

AD-A171 794

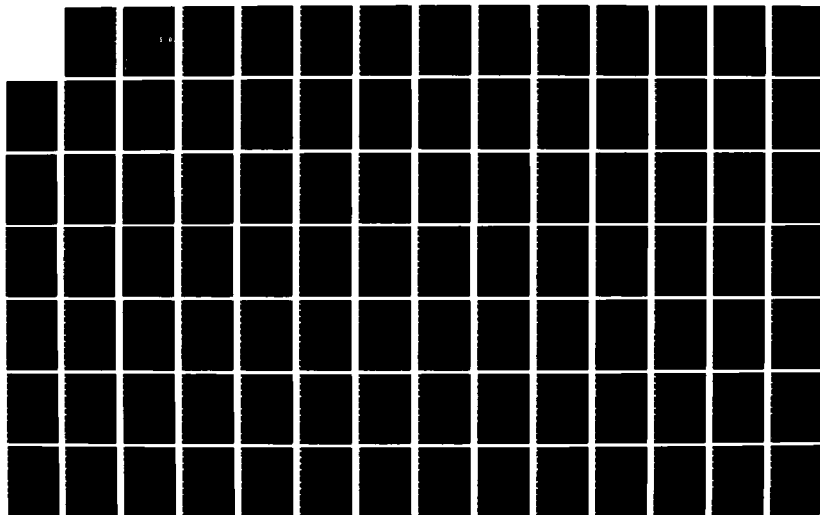
LEARNING OBJECT-LEVEL AND META-LEVEL KNOWLEDGE IN
EXPERT SYSTEMS(U) STANFORD UNIV CA DEPT OF COMPUTER
SCIENCE L FI NOV 85 STAN-CS-86-1091 N00039-83-C-0136

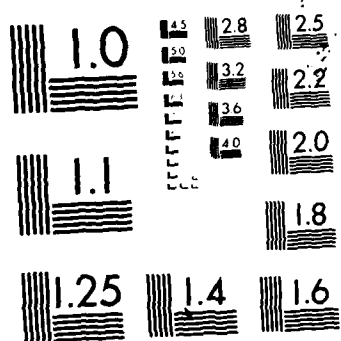
1/3

UNCLASSIFIED

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

November 1985

Report No. STAN-CS-86-1091
Also numbered KSL-85-44



AD-A171 794

Learning Object-Level and Meta-Level Knowledge in Expert Systems

by

Li-Min Fu

Contract N00039-83-C-0136

DTIC
ELECTE
SEP 09 1986
S D

Department of Computer Science

Stanford University
Stanford, CA 94305

ALL INFORMATION CONTAINED
HEREIN IS UNCLASSIFIED



DTIC FILE COPY

86 9 08 109

November 1985

Report No. STAN-CS-86-1091
Also numbered KSL-85-44

Learning Object-Level and Meta-Level Knowledge in Expert Systems

by

Li-Min Fu

Contract N00039-83-C-0136

Department of Computer Science

Stanford University
Stanford, CA 94305



86 9 08 10

**Knowledge Systems Laboratory
Report No. KSL-85-44**

November 1985

LEARNING OBJECT-LEVEL AND META-LEVEL KNOWLEDGE IN EXPERT SYSTEMS

**A Dissertation
Submitted to the Department of Electrical Engineering
And the Committee on Graduate Studies
Of Stanford University**

LI-MIN FU

**Department of Computer Science
Stanford University
Stanford, CA 94305**

Abstract

A high performance expert system can be built by exploiting machine learning techniques. A learning model has been designed and implemented that is capable of constructing a knowledge base, in the form of rules, from a case library and continuously updating it to accommodate new facts. This model is designed primarily for EMYCIN-like systems in which there is uncertainty about data as well as about the strength of inference and in which the rules chain together to infer complex hypotheses. These features greatly complicate the learning problem.

In machine learning, two issues that cannot be overlooked practically are efficiency and noise. A subprogram, called "CONDENSER", is designed to remove irrelevant features during learning and improve the efficiency. The noise can be handled by optimizing the result to achieve minimal prediction errors.

Another subprogram has been developed to learn meta-level rules which guide the invocation of object-level rules and thus enhance the performance of the expert system using the object-level rules.

Using the ideas developed in this work, an expert program called JAUNDICE has been built, which can diagnose the likely disease and mechanisms of a patient with jaundice. Experiments with JAUNDICE show the developed theory and method of learning are effective in a complex and noisy environment where data may be inconsistent, incomplete, and erroneous.

1v



Accession For	
NTIS CRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Acknowledgments

I am fortunate to get acquainted with Bruce G. Buchanan, who is such a great philosopher, teacher, and master in both expert systems and knowledge acquisition. Many ideas in this thesis are, consciously or subconsciously, spawned and matured by constant inspiring from him; I can't even remember exactly when those ideas emerge. He also provides me strong psychological support for my devoting to this work. No doubt my first thank goes to him, Bruce G. Buchanan, a judicious and respectable man. I thank Susan S. Owicki for her feedback from another angle and her enthusiasm for my work. In addition, I thank Oscar Buneman, Richard H. Pantell, and Edison T. S. Tse, the other members of my reading or oral committee, for their commitment to my work. I would like thank Robert L. White who is the first man I can consult in Stanford University. I also appreciate the help from Gabriel Garcia and Peter B. Gregory for providing us with liver biopsy cases and the help from John Haggerty for providing information of the REFEREE system; both of their help is valuable to validate our program. I thank my family particularly Minyuen, my wife, who seldom complains about my being monopolized by computer and gives me encouragement and advice when I need them. I thank as well SUMEX-AIM, Heuristic Programming Project, Electrical Engineering department, Stanford University, and Scribe facilities for generating this text. This work is in part supported by the Advanced Research Project Agency under the contract DARPA N00039-83-C-0136 and National Institute Health under the grant NIH RR-00785-11.

Table of Contents

1. Introduction	1
1.1. Two Main Problems	3
1.1.1. Accuracy of Performance	3
1.1.2. Efficiency of Performance	4
1.2. Learning in Expert Systems	5
1.2.1. Learning from Examples	5
1.2.2. Efficiency Consideration	8
1.2.3. Noisy Learning Environments	9
1.3. Learning as an Approach to Debugging the Knowledge Base	9
1.4. Meta-Level Knowledge	10
1.4.1. The Role of Meta-Rules	10
1.4.2. Rule Model and Function Template in Learning	11
1.4.3. Machine Learning of Meta-Level Knowledge	12
1.5. JAUNDICE as a Set of Experimental Programs	12
1.5.1. The Performance Program	14
1.5.2. Knowledge Base	15
1.5.3. Database	16
1.5.4. The RL Program	17
1.5.5. The Debugging Program	18
1.5.6. Meta-RULEGEN	18
1.6. Contribution of the Thesis	18
1.7. Outline of the Thesis	19
2. Knowledge Acquisition via Machine Learning	21
2.1. Introduction	21
2.1.1. Learning in JAUNDICE	25
2.2. Learning via Search from the Most General Hypothesis	29
2.2.1. Procedure	31
2.2.1.1. One Example	35
2.2.2. Areas of Application	37
2.2.3. Comparison and Discussion	39

2.2.3.1. Comparison with Related Works	39
2.2.3.2. Model-Driven vs. Data-Driven Learning	42
2.2.4. Focusing Mode of Learning	44
2.2.4.1. Procedures	45
2.3. Learning Intermediate Knowledge	48
2.3.1. Intermediate Concepts in the Initial Vocabulary	50
2.3.1.1. Bottom-Up Learning	51
2.3.1.2. Top-Down Learning	55
2.3.1.3. Bidirectional Extension Strategy	56
2.3.2. Intermediate Concepts not in the Initial Vocabulary	58
2.3.2.1. Technique of Symbolizing Taxonomy Point	58
2.3.2.2. Technique of Symbolizing Switchover Point	62
2.3.3. Comparison and Discussion	65
2.4. Learning Disconfirming Rules	66
2.5. Constructing a Hierarchical Knowledge Base	69
3. Feature Condensation	71
3.1. Introduction	71
3.2. The Learning System	74
3.2.1. Structure and Behavior	74
3.2.2. CONDENSER	76
3.3. The Rule of Condensation	77
3.4. Techniques of Condensation	79
3.4.1. Ordered Scanning Algorithm	81
3.5. Why Does CONDENSER Work?	85
3.6. Application	90
3.6.1. Applicable Domains	90
3.6.2. Compatible Learning Systems	92
3.7. Comparison and Discussion	93
3.8. Summary	95
4. Learning in Noisy Environments	96
4.1. Introduction	96
4.2. Imperfect Training Instances	99
4.2.1. Inconsistency of Training Instances	99
4.2.1.1. Spontaneous Inconsistency	99
4.2.1.2. Incorrectly Classified Training Instances	101
4.2.2. Inadequacy of Training Instances	103
4.2.2.1. Incompleteness of Data	103

4.2.2.2. Sampling Insufficiency	104
4.2.2.3. Unreliability and Inconsistency of Data	106
4.3. Imperfect Learning Systems	106
4.3.1. Insufficiency of the Descriptive Language	106
4.3.2. Insufficiency of Rules of Generalization or Specialization	108
4.3.3. Procedural Bias	108
4.3.4. Representational Bias	110
4.4. Error Measurement	111
4.4.1. Calibration Error	112
4.4.2. Prediction Error	113
4.5. Error Handling	115
4.5.1. Pre-Filter	117
4.5.2. Mid-Filter	118
4.5.3. Post-Filter: Optimizer	119
4.5.3.1. Minimal Error Principle	119
4.5.3.2. Procedures	121
4.6. One Example	123
4.7. Comparison and Discussion	124
4.8. Summary	127
5. Automated Knowledge Base Updating	128
5.1. Introduction	128
5.2. Faults in the Knowledge Base	131
5.2.1. In Domains without Uncertainty	132
5.2.1.1. Overly Generalized Rules	132
5.2.1.2. Overly Specialized Rules	132
5.2.1.3. Erroneous Rules	133
5.2.1.4. Missing Rules	134
5.2.1.5. Subsumption	134
5.2.1.6. Redundancy	134
5.2.2. In Domains with Uncertainty	134
5.2.2.1. Overly Generalized Rules	134
5.2.2.2. Overly Specialized Rules	135
5.2.2.3. Erroneous Rules (or Erroneous Degree of Certainty)	135
5.2.2.4. Missing Rules	136
5.2.2.5. Subsumption	136
5.2.2.6. Redundancy	137
5.3. Fault Corrections	137

5.4. Automated Debugging	140
5.4.1. Fault Analysis	142
5.4.2. Application of Machine Learning	145
5.4.3. Retrospective Inspection after Learning	146
5.4.3.1. Experimentations	147
5.4.3.2. Verifications	149
5.4.4. One Example	150
5.5. Comparison and Discussion	151
5.6. Summary	154
6. Discovery of Meta-Rules	156
6.1. Introduction	156
6.2. Learning Meta-Rules: Design Considerations	157
6.2.1. Format of Meta-Rules	157
6.2.2. Utility Consideration of Meta-Rules	159
6.2.2.1. Utility Value for Meta-Rules	159
6.2.2.2. Selecting Useful Meta-Rules	163
6.2.3. Overview of Two Approaches to Learning Meta-Rules	164
6.2.3.1. From Object Rules	164
6.2.3.2. From Attributes	164
6.3. Implementation	165
6.3.1. Overview of META-RULEGEN	165
6.3.2. Algorithm	167
6.3.2.1. Approach from Object Rules	168
6.3.2.2. Approach from Attributes	171
6.4. Results	172
6.5. Conclusion	175
7. Results and Conclusions	178
7.1. Results of Learning in JAUNDICE	179
7.2. A Sample Dialogue of Interactive Mode in JAUNDICE	183
7.2.1. Gathering Information	185
7.2.2. Providing Interpretations	188
7.2.3. Explanation	190
7.2.4. Asking for the Expert's Diagnosis	191
7.2.5. Debugging the Knowledge Base	192
7.3. Validation	195
7.3.1. Rediscovery of Well-Known Concepts	196
7.3.2. Testing Generality in the Same Domain	196

7.3.3. Testing Generality in Other Domains	196
7.4. Assumptions and Discussions	198
7.4.1. Basic Assumptions	199
7.4.2. Requirement of Domain-dependent Knowledge and Heuristics	199
7.4.3. Case Selection	200
7.4.4. Domain-Dependent Rules of Generalization or Specialization	200
7.4.5. Representational Adequacy	201
7.4.6. Rule Redundancy	202
7.5. Comparison with Related Work	202
7.6. Future Extensions	203
7.6.1. An Expert System with the Ability of Discussion	203
7.6.2. Unsupervised Learning	204
7.6.3. Training Instances with Multiple Classifications	205
7.7. Conclusion	205
Appendix A. Degree of Certainty	206

Chapter 1

Introduction

Artificial Intelligence (AI) techniques have been employed in designing expert systems¹ which solve problems in specific domains by means of expertise encoded in knowledge bases. Automated consultation programs (expert systems) have been built in the area of inferring chemical molecular structure from mass spectra [Buchanan 69] mathematics [Martin 71], mineral exploration [Duda 78], diagnosing medical disease and giving therapy (e.g., [Shortliffe 76], [Kulikowski 82], [Miller, Pople, and Meyers 82]), and so forth. Why do we need expert systems? Solving complex problems (which may even daunt experts) and saving human resources are major motivations. For instance, in medicine, because of the uneven distribution of medical personnel, expert systems can raise the average quality of medical care, particularly in rural areas. And medical expert systems have demonstrated better diagnostic accuracy than junior physicians [Buchanan and Shortliffe 84]. Moreover, the medical cost can be reduced by, for instance, allowing nurse practitioners to prescribe under the guidance of expert systems. In another stream, AI researchers have begun to investigate "machine learning" for the purposes of economizing the time and energy of knowledge acquisition, discovery of new concepts, and understanding of human cognitive mechanisms.

