, 1959, pp. 211-229.

[SAMUE67] Samuel, A. L. "Some Studies in Machine Learning Using the Game of Checkers II - Recent Progress". IBM Journal of Research and Development, vol. 11, no. 6, 1967, pp. 601-617.

[SCHAN82] Schank, R. C. "Looking at Learning". Proceedings of the European Conference in Artificial Intelligence, Orsay, France, July 1982, pp. 11-18.

[SCOTT83] Scott, P. D. "Learning: The Construction of A Posteriori Knowledge Structures". Proceedings of the Third National Conference on Artificial Intelligence, Washington, D.C., August 1983, pp. 359-363.

[SEGEN85] Segen, J. "Learning Concept Descriptions from Examples with Errors". Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, CA, August 1985, pp. 634-636.

[SILVE83] Silver, B. Learning Equation Solving Methods from Examples". Proceedings of the Eighth International Joint Conference on Artificial Intelligence, Karlsruhe, W. Germany, August 1983, pp. 429-431.

[SIMON83] Simon, H. A. "Why Should Machines Learn?". In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds., Machine Learning: An Artificial Intelligence Approach. Tioga, Palo Alto, CA, 1983.

[SILEA74] Simon, H. A., and Lea, G. "Problem Solving and Rule Induction: A Unified View". In L. W. Gregg, ed., Knowledge and Cognition. Lawrence Erlbaum Associates, Potomac, Maryland, 1974.

[TURIN50] Turing, A. "Computing Machinery and Intelligence", 1950. In E. Feigenbaum, and J. Feldman, eds., Computers and Thought. McGraw-Hill, New York, 1963.

[UTGOF83] Utgoff, P. E. "Adjusting Bias in Concept Learning". Proceedings of the Eighth International Joint Conference on Artificial Intelligence, Karlsruhe, W. Germany, August 1983, pp. 447-449.
Also in Proceedings of the Second International Machine Learning Workshop, University of Illinois at Urbana-Champain, June 1983, pp. 105-109.

[UTMIT82] Utgoff, P. E., and Mitchell, T. M. "Acquisition of Appropriate Bias for Inductive Concept Learning". Proceedings of the Second National Conference on Artificial Intelligence, Pittsburgh, Pensylvania, August 1982, pp. 414-417.

[UTNUD83] Utgoff, P. E., and Nudel, B. "Comprehensive Bibliography of Machine Learning". In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds., Machine Learning: An Artificial Intelligence Approach. Tioga, Palo Alto, CA, 1983.

[VALIA84a] Valiant, L. G. "A Theory of the Learnable". Communications of the ACM, vol. 27, no. 11, November 1984, pp. 1134-1142.
Also, in Proceedings of the Sixteenth Annual ACM Symposium on the Theory of Computing, Washington, D.C., May 1984, pp. 436-444.

[VALIA84b] Valiant, L.G. "Deductive Learning". Philosophical Transactions of the Royal Society of London, series A, vol. 312, December 1984, pp. 441-446.

[VALIA85] Valiant, L. G. "Learning Disjunctions of Conjuctions". Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, CA, August 1985, pp. 560-566.

[WINST70] Winston, P. H. "Learning Structural Descriptions from Examples", 1970. In P. H. Winston, ed., The Psychology of Computer Vision. McGraw-Hill, New York, 1975.

[WINST80] Winston, P. H. "Learning and Reasoning by Analogy". Communications of the ACM, vol. 23, no. 12, December 1980, pp. 689-703.

[WINST84] Winston, P. H. Artificial Intelligence, 2nd edition. Addison-Wesley, Reading, MA, 1984.

[YAMAX84] Yaghmai, N. S., and Maxin, J. A. "Expert Systems: A Tutorial". Journal of the American Society for Information Science, vol. 35, no. 5, September 1984, pp. 297-305.