¹Expert systems differ from classical decisional analysis in several aspects: first, the expertise is encoded into many symbolic rules which simulate the experts' thought; second, the representation is emphasized on its understandability; third, the system should have intelligent features, such as the ability of explanation.

This thesis, developed under the confluence of the impact of "expert system" and the recent enthusiasm about "machine learning",² is motivated by the following considerations:

- The acquisition, representation, and use of knowledge are the key issues in building expert systems [Hayes-Roth 83]. Among these, the most difficult issue is knowledge acquisition. Even expert knowledge may often be ill-defined or imprecise, and disagreement may exist among experts. Moreover, in a relatively unexplored domain, expert knowledge is incomplete. Machine learning can lend itself to these situations;³ however, it is still in its incipient stage, and any work in this area suffers from weaknesses or limitations in some aspect.⁴ Finding a more general and better solution to knowledge acquisition by machine learning is the first motivation of this work.
- In inductive concept learning, there are still so many issues remaining to be solved; for example, how to learn if uncertainty is involved, how to learn new concept descriptors, how to learn efficiently, how to handle noisy data, and so on. Finding some solutions to these issues is the second motivation.
- An expert system will be more intelligent in solving problems if endowed with the ability to learn meta-rules that control and guide the invocation of object-level (domain) rules. This is the third motivation of this thesis.

In this work, we develop theories and methods of building an intelligent and robust expert system that can perform efficiently and accurately, and can improve its

²"Learning" here denotes active learning rather than passive learning (e.g., learning by being told). For example, TEIRESIAS [Davis 79] is a passive learning program; the SEEK program [Politakis 82] is more than a passive program but still cannot find missing rules; Meta-DENDRAL [Buchanan 78a] is an active program.

³Inductive concept learning techniques have been applied to construct a knowledge base for expert systems, e.g., Meta-DENDRAL [Buchanan 78a], and RULEMASTER [Michie 84].

⁴This will be analyzed in Chapter 2.

performance through learning. We primarily focus on EMYCIN-like⁵ systems where the learning task is complicated by the facts that reasoning involves complex interactions among rules, and involves uncertainty about the data and the strength of inference.

1.1. Two Main Problems

The recognition of expert systems as substitutes for or complements to human experts requires that expert systems should perform accurately and efficiently. Therefore, we focus on these two issues: accuracy and efficiency.

1.1.1. Accuracy of Performance

By "accuracy", we mean the advice given by the system should be able to solve the problems concerned. For example, the therapy recommended by a medical expert system can relieve the patient's discomfort. Practically, we often use a comparison between a program's advice and an expert's to evaluate the accuracy of the system [Buchanan and Shortliffe 84].

The problem considered here comprises two stages:

- The first stage problem is defined as "constructing an accurate knowledge base (KB) from a given case library".

Given: A case library.

Find: A set of rules that can diagnose each case correctly.

⁵EMYCIN [Van Melle 80] is a domain-independent expert system building tool, based on the core of MYCIN.

- Since the KB may not be perfect once constructed, the second stage ensues:

Given: A faulty conclusion (or misdiagnosis)
from the performance program.

Find: Improvements to the KB in order to achieve a
correct conclusion (or diagnosis).

In an expert system, because the reasoning often involves multiple (hundreds of) rules (structured in multiple levels) which may be chained together, or either positively or negatively interact with one another, tracking down the faults in the KB is generally a difficult task. Even more difficultly, in a domain with uncertainty, there are few exact solutions. There are two approaches to debugging the KB: human-oriented and machine-oriented approaches. As mentioned earlier, expert knowledge may be unorganized, inconsistent, incomplete, or redundant. Thus it is worthwhile to explore machine-oriented learning, which is one of the major topics in this thesis.

1.1.2. Efficiency of Performance

By "efficiency", we mean the system can generate advice in a "reasonable" time without loss of quality. Though this may be a trivial issue in a small KB, it can't be ignored in a large KB where exhaustive invocation of all rules may make the system impractical owing to a great deal time consumed. Thus, this is also one of the important criteria for evaluating an expert system.

Dynamic mobilization of appropriate knowledge under a certain circumstance is a matter of "control". As discussed in some works, such as [Davis 80], [Aiello 83], control strategies are crucial in expert systems for the reason of efficiency. Thus we focus on learning good control knowledge, in the form of meta-rules, as follows:

Given: A set of object-rules.

Find: Meta-rules that can guide effectively
the invocation of object-rules.

Again, this problem can be solved by human-oriented or machine-oriented approaches. Though human experts are good at the domain specific knowledge, it may be awkward for them to write down good control knowledge. "Good" control knowledge is able to enhance efficiency in a given organization of the KB. In this thesis, a theoretical basis and methods are developed to generate a set of useful meta-rules.

1.2. Learning in Expert Systems

In this work, we use the technique known as "learning from examples" or "inductive concept learning" as the primary strategy, though we also augment its capability such that it can learn new concepts (newly defined symbols) which are not embedded in the initial given language (see Section 2.3).

1.2.1. Learning from Examples

This learning task is formulated as follows:

Given: A set of training instances.

Find: Concept descriptions that are consistent
with the training instances.

Training instances are classified⁶ into positive instances (examples of the concept) and negative instances (counter-examples of the concept). The objective of learning is to find

⁶We assume that instances are pre-classified with a high degree of accuracy. As discussed in Chapter 4, however, complete accuracy is not necessary.

concept descriptions that satisfy the positive instances but exclude the negative instances.

We may view this learning problem from an instance diagram shown in figure 1.1.

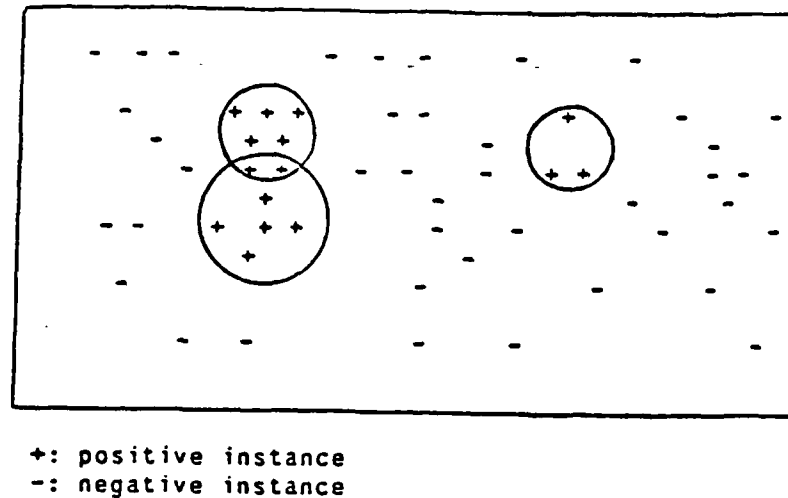


Figure 1.1 In this instance diagram, each boundary represents a concept description or a rule that includes positive instances and excludes negative instances.

Thus, the objective of learning is to delineate the boundaries between the positive and negative instances. Basically, there are two operators used in making induction from instances: generalization and specialization. Generalization broadens the scope of descriptions or enlarges the boundary to cover more instances. In contrast, specialization narrows the scope of descriptions or contracts the boundary to cover a less number of instances. If we see two different positive instances with different descriptions, we can abstract a description which covers both instances by finding a common generalization of these two instances. For example, from two positive instances "2" and "4" for the concept "even", we may abstract a more general description " $2n$, n is an integer". By properly

applying these two operators, descriptions that are general enough (by generalization operator) to cover positive instances while specific enough (by specialization operator) to exclude negative instances can be found.

Works dealing with inductive concept learning include, [Winston 70], [Vere 75], [Hayes-roth 76], [Buchanan 78a], [Mitchell 78], and [Michalski 83a]. The learning method described in this thesis is designed in a way such that it can,

1. Construct a knowledge base in an expert system, and the knowledge base will incrementally be updated to accommodate cases with faulty conclusions made by the system.
2. Discover intermediate knowledge. Intermediate knowledge denotes the deep-level instead of superficial-level reasoning knowledge. For example, in medicine, the analysis or diagnosis of a disease often involves the reasoning along the dimensions of pathophysiology, anatomy, and etiology (deep-level reasoning), rather than simply associates the clinical picture with a disease (superficial-level reasoning). In a reasoning network, intermediate concepts denote those intermediate nodes in reasoning chains.
3. Handle uncertainty.
4. Handle noisy data.
5. Learn fast.

In medicine, considering cost and risk, a rule should,

1. Have minimal features (avoid unnecessary features) to save cost and avoid unnecessary risk.
2. Be maximally specific to avoid false positive diagnoses.

In Chapter 2, a model-driven type learning method, which performs a heuristic search from the most general hypothesis, is developed to learn multiple rules from a case library.

It is an often encountered situation in an expert system that wrong advice or misdiagnosis is provided by the system. The faulty advice may be traced back to faults in the knowledge base. Since, for instance, in medicine, a variety of manifestations can occur in one disease, different cases (patients) with the same disease may be diagnosed by different rules; a misdiagnosis indicates the KB is inadequate to detect a certain combination of clinical features. Hence, learning is useful to find the right rules. "Focusing mode"(described in Section 2.2.4) is designed to discover rules covering a specified case (usually a misdiagnosed one).

1.2.2. Efficiency Consideration

Learning becomes a complicated issue in a complex domain like medicine where there may be hundreds (even thousands) of features (symptoms, signs, and laboratory tests). The difficulty is reflected by the fact that medical experts abstract a limited number of medical rules from decades of practice. Our ambitious goal is to build an expert system with fast learning ability. The important idea of data compression in information theory can lend itself admirably to handling a large volume of data in learning.⁷ As long as the desired information content is preserved, the representation can be as simple as possible. A feature condensation technique, that removes unnecessary or irrelevant features dynamically during learning, is implemented and described in Chapter 3. With this technique, learning is much faster because of the reduction of the dimensions of the search space while the quality of the output is preserved.

⁷ [Quinlan 79] discusses an alternative for learning with large data bases, but the emphasis there is on the number of examples not on the number of features in each example.

1.2.3. Noisy Learning Environments

With respect to learning, there are various types of error-sources, which may be associated with the input (the set of training instances) or the learning system (the learner). Inadequacy and bias of the learning environment are two major sources of errors. For example, a small case library renders the learned rules limited or incorrect in predicting the cases outside the case library; false positive or negative instances make the learned rules inconsistent. Chapter 4 describes possible types of error-sources and the method of their elimination.

1.3. Learning as an Approach to Debugging the Knowledge Base

As mentioned earlier, a misdiagnosis indicates faults in the KB, which may be incorrect rules or missing rules. This problem is closely related to the so-called "credit and blame assignment problem" (refer to [Dietterich and Buchanan 81]). Only those rules that are determined to be "blamed" should be corrected.

Automated debugging of the KB in a complex rule-based expert system is generally difficult because of the following reasons:

1. There are so many rules involved.
2. There are so many ways to correct a rule.
3. There may be more than one fault.
4. If uncertainty is involved, there may actually not exist any perfect solution.

TEIRESIAS [Davis 79] assists human experts in editing the KB by tracking down the relevant rules and allowing them to correct the faulty rules or add missing rules on the

basis of their knowledge and intuition. But, even experts may go astray if the faults are multiple and the solutions are not exact (i.e., optimization is required). Moreover, misdiagnosed cases are often due to missing rules. Therefore, we would rather view this problem as a learning problem. A strategy called "retrospective inspection after learning" is described in Chapter 5. With this strategy, rules that can make the misdiagnosed case diagnosed correctly are first found; then the found rules are compared with the old rules in the KB to detect missing rules or decide how rules should be generalized or specialized. This approach is more advantageous than the one which tries to modify the KB in every possible way, especially if the faults are due to missing rules.

1.4. Meta-Level Knowledge

Meta-level knowledge is the "knowledge about knowledge". So, meta-rules are rules about rules. Three types of meta-level knowledge are briefly introduced in this section: meta-rule, rule model, function template.⁸

1.4.1. The Role of Meta-Rules

Meta-rules guide the invocation of object-rules effectively by reordering or pruning them [Davis 76]. The syntactical structure of a meta-rule is seen in Chapter 6. The main reason for incorporating meta-rules is to increase the speed of performance without degrading the quality of advice. Meta-rules are particularly important in a large KB where exhaustive use of the KB will make the system awkwardly slow and, thus, impractical. In a goal-oriented reasoning system, to pursue a different goal will trigger a different set of meta-rules.

⁸ Refer to [Davis and Buchanan 77].

1.4.2. Rule Model and Function Template in Learning

A rule model summarizes the premise from a set of rules with the same area of conclusion in the KB [Davis and Buchanan 77]. The application of rule models in machine learning has three advantages:

1. They provide the knowledge about parameters (attributes) used for a certain context.
2. They provide the knowledge about the predicates commonly used for a certain attribute.
3. They provide the knowledge about the ordering of attributes in the premise and the knowledge of the necessary components for concluding a certain fact.

We may view the use of rule models in learning as "guiding future behavior by past experience", which is a key factor to make learning more semantically meaningful and of higher quality.

Function templates record the argument format for certain predicates [Davis and Buchanan 77] and are important in generating code for new rules.

As described in Chapter 2, the knowledge embedded in rule models and function templates can be formulated directly by experts as the initial knowledge for the learning system to construct a KB from nil.

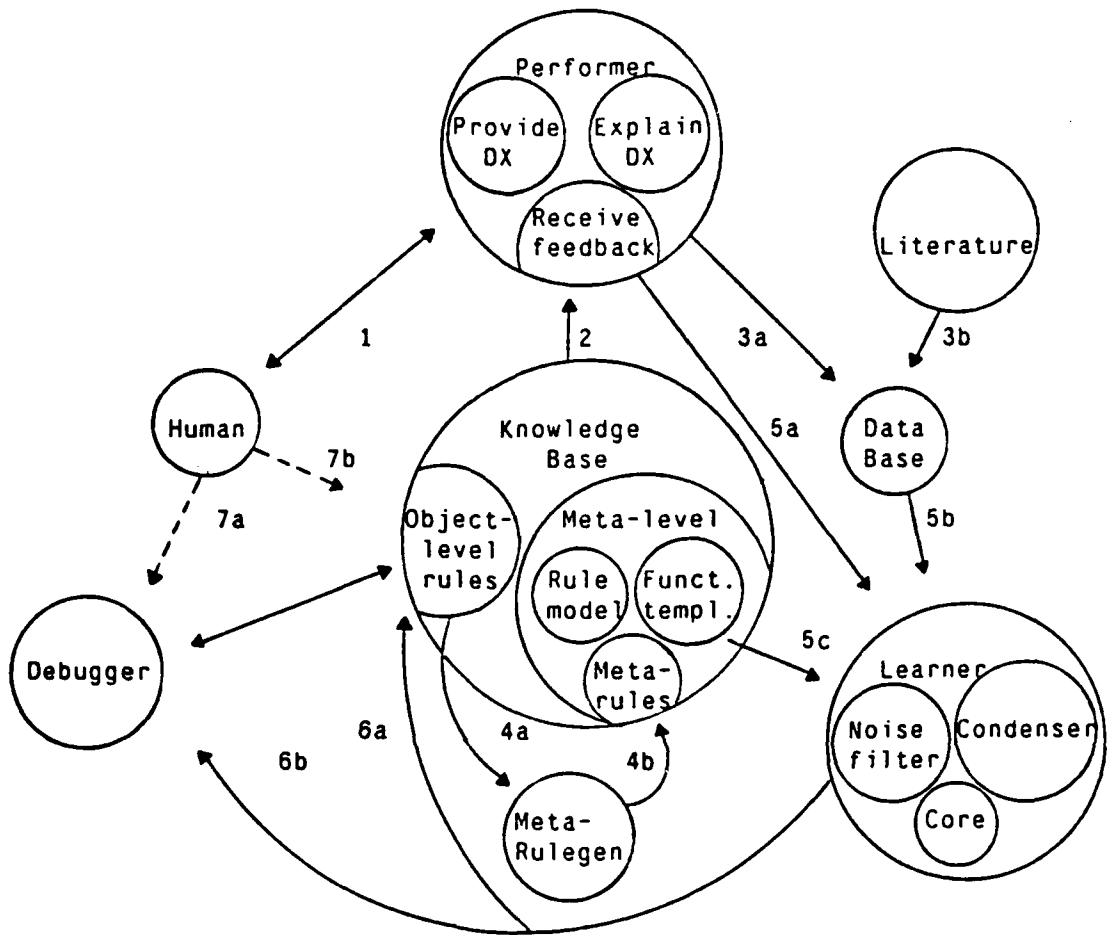
1.4.3. Machine Learning of Meta-Level Knowledge

In a hierarchical knowledge structure, as object-level knowledge is derived from observation of objects, so meta-level knowledge should be able to be derived from object-level knowledge. This is one motivation of learning meta-rules from a set of object-rules, which is described in Chapter 6. *Hierarchical learning* (learning is done level by level) is significant in constructing a multi-level knowledge structure, in which the knowledge of a higher level is abstracted from the lower levels. Learning of meta-rules from object-rules is one example of hierarchical learning.

1.5. JAUNDICE as a Set of Experimental Programs

Medicine, because of its complexity, is a good area to build expert systems. In this thesis, we use the diagnosis of jaundice as an experimental domain for the following reasons. First, since the reasoning for jaundice cases involves several different dimensions, such as pathophysiological, anatomical, and etiological reasoning, this domain is sufficiently complex to test the capability of a learning system. Second, the knowledge in this domain can be easily codified because this domain is well-studied and relatively small in contrast to the KB of INTERNIST [Miller, Pople, and Meyers 82]. JAUNDICE⁹ is an expert program which provides diagnosis for jaundice cases and learns new rules; however, this program is designed to screen the jaundice (adult) patients based on the preliminary clinical data without resorting to invasive tests, such as liver biopsy. Several features of this program are shown in figure 1.2. As seen in the above figure, the main program JAUNDICE includes several subprograms: the performance program (the

⁹It is implemented on SUMEX-AIM, TOPS-20 system and written in INTERLISP.



- 1: Interaction between human and the performance system.
- 2: Knowledge base is used in consultation.
- 3: Sources of case information for data base:
 - 3a: From consultations.
 - 3b: From literature.
- 4a 4b: Meta-rules formed from object-rules by Meta-Rulegen.
- 5: Sources of learning new knowledge:
 - 5a: From human (experts) feedback.
 - 5b: From data base
 - 5c: From meta-knowledge.
- 6a: Automatic knowledge base construction.
- 6b: Automatic knowledge base debugging.
- 7a 7b: Human-centered KB editor.

Figure 1.2 Overview of the JAUNDICE programs. Arrows indicate knowledge or data flow. Dotted links (7a, 7b) are not stressed in this work.

performer), RL (a domain independent rule learning program), the debugging program (the debugger), and Meta-RULEGEN (a domain independent program that generates meta-rules). Each subprogram will be briefly described in the following subsections. The KB stores both object-level and meta-level knowledge. The DB is a case library which stores patient data. A feature base that has 81 clinical features (binary or multiple values) is used to write rules in the KB and describe patient data in the DB. Feature values may be numerical or non-numerical. Figure 1.3 shows some examples.

<u>Feature:</u>	<u>Expected value:</u>
• Malaise	Yes, No
• GOT	positive number
• Hepatomegaly	Mild, Moderate, Marked

Figure 1.3 Examples of feature values in JAUNDICE.

1.5.1. The Performance Program

This is mainly an interactive program,¹⁰ built from scratch but similar to MYCIN¹¹. The user enters patient data; then the program provides an interpretation which includes disease diagnosis, anatomical and pathological mechanisms. An explanation of the given interpretation, based on the patient data, will be given if requested. If the user is an expert, he is allowed to give his view about the diagnosis. If the expert's conclusion is different from the system's conclusion, the expert may either edit the KB (as in

¹⁰Though, there is another version, batch mode, which can run multiple cases in a time and is designed primarily to test the performance of the program when the KB is edited.

¹¹The syntax of rules is the same; the interpreter is also goal-oriented.

TEIRESIAS [Davis 79])¹² or simply give his diagnosis without editing the KB. In the latter case, the system will debug the KB automatically (this is one of the main topics in JAUNDICE).

The performance program uses the KB to solve problems by goal-oriented (backward-chaining) reasoning. The main goal is to conclude the disease entity causing jaundice, which may be one or more of ten disease categories, including acute hepatitis, chronic hepatitis, cirrhosis of liver, primary biliary cirrhosis, and so forth.¹³ Subgoals are to conclude disease mechanisms, such as pathological and anatomical mechanisms. The program uses a measure of "degree of certainty" to handle uncertainty, which is extended from MYCIN's CF model [Buchanan and Shortliffe 84]. (See more detail of "degree of certainty" in appendix A.) The accumulation of uncertainty in JAUNDICE is the same as in MYCIN. The performance program will not be discussed further since it is not the main focus in this work, but it is used to measure the ability of the learning program to learn useful new rules. An example of consultation is shown in Chapter 7.

1.5.2. Knowledge Base

There are 141 diagnostic rules, and 80 non-diagnostic rules reflecting causal and taxonomic knowledge in the initial KB. The knowledge base was constructed by encoding knowledge from medical textbooks and journals (e.g., [Schiff 46], [Petersdorf 83], [Krupp 82], [Winkelman 81], etc.) and talking with experts. After learning, the number of

¹²This is not implemented in JAUNDICE.

¹³For simplicity, diseases are generally assumed to be mutually exclusive.

diagnostic rules increases to 304. Each rule is put in EMYCIN [Van Melle 80] rule format.

Figure 1.4 shows one example.

```
Rule62: If 1. Serum total bilirubin is greater than 1.2 mg/dl.  
          2. Serum GOT is greater than 300 I.U./dl.  
          3. Serum GPT is greater than 300 I.U./dl.  
          4. Serum Alkaline-phosphatase is less than 10 B.U.  
  
          then it is probable (.8) that the mechanism is  
              hepatocellular injury.  
  
(RULE62  (($AND (GREATER SERUM TOTAL-BILIRUBIN 1.2)  
                (GREATER SERUM GOT 300)  
                (GREATER SERUM GPT 300)  
                (LESS SERUM ALKALINE-PHOSPHATASE 10))  
          (JAUNDICE MECHANISM PARENCHYMAL-DYSFUNCTION .8)))
```

Figure 1.4 One example of a rule in JAUNDICE.

1.5.3. Database

Here, the database is the collection of patient data. In our experimental model, we carefully¹⁴ collected 72 jaundice cases from the literature (e.g., [Malchow-moller 81], [Stern 75], [Winkelman 81], etc.) to construct a case library as the starting point. Each case is stored as a frame. Figure 1.5 shows an example. The data description is represented as a feature set (or list), a set of feature value pairs.

¹⁴We only select those cases with rather complete data descriptions and confirmed diagnoses.


```
(CASE1  (DATA  ( (AGE 56)
                  (SEX MALE)
                  (ONSET ABRUPT)
                  (TOTAL-BILIRUBIN 3)
                  (GOT 150)
                  (GPT 180)
                  (URINE-UROBILINOGEN ELEVATED)
                  (MALAISE YES)
                  ...
                  ... )) )
        (COMPLICATION NIL)
        (FINAL-DX ACUTE-HEPATITIS))
```

Figure 1.5 Frame representation of cases in JAUNDICE.

1.5.4. The RL Program

The RL program (standing for rule learning program) has three subprograms: the core, CONDENSER, and the noise filter. The core of the RL program comprises three types of learning methods: "search from the most general hypothesis" method, the method of learning intermediate knowledge, and the method of learning disconfirming knowledge. CONDENSER improves the efficiency of learning by removing irrelevant features dynamically during learning. The noise filter makes the results of learning more precise by minimizing errors caused by the undesirable perturbation factors associated with the input or the system. The RL program receives input from database, expert diagnosis, and meta-level knowledge (rule model and function template). The output of the RL program (new rules) is sent to the KB directly if the purpose is to construct a new KB, or sent to the debugging program for debugging the KB if there has already been a KB (constructed some time ago).

The RL program is designed to be domain independent; as will be shown in Chapter 7, it is effective in JAUNDICE (a medical domain) as well as REFEREE (a non-medical domain).

1.5.5. The Debugging Program

Each time a misdiagnosis occurs, the learning system will be triggered to learn rules (ignoring temporarily the old rules in the KB) such that a correct diagnosis can be reached. Then, the debugging system compares the newly learned rules with the old rules to pinpoint the possible faults in the old rules. For instance, an old rule may be indicated to be too specific and should be generalized.

1.5.6. Meta-RULEGEN

Meta-RULEGEN is a second order learning program, which can generate useful meta-rules from a set of object-rules and can control the invocation of the object-rules in order to enhance the performance of expert systems.

1.6. Contribution of the Thesis

The major contributions of this thesis include the following:

1. It develops the first system which can learn new knowledge in EMYCIN-like environments where uncertainty is involved and evidence can be combined either positively or negatively. It confirms the value of the model-driven learning strategy in constructing the knowledge base with multiple decision rules in an expert system; and it differs from previous model-driven methods in its ability to learn incrementally.
2. The idea of learning intermediate concepts from a set of training instances which are not described by any intermediate concept is novel; the techniques, such as bi-directional extension and symbolizing switchover points, are novel.

3. It is the first work which automatically learns meta-rules which can guide effectively the invocation of object-level (domain) rules.
4. It is the first work in AI which develops the theory and the method of feature condensation to enhance the efficiency of inductive concept learning. Previously, the efficiency is improved by adding heuristics.
5. It generalizes and amplifies previous approaches to handling errors in inductive concept learning.
6. It is the first work which applies machine learning techniques to debugging the knowledge base automatically in EMYCIN-like environments where, previously, human experts are relied on to debug the knowledge base.

1.7. Outline of the Thesis

Chapter 2 describes three learning methods with illustrations. "Search from the most general hypothesis" method learns multiple rules for each diagnosis class in a case library. Learning intermediate knowledge is required to build a KB with good accuracy. Learning disconfirming rules is important in an EMYCIN-like system where uncertainty is involved.

Chapter 3 introduces the condensation principle, defines the role of CONDENSER, and describes a useful feature condensation technique. A cost and benefit analysis and some illustrations are made to demonstrate the value of feature condensation in learning from examples.

Chapter 4 provides solutions of noise problems in AI learning. The source, measurement, and filtration of errors are described. "Minimizing errors" is the basic principle to recover the desired information from a noisy learning environment.

Chapter 5 describes automated debugging of the knowledge base for a misdiagnosed case. It takes advantage of learning to achieve the purpose. Techniques of editing the KB particularly in an EMYCIN-like framework are described with illustrations.

In Chapter 6, a theory and methods of formulating new meta-rules are developed. The effect of introducing meta-rules is carefully analyzed by defining utility-value on the basis of the cost and benefit and by some experiments.

Chapter 7 shows results obtained from implementing all the ideas developed in the previous Chapters. The validation of the the developed learning method is the central issue. Then, discussions and conclusions are made.

Chapter 2

Knowledge Acquisition via Machine Learning

2.1. Introduction

In this chapter, an inductive learning program, named *RL* (standing for rule learning), is developed for constructing and maintaining the knowledge base (KB) in expert systems. This program differs from other inductive concept learning programs in that it can define newly useful concepts which are not in the initial given language, in order to fill in the possible missing links in a complex reasoning network. In addition, this program possesses two other distinct features. First, each learned rule is assigned a number representing "degree of certainty" (see appendix A); second, disconfirming rules are learned as well. All the above features are intended to augment the capability of inductive concept learning, and can be tailored to different domains. The learning methods employed in this program are described in general terms, illustrated with its application to the jaundice domain.

The problems we concern here, as described in Chapter 1, are as follows:

- The first stage problem is,

Given: A case library.

Find: A set of rules that can diagnose
each case correctly.

- The second stage problem is,

Given: A faulty conclusion from the performance program.

Find: Improvement to the KB in order to achieve a correct conclusion.

In the case library, each case (training instance) is represented by a frame with two basic slots: a feature set (i.e., a set of feature value pairs), and a correct classification. The goal of the first problem is to construct a KB which can diagnose all (or most) cases correctly; the second problem is to update the KB as a new case with faulty diagnosis appears. A KB, after constructed, will constantly be updated when more and more cases appear. Here, one important issue may arise: if the original KB is poor because of the poor initial case library, inadequate language, or whatever (see error-source considerations in learning in Chapter 4), then the KB may not converge to its ideal state (no longer with faulty conclusions) even with drastic updating (editing). Therefore, careful selection of training instances and other noise-elimination measures are mandatory to construct a good KB and minimize the future editing. But can we assure a KB will converge to its ideal state if we have a good learning environment? Here, it should be stressed that a KB may continue to grow even if it is in the ideal state because as more features are added, more rules can be created; we do not mind adding more pieces of knowledge if they are true. Thus the above question actually indicates whether each individual piece of knowledge will converge to the truth. Since a piece of knowledge will be more accurate if it is based on a large amount of accumulated data, the answer to the above question is positive if the

learning system can incrementally update the knowledge created. This worry may further be diminished when we notice that a KB is constructed by human experts and edited after some tests, then it is in good shape. Perhaps, unless well-established theories are overthrown or the statistics of instances in a community is altered by some intruded destabilizing factors, we may generally assume that a KB will converge to a desirable state, though a limited amount of editing (e.g., adding new knowledge) is still required.

In order to solve the first problem, a method which performs a heuristic search from the most general hypothesis is developed; this model-driven and *batch-processing* (i.e., process all data at once) learning method is described in Section 2.2. The second problem deals with the KB debugging. In this chapter, a "focusing mode" of learning, described in Section 2.2.4, is developed to cope with the situation when a new case with faulty conclusion made by the performance program appears; rules which can achieve a correct conclusion are first learned with this mode, then the KB is edited by comparing the learned rules with the old rules (the part of the subsequent editing of the KB after learning is described in Chapter 5).

Section 2.3 describes learning intermediate knowledge, which is important to construct a KB with good accuracy and understandability. Section 2.4 describes learning disconfirming rules. Section 2.5 describes how to combine all the methods developed to construct a KB with both confirming and disconfirming knowledge, including intermediate knowledge.

As mentioned in Chapter 1, rule models and function templates are important for rule learning because what they provide, which we call *rule-forming* knowledge, is essential to

make the machine-learned rules more uniformly encoded and more semantically meaningful. We recapitulate the advantages brought about by this "rule-forming" knowledge as follows:

- It indicates the required components to add such that rules learned will conform to the rules written by human experts. For example, a rule learned is:

"if P, then C"

but a rule written by human experts is:

"if (1) C is unknown.
 (2) P
then C."

Though the two rules above do not make much difference, the second rule will be more realistic because it also states unless "C" is unknown, it is unnecessary to conclude "C".

- It tells about the commonly used predicates for a certain attribute.
- It guides generating code for newly learned rules in compliance with the old coding in the KB. Therefore, the new rules can be used by the inference engine once they are learned and can thereby be evaluated immediately to see their effects on the system performance (upgrading or degrading).
- It provides other common sense; e.g., mutual exclusiveness and relative priority among attributes. In the LHS of a rule, it allows no two conjuncts that are mutually exclusive with each other; also, the conjuncts should be ordered according to the sequence of occurrence or priority. If the failure of one conjunct will make other conjuncts meaningless, then this conjunct should be placed as the first one. For instance, it is logical to place the conjunct "LFT is known" before "SGOT is elevated" and "SGPT is elevated" because if "LFT is not known", then no data are available for "SGOT" and "SGPT".

This rule-forming knowledge, which is collected from observing rules written by experts, can actually be formulated directly by experts as the initial knowledge for the learning system to construct a KB from nil.

2.1.1. Learning in JAUNDICE

Though the methods we developed are general, we use JAUNDICE as the main experimental domain. There are 72 cases in the initial case library; a feature base with 81 medical features (some have binary values, others have multiple values) is used for describing cases; every case in the case library has been assigned a correct diagnosis given by human experts. A KB with 141 diagnostic rules and 80 nondiagnostic rules encoding causal and taxonomical knowledge was used as its starting point. The learning program in JAUNDICE, is involved in learning of diagnostic rules. Descriptions about cases or concepts or hypotheses in the search space are represented by *feature sets*, sets of feature and value pairs: e.g., {.....(SGOT 250) (SGPT 200) (Alk-P 25).....}. The temporal characteristics are also encoded into features; e.g., (disease-course rapid-downhill).

The rules of generalization used are listed as follows:¹⁵ (notation "G>" means "replaced by a more general form")

1. Dropping conditions:

$$\{(A_i V_i) (A_j V_j)\} \quad G> \quad \{(A_i V_i)\}$$

Based on this rule, generalization is done by removing some feature value pairs from the feature set.

2. Climbing up the value hierarchy tree:

$$\begin{array}{l} \text{If } V_i \text{ implies } V_k, \text{ and } V_j \text{ implies } V_k, \\ \text{then,} \\ \{(A_i V_i) (A_j V_j)\} \quad G> \quad \{(A_i V_k)\} \end{array}$$

¹⁵ Rules of generalization used in learning from examples can be seen in [Michalski 83b]. But, in JAUNDICE, we also add some domain specific rules.

This rule states that a value can be replaced by a more general value in order to cover more instances in the same class.

3. Creating new symbols: In Section 2.3, we will describe how to define new symbols for a higher level abstraction when indicated by some heuristics.
4. Taking minimum or maximum:

$$\begin{array}{l} \{(A_i \ a)\} \\ \{(A_i \ b)\} \end{array} \quad G> \quad \begin{array}{l} \{(A_i \ \geq \min(a \ b))\} \\ \text{or, } \{(A_i \ \leq \max(a \ b))\} \end{array}$$

This rule is designed for medical numerical parameters and can be understood by the following example. For two patients with the disease "hepatitis", one patient's data show "SGOT = 200", another's show "SGOT = 400"; then it may be hypothesized that "SGOT \geq 200" implies "hepatitis". Depending on the distribution of values among the normal and diseased populations, "taking minimum" or "taking maximum" rule is chosen. In JAUNDICE, this rule is made domain-specific by adding the following heuristic: if the high range of values suggests disease, then use the "taking minimum" rule; if the low range of values suggests disease, then use the "taking maximum" rule.

5. Allowing disjunction:

$$\begin{array}{l} \{(A_i \ V_i) \ (A_j \ V_j)\} \\ \{(A_i \ V_i) \ (A_j \ V_k)\} \end{array} \quad G> \quad \{(A_i \ V_i) \ (A_j \ V_j\text{-or-}V_k)\}$$

To avoid trivial disjunction, this rule is invoked only under certain circumstances. For example, the important features of two instances in the same class are matched; then the less important features which are not matched and there are no other ways to generalize can be generalized by this rule.

Rules of specialization are logically opposite to those of generalization. There are three rules listed in the following: (notation "S>" means "specialization" of concept descriptions or the LHS of rules)

1. Adding conditions:
2. Climbing down the value hierarchy tree: In the tree, the highest level nodes are the most general values or descriptions; the lowest level nodes are the most specific values or descriptions. For example, a value "2" is more specific than a value "even", and the value "even" has infinite successors: -2, 0, 2, 4,...
3. Closing interval: If the value is numerical and the interval is too open, then it can be specialized by closing the interval as follows.

$$\{(Ai \geq a)\} \quad S> \quad \{(Ai [a \ b])\} \\ \text{or } \{(Ai \geq b)\}$$

"b" is the next higher marking level of "a".

For example, a description $\{(SGOT \geq 50)\}$ is too general for the disease "Cholangitis" and can be specialized into $\{(SGOT [50 \ 300])\}$ or $\{(SGOT \geq 300)\}$; however the latter is not proper.

The specialization operator of "changing disjunction to conjunction" is not applied because, as described in next section, the learning employs search from the most general hypothesis: "NIL".

The main features of learning in JAUNDICE are summarized in table 2.1.

Table 2.1 Learning in JAUNDICE

Paradigm:	Learning from example paradigm.
Representation: (of training instances)	Feature set.
Rules of generalization:	<ol style="list-style-type: none"> 1. Dropping conditions 2. Climbing up the value hierarchy tree. 3. Creating new symbols. 4. Taking minimum or taking maximum. 5. Allowing disjunction.
Rules of specialization:	<ol style="list-style-type: none"> 1. Adding conditions. 2. Climbing down the value hierarchy tree. 3. Closing interval.
Control rules:	Learning meta-rules (see Chapter 6).
Intended applications:	Expert systems in general.
Applications:	JAUNDICE (REFEREE)
Efficiency enhancement:	CONDENSER (see Chapter 3). Heuristics
Noise elimination:	Noise filter (see Chapter 4).

As described in Chapter 1, in order to avoid unnecessary risk and cost, the RL program, which is adopted as the learner of JAUNDICE, is designed to keep the learned rules small (i.e., mention no unnecessary features), and keep them maximally specific while sufficiently general (may be viewed as the most specific rules in the version space) for minimizing false positive errors (incorrect diagnoses). The subprogram CONDENSER (described in Chapter 3), by removing unnecessary features, can not only enhance the efficiency of learning, but also save cost and risk in a given domain. In medicine, a

diagnostic rule is valuable if it can arrive at a conclusion by using cheap and safe clinical features. Error-sources (noise) associated with the learning system will be considered in Chapter 4.

2.2. Learning via Search from the Most General Hypothesis

This learning method, performing a heuristic search from the most general hypothesis, is model-driven and is designed to learn multiple disjunctive concepts (i.e., there are multiple concepts and there are multiple rules for each concept). This method is intended to abstract rules from a case library; the learning task is formulated as follows:

Given: A case library.

Find: A set of rules that can diagnose each case in the case library correctly.

Note that the ultimate goal is that the learned rules should also provide correct diagnoses for cases that are not in the case library used for training; therefore, the learned rules should be sufficiently general and specific.

Since we use a set of training instances to estimate the "true" boundaries (i.e., rules) between positive and negative instances, the learned rules will be associated with some degree of error. The errors concerned here are false prediction errors, i.e., rules make wrong predictions. We particularly desire to minimize false positive predictions because of reasons described in Section 2.2.2. Therefore, this learning method is intended to discover rules that describe a group of positive training instances in maximally specific fashion. They thereby minimize false positive predictions (i.e., predict negative instances

as positive instances). Moreover, a learned rule will tend to be overly generalized (overgeneralization implies false positive predictions) if there are not adequate negative training instances to constrain or guide generalization properly [Carbonell 83]. Hence, this method will be even more useful if negative training instances are only limitedly available. Refer to Section 4.2.2.2 for the philosophy of handling sampling insufficiency.

There are domain dependent constraints for rules. The learning method searches for rules which are maximally specific without breaking the constraints. The constraints, just like the half-order theory in Meta-DENDRAL, are based on the domain knowledge. In JAUNDICE, the constraints are defined as follows:

1. The LHS of a rule should have less than seven conjuncts.¹⁶
2. A rule should cover (be matched by) at least 20% of positive training instances (the class of instances for which we want to learn classification or diagnostic rules).¹⁷
3. The "degree of certainty" of a rule should be at least ".4". That is, the prediction should be reasonably certain.
4. A rule should not cover more than 10% of all negative training instances.¹⁸

¹⁶This constraint is based on the observation that a rule in rule-based medical expert systems usually has less than seven components in its LHS.

¹⁷This is domain dependent. If there should be only one rule that covers all the positive instances, then the rate of coverage should be 100% ideally. But we assume there are multiple disjunctive concepts to be learned (as in Meta-DENDRAL); so it is unlikely that a single rule will cover all instances. As another example, in diagnosing acute appendicitis, the rules should be more general to cover more positive instances because the mortality of this disease is high whereas the surgical risk is small; therefore the threshold should be set higher.

¹⁸Though, ideally, a rule should not cover even a single negative instance, this is, however, not probable because of uncertainty. It is also noted that, in the EMYCIN system, for instance, a case in class A, which is covered by a rule, rule B, inferring class B, will still be classified correctly if a rule, rule A, which infers class A and covers this case overrides rule B; recall the phenomenon of hypotheses competition in such a system.

The first two constraints define the minimal generality whereas the last two constraints define minimal specificity.

2.2.1. Procedure

In a case library, if we want to learn rules for a certain class, then label all cases in that class as positive instances and label other cases as negative instances. Inasmuch as the goal is to construct a KB from the case library, each class will be labelled as positive instances at a certain stage during the whole learning process.

During learning, CONDENSER condenses the feature base dynamically with respect to the class labelled as positive instances to determine the set of required features, which is a subset of the feature base (see Chapter 3). The learning procedure includes four main steps described as follows:

- step 1. Starting from the most general version, "NIL", search for the maximally specific hypotheses that does not break two following constraints: the number of conjuncts should be less than seven (adjustable), and the hypotheses should cover at least 20% (adjustable) of positive instances. The hypotheses, thus found, are formed as raw rules. Since the constraints merely involve positive instances, only positive instances are considered in this step.
- step 2. Prune those rules which are assigned a degree of certainty smaller than ".4" (adjustable), or which cover more than 10% (adjustable) of negative instances. Negative instances are considered in this step for calculating degree of certainty (refer to Appendix A).
- step 3. Optimize each raw rule by iteratively applying generalization operators (generalization rule #1, #2, and #4 in Section 2.1.1) until a local optimum is reached (perform a hill-climbing search, so to speak). The local optimum is the state with minimal weighted prediction error (as defined in Section 4.5.3.1) under the following constraints: the local optimum should not cover more than 10% of negative instances as mentioned in step 2, and the difference of

degree of certainty between the local optimum and the initial state (a raw rule) should be within ".15"; the latter constraint stems from the argument that, in EMYCIN-based systems, it is hard to compare two rules with different ranges of degree of certainty.

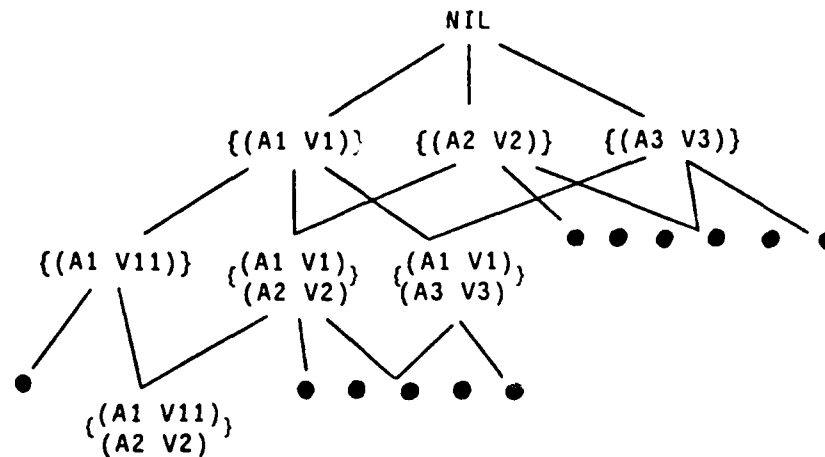
- step 4. If all positive instances are covered or the rate of uncovered positive instances is below a certain threshold or the number of iterations has reached a certain threshold, then exit. Otherwise, go to step 1 and reset the constraints in step 1 as follows:
 1. Reduce the rate of coverage for positive instances: for example, the first iteration uses 20%, the second iteration uses 10%, and so on.
 2. The hypotheses should cover at least one of the uncovered positive instances.
 3. The number of conjuncts is still kept under seven.

If the constraints used in the procedure are properly chosen, then only a few iterations are required, provided that the training instances selected are not too noisy.

The search in step 1 proceeds as follows:

- substep 1. Initialize the hypothesis space H with the most general version as follows: set $H := \{ \text{NIL} \}$.
- substep 2. Generate new hypotheses by specializing each hypothesis in H in all possible ways. But, the specialization for generating new hypotheses should be maximally general (i.e., specialize as little as possible). Each parent hypothesis may have more than one successor hypothesis. The specialization may be done by either adding a new feature with the most general value, or replacing a feature value by a more specific value. Figure 2.1 shows part of the search tree.

Since the number of conjuncts is limited below seven, the depth of the search tree is mainly affected by (but not the same as) the depth of the value hierarchy tree, and its breadth is affected by the breadth of the value



Suppose the system is learning classification rules for "class A" instances, then the hypothesis generation in the above diagram can be formulated as:

instances \Rightarrow class A instances

instances with (A1 V1) \Rightarrow class A instances

.....

instances with (A1 V11) \Rightarrow class A instances

.....

instances with (A1 V11) and (A2 V2) \Rightarrow class A instances

.....

Note: For the feature "A1", "V11" is more specific than "V1" in the value hierarchy tree.

Figure 2.1 Part of the search tree in the "search from the most general" mode.

hierarchy tree and the number of the available features. Inasmuch as the search space may be huge, heuristic search is necessary. The heuristics used will be described next.

- substep 3. If a successor hypothesis is justified (i.e., does not violate the constraints defined by the minimal generality, as described in step 1), then retain it in H and prune the parent hypothesis. If a successor hypothesis is not justified, then prune it. If no any successor hypothesis is justified, then output the parent hypothesis as a raw new rule and remove the parent hypothesis from H. Also, remove redundant hypotheses during the search.
- Repeat substep 2 and substep 3 until H is empty.

If a hypothesis is pruned, it indicates that the number of conjuncts is greater than six or the positive instances covered is less than 20% of all positive instances; successors (specializations) of this hypothesis will also be unjustified, because specialization never causes more instances to be covered or causes the number of conjuncts to drop. Therefore, pruning of unjustified hypotheses will not hurt the completeness of the search for desirable rules.

Step 4 is designed for handling more special cases (or exceptional cases) which are not covered by rules learned in the first iteration. As more iterations go, the learned rules become more and more specific and cover a smaller number of positive instances.

The search space is reduced to reasonable dimensions by:

1. The features are condensed by CONDENSER before learning.
2. Heuristics are used to prune the search space.

Notice that *the system parameters, such as "the minimal coverage of positive instances" (defined to be 20% in JAUNDICE), can be adjusted to different domains.*

2.2.1.1. One Example

Here, an example taken from JAUNDICE is used to demonstrate the search process in the above described procedure. For simplicity, we assume only four cases in the case library and only two features are used to describe the cases.

Case1: Hepatitis, {(GPT 1200) (Alk-P 8)}
 Case2: Hepatitis, {(GPT 450) (Alk-P 6)}
 Case3: Calculous-jaundice, {(GPT 200) (Alk-P 20)}
 Case4: Calculous-jaundice, {(GPT 60) (Alk-P 8)}

The value hierarchy tree rests with the domain knowledge. In JAUNDICE, the specialization of a numerical feature is done by specialization rule #3:

$$\{(A \geq a)\} \quad S > \quad \text{or} \quad \{(A [a \ b])\} \\ \{(A \geq b)\}$$

where, "b" is the next higher marking level of "a".

Now, the search tree of learning rules for "Hepatitis" is diagramed in figure 2.2. Assume the values of "GPT" are marked off by the following two levels: 50 and 300, and the most general value is assumed to be " ≥ 50 "; the values of "Alk-P" are marked off by the following two levels: 4 and 10, and the most general value is assumed to be " ≥ 4 "¹⁹. In the figure 2.2, the output raw rule is as follows:

R1: {(GPT ≥ 300) (Alk-P [4 10])} => Hepatitis

This raw rule then goes through step 2 and step 3 described in the above procedure.

¹⁹ In the real experiment, the most general value for any numerical feature is ≥ 0 .

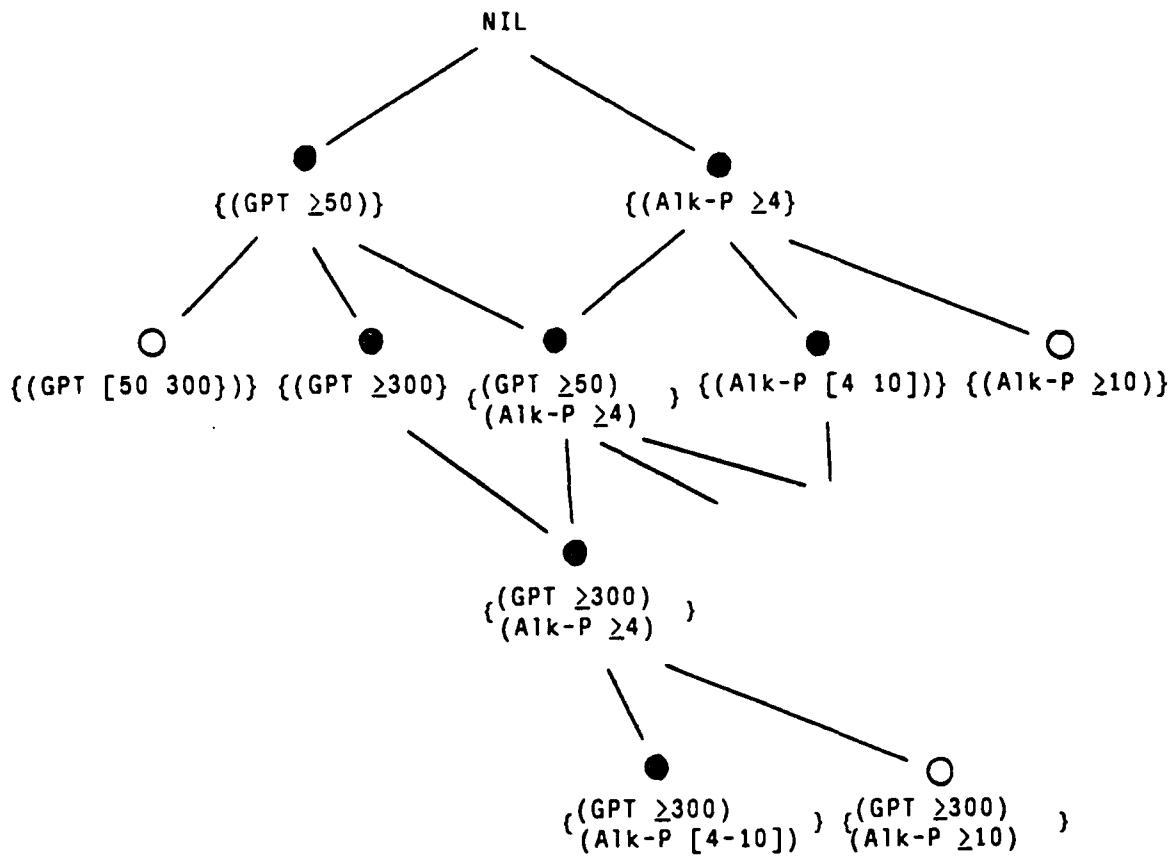


Figure 2.2 The search tree of applying the "search from the most general" mode to one example.

2.2.2. Areas of Application

Inasmuch as this learning method is initially focused on systems with EMYCIN-like frameworks, its applicability is expected in domains where EMYCIN can apply, e.g., medical examples, such as MYCIN, PUFF, HEADMED, CLOT, and nonmedical examples, such as SACON [Van Melle 80]. However, from the methodological viewpoint, this developed method can be applied in a domain where there are multiple disjunctive concepts²⁰. This method is particularly useful when the following features exist:

- Uncertainty is involved, or,
- False positive predictions are to be minimized, or,
- Negative training instances are of limited availability.

The medical domain, bearing all the features above, is expected to be uniquely well applied by this described method.

Determinism (or exactness) of a domain will not preclude the use of this described method, since such a domain is merely an extreme case of an imprecise domain, where "degree of certainty" is quantized into two levels: "yes" and "no".

With respect to the accuracy of performance, false negative predictions (cases which are not predicted to be any pre-defined category) are more advantageous than false positive predictions (incorrect predictions) since the system will continue to request desired (or missing) information which helps to attain a prediction. For example, assume there is only one rule in the KB as follows: "A1 & A2 => Class C"; then an instance in class C will be

²⁰ A disjunctive concept denotes a concept incorporates multiple rules.

falsely negatively concluded if it is known to have only the attribute A1; and the system may continue to gather information about the attribute A2. In medicine, a physician might argue that, sometimes, a false negative diagnosis will be hazardous owing to delayed therapy: however, even without a (specific) diagnosis, a therapy can still be instituted immediately under the worst assumption (default therapeutic decision) while more information is being gathered for arriving at a diagnosis. On the other hand, as AI people working on expert systems have been aware the system's performance is often more stringently evaluated by the public than an expert's performance; that is, there is a double-standard. Thus a false positive prediction (an incorrect prediction) will jeopardize the image of the system more than a false negative prediction (a case which is not predicted to be any pre-defined category) with a list recommending missing information. Consequently, minimizing primarily false positive predictions is justified if false positive predictions and false negative predictions can not be minimized simultaneously (i.e., if there is a tradeoff between false positives and false negatives).

The next issue is, why and when are negative instances of limited availability? According to [Carbonell 83], training instances are obtained from three sources: the learning system (if the system can generate and verify instances), the teacher (so called "supervised learning"), and the environment (by observation). In medicine, training instances that can be generated hypothetically by human experts are limited to more typical or simpler cases; more complex cases usually can only be obtained by clinical observation. Also, in a relatively unexplored domain, the better (or more reliable) way to obtain instances is by observation.

Then, what areas are not suitably applied with this method? For single concept learning (only a single or a few rules for a given concept), though this described method can still be applied in a somewhat awkward way (because of inefficiency), we would rather adopt other learning algorithms, such as the version space approach [Mitchell 78]. The version space approach, assuming there is only one single description (one single rule) for a given concept and performing a bidirectional search by maintaining two boundary sets ("G" set and "S" set), will render the desired concept description more rapidly converged upon than our method, as described, which performs an unidirectional search. On the contrary, if there are multiple different descriptions (rules) for a given concept, the version space approach has to be modified (e.g., A^q algorithm [Michalski 75]); the learning system has to determine which group of positive instances belong to the same description (or are included by the same rule) before attempting to generalize. More discussion of model-driven vs. data-driven type learning follows in the next subsection.

2.2.3. Comparison and Discussion

2.2.3.1. Comparison with Related Works

For comparison, we pick up, among the machine learning work in AI, some important prototypes which also deal with learning multiple concepts or multiple rules; they include the following works: Meta-DENDRAL [Buchanan 78a], AQ11 [Michalski 78], and ID3 [Quinlan 83].

Meta-DENDRAL is similar to the above developed method in the following aspects:

1. They both are model-driven learning systems, performing a heuristic search from the most general hypothesis.

2. They both are intended to discover rules that are sufficiently general and specific.
3. They both consider positive training instances first, and then negative training instances.

However, the output of the two programs will differ because of different heuristics used in the search. The RULEGEN program in Meta-DENDRAL assumes that the "improvement criterion", which compares one hypothesis with its successors with respect to plausibility, increases monotonically; therefore a cleavage rule will be formed from the hypothesis space if the improvement criterion reaches a local maximum [Buchanan 78a]. In contrast, the above developed method doesn't assume so, and a rule will be formed only if it is maximally specific without breaking the constraint defined by minimal generality; in other words, the method seeks boundary conditions of a region bounded by the pre-defined constraints instead of seeking a local optimum. The rationale behind this is twofold:

1. Unless the heuristic function used increases monotonically, the local maximum (or minimum) is not necessarily the most desirable result.
2. As described earlier, it is desirable to minimize false positive predictions. Finding the most specific rules in the version space is the most important solution if negative training instances are not easily available.

AQ11 uses A^9 algorithm [Michalski 75] and differs from the above developed method in the following aspects. AQ11 uses the version space approach, a data-driven learning. In terms of the version space defined by [Mitchell 78], AQ11 discovers rules in the G set (the most general rules in the version space) while the above developed method discovers rules in or near the S set (the most specific rules in the version space). If the version space

converges such that $G = S$, both methods will achieve the same result. There are two possible weak points for AQ11 algorithm in EMYCIN-like frameworks:

1. If there are no adequate negative training instances to update the G set, the rules in it will be overly general and cause more false positive predictions. However, if we can ascertain that we have adequate negative training instances to guide the generalization, finding the most general rules is more advantageous in the aspect of reducing the cost of using rules because these rules tend to have a smaller number of features.
2. With A^q algorithm, the set of rules found is incomplete. This is due to that A^q algorithm repeatedly applies the candidate elimination algorithm with a portion of positive training instances removed during each iteration, and the procedure is terminated when all positive training instances are covered by a set of rules, rather than when all desired rules are found (refer to [Michalski 75] and [Michalski 78]).

The difference between the above developed method and the ID3 algorithm is derived from different representation schemes. The ID3 algorithm uses decision trees instead of production rules to represent knowledge. The weakness of ID3 algorithm in EMYCIN-like frameworks includes the following aspects:

1. A decision tree representation is more restrictive than a production rule representation. For example, if we transform the decision tree, which is constructed by the algorithm, into a set of rules, then each rule will rigidly share at least one common feature that occupies the first decision node. The distinction would be less, however, if the algorithm were intended to discover a set of decision trees.
2. Search is incomplete because, to construct the desired decision tree, features are selected on the basis of their discriminating ability with respect to some criterion. However, note that conjunction of two trivial features may be significant.
3. ID3 will fail if uncertainty is involved; for instance, some positive instances and negative instances share an identical set of feature-values.

2.2.3.2. Model-Driven vs. Data-Driven Learning

By "model-driven", we mean the hypotheses are generated by a model (and then tested by data); e.g., Meta-DENDRAL [Buchanan 78a]. By "data-driven", we mean the hypotheses are generated on the basis of data (training instances); e.g., the version space approach [Mitchell 78]. In the following discussion, we compare model-driven and data-driven learning systems along several different dimensions to justify why model-driven learning is selected to be the approach in EMYCIN-like rule based systems.

- *Completeness*: The heuristics used in pruning the search space in the above developed method still preserve the completeness of the search for the desirable rules (see Section 2.2.1). In contrast, AQ11, the most important example of a data-driven learning system designed to discover multiple disjunctive concepts, performs an incomplete search, as described in last subsection. Intuitively, since our goal is to discover all desirable rules, a model-driven search in the rule space (a space equivalent to the power set of the set of all descriptors used to describe rules) will tend to be more complete than a data-driven search in a subspace of the rule space by interpreting the instance space.
- *Noise immunity*: Model-driven learning systems are more immune to noise than data-driven learning systems [Dietterich 83]. Because model-driven techniques are intended to find rules that are good in a global sense (i.e., there is a global criterion to evaluate rules), the effect of noise associated with the individual data (e.g., false positive or negative training instances) can be relieved under the assumption that the imperfect training instances are the minority.²¹ In contrast, data-driven techniques handle the instances on the individual basis, and thus is harder to escape the noise associated with the data. One false positive instance will force a rule to be overly generalized while one false negative instance will force a rule to be overly specialized (see Chapter 4 for noise considerations).
- *EMYCIN-rules*: In EMYCIN-based systems, a case is concluded by

²¹ This assumption often holds; if not, then actually no method can learn good rules.

combining several different rules that are described by different sets of features and reason from different angles. Likewise, a case is often analyzed, based on several simple rules rather than a single long rule. For example, in JAUNDICE, the number of conjuncts in the LHS of a rule is restricted below seven. This fact makes data-driven techniques difficult to learn in EMYCIN-based systems because the learning system has to determine how to decompose one generalization (which may have many features) drawn from instances into a set of proper rules.

- *Efficiency:* In a domain with multiple disjunctive concepts, if a model-driven method is used, the search space will be roughly the power set of the set of all descriptors involved; if a data-driven method is used, the search space can be roughly estimated from the power set of the set of all positive instances since the system has to determine which group of instances should be hypothesized together. If the number of descriptors is greater than the number of positive instances, it may be more efficient to use a data-driven approach (if we ignore the disadvantages described above). Nonetheless, in real practice, the number of positive instances is often greater than the number of the descriptors (unless we carefully select instances as we did during constructing the initial case library in JAUNDICE), it will be more efficient to use a model-driven method. Furthermore, the CONDENSER program controls the number of features during learning in order to enhance the efficiency of learning; therefore, we don't think efficiency will be the bottleneck for applying the method we develop.
- *Incremental learning:* In version space approach, it is claimed that learning is incremental by constantly updating the boundary sets without re-examining the old instances as new instances emerge. However, this claim assumes that the learning environment is perfect initially, e.g., there are adequate rules of generalization or specialization, etc.; otherwise, the maintained boundary sets may not necessarily reflect the genuine information stored in the instances. Our strategy for incremental learning is by applying "focusing mode" of learning (described in Section 2.2.4) to update the KB each time when a faulty conclusion occurs.

In conclusion, we determine to use a model-driven strategy because of the following advantages:

1. It provides a complete search for the desired rules.
2. It has better noise immunity.
3. It can accommodate EMYCIN-like rule based systems.
4. The efficiency can be reasonably controlled.
5. The learning can still be incremental.

2.2.4. Focusing Mode of Learning

Focusing mode of learning is a learning strategy which focuses on a specified instance.

The task is formulated as follows:

Given: 1. A specified instance.
2. A set of training instances.

Find: Concept descriptions which are consistent with the specified instance and most of the other instances.

If the specified instance is a positive instance, then the learned concept descriptions should cover it as a positive instance; if the specified instance is a negative instance, then the learned concept descriptions should reject it as a negative instance.

The main purpose of this mode is twofold:

1. In a domain like medicine, inconsistency often occurs. Though we don't expect a rule will be consistent with all instances, however, sometimes, we do hope the rule will be consistent with an interesting, valuable instance.
2. In an expert system, if the system comes up with a wrong conclusion about a new case, then the system might be able to learn by focusing on that new case.

In this section, however, we describe how to learn multiple rules, based on a misdiagnosed case. The task of focusing mode of learning is defined as follows:

Given: 1. A misdiagnosed case* by the performance system.
2. A case library.

Find: Rules which can make the misdiagnosed case diagnosed correctly.

*: "Misdiagnosis" may include "underdiagnosis",
i.e., no diagnosis is made.

There are two solutions for this problem: finding confirming rules to support the expert's diagnosis, and finding disconfirming rules to reject the system's misdiagnosis; described here is the former solution; the latter solution is described in Section 2.4. Notice, however, the rules found should be *relevant*, i.e., they should satisfy (be matched with) the specified case.

2.2.4.1. Procedures

For reasons given in Section 2.2.3, we also select model-driven strategy for focusing mode of learning. The efficiency of learning can be much improved by CONDENSER, which picks up the relevant features from the feature base on the basis of the specified instance.

Focusing mode of learning uses the method described in Section 2.2.1 except that there is one additional constraint, which is that the rules should satisfy (be matched with) the specified instance.

First, label the misdiagnosed case as "positive instance", and the case library (or instance space) is classified into positive and negative instances, based on the expert's diagnosis for the misdiagnosed case as the following example. If a case, Case01, is misdiagnosed as disease B whereas the expert diagnoses it as disease A, then all cases

diagnosed as disease A in the given case library are labelled as "positive instance", so is the misdiagnosed case; and all other cases are labelled as "negative instance". (Note that each case in the case library has already been assigned a correct diagnosis.) Then, the learning is based on this dichotomy, focusing on the misdiagnosed case. The learning task in the above example is to learn LHS of the confirming rules with RHS, which is "disease A".

The procedure includes the same four main steps procedure as the search from the most general hypothesis with minor changes noted in *italics*:

- step 1. Starting from the most general version, "NIL", search for the maximally specific hypotheses that does not break three following constraints: *the hypotheses should satisfy (be matched with) the specified instance*, the number of conjuncts should be less than seven, and the hypotheses should cover at least 20% of positive instances. The hypotheses, thus found, are formed as raw rules. Since the constraints merely involve positive instances, only positive instances are considered in this step.
- step 2. Prune those rules which are assigned a degree of certainty smaller than ".4", or which cover more than 10% of negative instances. Negative instances are considered in this step for calculating degree of certainty (refer to Appendix A).
- step 3. Optimize each raw rule by iteratively applying generalization operator until a local optimum is reached (perform a hill-climbing search, so to speak). The local optimum is the state with minimal weighted prediction error (as defined in Section 4.5.3.1) under the following constraints: the local optimum should not cover more than 10% of negative instances as mentioned in step 2, and the difference of degree of certainty between the local optimum and the initial state (a raw rule) should be within ".15", the latter constraint stems from the argument that, in EMYCIN-based systems, it is hard to compare two rules with different ranges of degree of certainty.
- step 4. If there are rules learned or the number of iterations has reached a certain threshold, go to exit. Otherwise, go to step 1 and reset the constraints in step 1 as follows:

1. *The hypotheses should satisfy (be matched with) the specified instance.*
2. Reduce the rate of coverage for positive instances; for example, the first iteration uses 20%, the second iteration uses 10%, and so on.
3. The number of conjuncts is still kept under seven.

The detailed search procedure is described in Section 2.2.1, and is neglected here.

There are two outcomes of this learning: success and failure. If there are rules learned, then one might ask why the learning system didn't find them when the KB is initially constructed by using the batch-processing learning (described in Section 2.2.1). In the first place, the specified case may be an exceptional one and the rules that are consistent with it can cover only a small number of other positive instances; thus those rules dealing with this exception may not be found during the initial KB construction. Secondly, as the case library grows, the statistics may shift; hence a relatively bad rule may become a good one. However, as the case library grows, the statistics will converge (i.e., the sample statistics will approach the true population statistics), this second possibility will be less. (We have briefly touched on the issue of convergence of a KB in Section 2.1.) On the contrary, if there are no rules learned, what does this mean? Since the objective of this learning is to find rules that are consistent with the specified instance and most of other instances, no rules being found implies the inconsistency between it and other instances or the incompleteness of its data. And this may indicate the wrong diagnosis given by the expert.

2.3. Learning Intermediate Knowledge

In expert systems, hierarchical reasoning can provide better accuracy and understandability. Here, we develop a method of learning hierarchical knowledge from a case library, in which each training instance is described by low level features and high level concepts but not by intermediate concepts. This reasoning hierarchy is shown in figure 2.3.

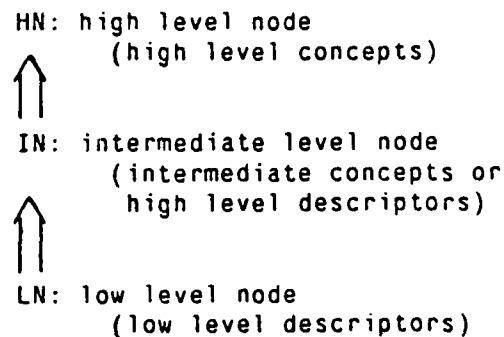


Figure 2.3 Multi-level reasoning network. Note that there may be several intermediate levels.

Intermediate concepts can be high level descriptors or intermediate classifications or mechanisms; in medicine, they are clinical syndromes or nosological categories or pathophysiological mechanisms.

Learning intermediate knowledge (intermediate concepts and links) is motivated by the following facts:

- Intermediate knowledge increases the accuracy of reasoning particularly if the data are incomplete. For instance, sometimes we are unable to tell whether an animal is a dog, but we may still be able to tell it is a mammal from the limited information.

- Intermediate knowledge provides better explanation capability of the system and thereby increases the understandability for the users [Clancey 83]. For example, the analysis of the underlying pathophysiological mechanisms is important for explaining a disease diagnosis.

In learning from examples, each training instance is described by features or descriptors (i.e., LN) and assigned a class name or diagnosis (i.e., HN). Thus, we may define the task of learning intermediate knowledge as:

Given: A set of training instances
(or a set of LN \rightarrow HN).

Find: Rules of the form LN \Rightarrow HN,
consistent with the training instances.

In the above formulation, " \rightarrow " represents a specific link in a training instance; " \Rightarrow " represents *general inferential knowledge* (a rule). In other words, in learning from examples, we intend to learn general knowledge from a set of very specific descriptions. Practically, this is an important issue and worthwhile to explore, because, for instance, medical records are often described only by clinical manifestations and disease diagnoses (there is no or limited discussion of the involved intermediate mechanisms).

So far as learning is concerned, we distinguish two situations:

1. Intermediate nodes exist in the initial vocabulary.
2. Intermediate nodes are not in the initial vocabulary.

In the first situation, the learning task involves exploiting the old partial (incomplete) intermediate knowledge to learn new intermediate knowledge; the partial knowledge may exist between LN and IN, or between IN and HN, or both. But the task in the second

situation is more abstract and constructive, because it involves creating and defining new intermediate nodes which are not provided in the original vocabulary. Automatically adding new symbols is one way to relieve the bias induced by the fixed language in inductive concept learning [Utgoff 82]. Since, in both situations, the existing knowledge may be only partial or missing, search is indispensable. In fact, there is a tradeoff between knowledge and search. A search procedure involves primarily two steps which are hypothesis formation and verification. In learning from examples, the hypotheses may be verified in a case library. The main difficulty of this learning problem is that training instances in a case library are described only by LN and HN, but we want to learn IN.

In medicine, LN names a medical manifestation, such as a symptom, a sign, and HN names a disease category. The link from manifestations to a disease is called an inference (or diagnostic) rule, the opposite is called a descriptive rule. We focus on learning diagnostic rules and their intermediate concepts (i.e., $LN \Rightarrow IN \Rightarrow HN$).

This section describes the learning techniques when the intermediate concepts are in the initial language and then when the intermediate concepts are not in the language.

2.3.1. Intermediate Concepts in the Initial Vocabulary

In this subsection, we assume there is some knowledge about the intermediate nodes (IN) including partial knowledge about their relationship with other level nodes (LN or HN). However, if each training instance is characterized with low, intermediate, and high level descriptors, we can apply machine learning algorithms level by level and discover new knowledge in different levels. (Some work, such as [Blum 82] can discover causal links by statistical analysis of temporal associations.) Nonetheless, in this work, we assume

each training instance is characterized only by LN and HN, and not by any intermediate description; thus the task becomes more abstract and difficult and the existing partial knowledge has to be exploited.

Basically, two methods are used to tackle the problem: bottom-up and top-down. The bottom-up method relies on the existing knowledge of "LN \Leftrightarrow IN"; the top-down method relies on the existing knowledge of "IN \Leftrightarrow HN". Therefore, if only the knowledge of "LN \Leftrightarrow IN" is available, only bottom-up method can be used; if only the knowledge of "IN \Leftrightarrow HN" is available, only top-down method can be used. If both types of knowledge are available (but incomplete),²² both methods can be applied and the results will be the union of results from each method. A third method called "bidirectional extension" employs these two basic methods bidirectionally and sequentially in order to construct more complex hierarchical concepts.

2.3.1.1. Bottom-Up Learning

If the knowledge about "LN \Rightarrow IN" is available, this strategy can be adopted. The task of learning under this situation is described as:

Given: 1. A set of training instances
(or a set of LN \rightarrow HN).
2. LN \Leftrightarrow IN
(i.e., rules linking LN and IN)

Find: Rules of the form LN \Rightarrow IN \Rightarrow HN,
consistent with the training instances.

Since "LN \Rightarrow IN" is known, the main task is to find "IN \Rightarrow HN". Each instance is

²²If both types of knowledge are complete, there is nothing to be learned.

represented by a direct link, $LN \rightarrow HN$. The training instances are classified on the basis of HN . Remember that HN is a class name (or a disease in medicine). The basic idea behind this strategy is to generalize from instances of the same HN ; it is analogous to generalizing from positive instances. In the JAUNDICE experiments, a set of rules in the form of " $LN \Rightarrow HN$ " are first learned by the method described in Section 2.2 from the given set of training instances, then we treat this set of rules as another set of training instances (more general, of course) and apply the following procedure to learn intermediate rules.

Suppose there are n classes of HN in the instance space: $H_1, H_2, \dots, H_i, \dots, H_n$. The algorithm proceeds as follows:

For $i = 1$ to n , Do:

- step 1. Label all instances in the class of H_i as positive instances and label other instances as negative instances.
- step 2. Generalize from positive instances by using the concept hierarchical tree (one example of the concept hierarchy is the animal tree seen in Section 2.3.1.2). The generalization should:
 - be maximally specific to avoid over-generalization.
 - test against negative instances.

Intermediate nodes are involved and intermediate links ($IN \Rightarrow HN$) are discovered during the process of generalization via hierarchy (see generalization rule #2 in Section 2.1.1).

The step 2 proceeds as follows:

- substep 1. Initialize the hypothesis space H with the set of all positive instances: i.e., each positive instance represents one hypothesis in H .

- substep 2. For each hypothesis h_i in H (starting from the head of H), do the following:
 - Form set H^* by finding all hypotheses in H with the same range of degree of certainty ($\pm .15$) as h_i (H^* excludes h_i).
 - For each element h_j in H^* , find the common maximally specific generalization (called h_k) of h_i and h_j . If h_k is plausible, i.e., if it does not break the following constraints: its associated degree of certainty is at least ".4" and in the same range as h_i 's and it should not cover more than 10% of all negative instances,²³ then retain it (h_k), put it in the end of H , and prune h_i from H . If no element in H^* can form a plausible generalization (without breaking the constraints above) with h_i or H^* is an empty set, then output h_i as a new rule and prune it from H .

Also, remove redundant hypotheses from H .

- substep 3. Repeat substep 2 until H is empty.

This data-driven algorithm differs from the version space approach [Mitchell 78] in that this algorithm considers not only that there is disjunction (i.e., there are multiple rules for each concept) but also that an instance may be covered by several rules instead of a single rule.

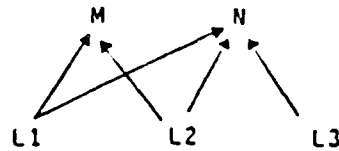
In the following simplified example, we assume no uncertainty is involved and that two hypotheses are mutually exclusive. (Both assumptions can actually be removed.)

Example 1. Suppose 3 instances in the instance space:

X1: L1 & H1
 X2: L2 & H1
 X3: L3 & H2

²³We do not consider the minimal generality here because, as stated earlier, this method is applied to the set of rules learned by the method described in Section 2.2; therefore they are already sufficiently general.

A simple hierarchy is given by five rules, shown schematically as:



The following are two possible generalizations from X1 and X2:

G1: $M \Rightarrow H1$

G2: $N \Rightarrow H1$

G2 is not justified because if it is true, then:

$L3 \Rightarrow N \Rightarrow H1$, contradicting X3.

Thus, the found link is:

Pi: $M \Rightarrow H1$

In the JAUNDICE domain, for example, the two instances, "esophageal varices \Rightarrow hepatic cirrhosis", and "ascites \Rightarrow hepatic cirrhosis" may be generalized into "portal hypertension \Rightarrow hepatic cirrhosis" by using the existing knowledge, "esophageal varices \Rightarrow portal hypertension" and "ascites \Rightarrow portal hypertension".

This technique is extended from climbing a generalization tree, used in other works, such as [Winston 70] and [Michalski 83a]. However, we emphasize a *concept* hierarchy instead of a *value* hierarchy. Moreover, uncertainty may be involved in the hierarchy.

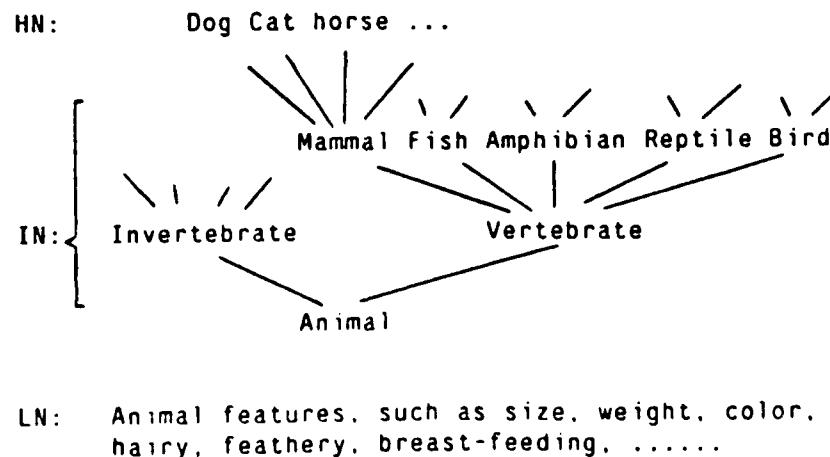
2.3.1.2. Top-Down Learning

If knowledge exists between IN and HN, this technique can be applied. The task is formulated as:

Given: 1. A set of training instances.
(or a set of $LN \rightarrow HN$)
2. $HN \Leftrightarrow IN$

Find: Rules of the form $LN \Rightarrow IN \Rightarrow HN$,
consistent with the training instances.

Since the knowledge about " $IN \Leftrightarrow HN$ " is known, the main task is to find " $LN \Rightarrow IN$ ". In order to learn " $LN \Rightarrow IN$ ", we may first, based on the available knowledge of " $HN \Rightarrow IN$ ", re-label training instances such that they have new class names which are IN instead of HN. After this transformation, the algorithm used in learning " $LN \Rightarrow HN$ " can be applied, and the results will be " $LN \Rightarrow IN$ " which is consistent with the training instances. One example is seen as follows.



The set of instances:

```

X1: {(white small ..... ) dog}
X2: {(black big ..... ) dog}
X3: {(white big ..... ) cat}
X4: {(gray small ..... ) cat}
.....
.....

```

Now, if we want to learn an intermediate concept "mammal", then based on the taxonomy tree, we re-label instances as follows:

The transformed space:

```

X1: {(white small ..... ) mammal}
X2: {(black big ..... ) mammal}
X3: {(white big ..... ) mammal}
X4: {(gray small ..... ) mammal}
.....
.....

```

After this transformation, we may learn classification rules for "mammals" by the same learning method with which we learn rules for dogs or cats.

In the JAUNDICE domain, for example, three diseases "acute hepatitis", "chronic hepatitis", and "hepatic cirrhosis", can be transformed into a common intermediate pathological category, "hepatocellular injury", and the inference rules for "hepatocellular injury" are learned from the same case library by the same learning method (learning confirming rules is described in Section 2.2; learning disconfirming rules is described in Section 2.4) we use to learn the inference rules for each disease.

2.3.1.3. Bidirectional Extension Strategy

This strategy combines the bottom-up and top-down methods to learn more complex hierarchical concepts. Consider a four-level hierarchy as follows:

$$LN \Rightarrow IN_{level1} \Rightarrow IN_{level2} \Rightarrow HN$$

Suppose we have the knowledge of " $LN \Leftrightarrow IN_{level1}$ " and " $IN_{level2} \Leftrightarrow HN$ " (see figure 2.4) and each training instance is described by " $LN \rightarrow HN$ ".

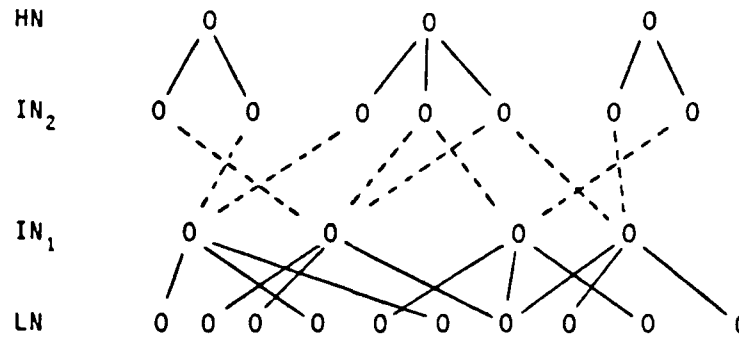


Figure 2.4 Diagram of bidirectional extension strategy: solid links represent existing knowledge, dotted links represent knowledge to be explored.

We can first learn " $IN_{level1} \Rightarrow HN$ " by bottom-up method which exploits the existing knowledge of " $LN \Leftrightarrow IN_{level1}$ ". Then we treat all links of " $IN_{level1} \Rightarrow HN$ " as another set of training instances, and we can learn the knowledge of " $IN_{level1} \Rightarrow IN_{level2}$ " by top-down method which exploits the existing knowledge of " $IN_{level2} \Leftrightarrow HN$ ". Thus, we obtain all inferential knowledge " $LN \Rightarrow IN_{level1} \Rightarrow IN_{level2} \Rightarrow HN$ " from a set of training instances by extending the knowledge of only " $LN \Leftrightarrow IN_{level1}$ " and " $IN_{level2} \Leftrightarrow HN$ ". If we view this learning task as a search, it actually proceeds bidirectionally. Whether the procedure starts bottom-up or top-down does not matter if we assume the training instances are correct and complete. In a domain without uncertainty, consider when inconsistency occurs in the following three instances (H1 and H2 are mutually exclusive): (L1 & H1), (L1 & H2), and (L2 & H1), and assume we have existing knowledge as follows: $L1 \Rightarrow I1$, $L2 \Rightarrow I1$, $H1 \Rightarrow I1$, and $H2 \Rightarrow I2$; then starting with the top-down method, we can first find the following consistent (i.e., no inconsistency) intermediate knowledge: $L2 \Rightarrow I1$, whereas starting with the bottom-up method, we find

no consistent intermediate knowledge. In the current implementation, we ignore such inconsistency.

In the example of four-level hierarchy, we can still obtain the knowledge of all levels by using the bottom-up method alone if only the knowledge of " $LN \Leftrightarrow IN_{level1}$ " and " $IN_{level1} \Leftrightarrow IN_{level2}$ " is available, or by using the top-down method alone if only the knowledge of " $IN_{level2} \Leftrightarrow HN$ " and " $IN_{level1} \Leftrightarrow IN_{level2}$ " is available.

Hence, it is possible to obtain even more complex hierarchical concepts by applying these two methods sequentially, depending on the available knowledge.

2.3.2. Intermediate Concepts not in the Initial Vocabulary

Sometimes intermediate nodes are not known at all, so it is necessary to create and define new intermediate nodes. The key issue is when and how to create new intermediate nodes. Two techniques are introduced: the technique of symbolizing taxonomy point and symbolizing switchover point.

2.3.2.1. Technique of Symbolizing Taxonomy Point

We assume there are n classes of objects or concepts in the instance space. The algorithm proceeds as follows:

- step 1. Construct a taxonomy tree²⁴ on the basis of a similarity or dissimilarity measurement. One way of measuring dissimilarity is based on the of "sum of the differences of weighted individual features". First, based on the domain knowledge, select some important features and assign them weighting factors.

²⁴Taxonomy is "classification" or "clustering": one example of a taxonomy tree is the animal tree seen in Section 2.3.1.2. Two different classes of objects will be put under the same category if they are similar with respect to a certain criterion.

Second, calculate the difference between the average value of an individual feature for the given two classes. If the feature values are not numerical, transform them into numerical values on the basis of domain knowledge. In medicine, this is a feasible approach because the clinical feature values can be quantized according to the clinical severity. (However, in some domains, symbolic measurements may be necessary.) For example, in JAUNDICE, the elevation of the serum enzyme is quantized into 0, 1, 2, 3 representing normal, mildly-elevated, moderately-elevated, highly-elevated. Third, calculate the sum of the differences of weighted individual features. Using different dissimilarity functions (i.e., using different features or different weighting factors) may yield different results. So, it is possible to build more than one taxonomy tree.

Example 2. Computation of dissimilarity between two diseases in a library of four cases. The weights of the two features used are assumed equal here for simplicity.

Instance1: {(GOT 2) (Alk-P 0) Disease A}
 Instance2: {(GOT 3) (Alk-P 1) Disease A}
 Instance3: {(GOT 1) (Alk-P 2) Disease B}
 Instance4: {(GOT 2) (Alk-P 2) Disease B}

(where 0: normal
 1: mildly abnormal
 2: moderately abnormal
 3: severely abnormal)

Compute as follows:

Average value for GOT:
 Disease A: $2+3/2=2.5$
 Disease B: $1+2/2=1.5$

Average value for Alk-P:
 Disease A: $0+1/2=.5$
 Disease B: $2+2/2=2$

Dissimilarity(A & B) = $(2.5-1.5) + (2-.5)$
 $=2.5$

Then, we set up a criterion to group different classes of objects in a common category if their mutual dissimilarity is smaller than a certain threshold. The criterion should be set in a way such that one class will not be grouped in two

different categories. In a taxonomy tree, the tip nodes are n classes of objects (or concepts) which are HN in our terminology.. and each non-tip node (excluding the root node) represents an IN.

- step 2. Assign a symbol to every intermediate node in taxonomy tree, and thus create new intermediate nodes (IN) (see figure 2.5).

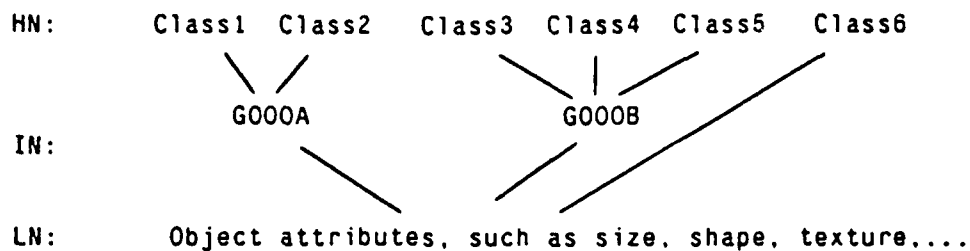


Figure 2.5 Suppose a simple taxonomy tree is built for objects, the intermediate taxonomy points are named as G000A and G000B.

- step 3. The taxonomy tree gives us the knowledge about " $HN \Rightarrow IN$ ". For example, in figure 2.5, "if X is a member of class 1 then X is in category G000A". Therefore, we can apply the top-down method (described previously) and learn the knowledge of the form " $LN \Rightarrow IN$ ".
- step 4. Learn knowledge of " $IN \Rightarrow HN$ " which is the links from IN (or mixed IN and LN) to HN by the following procedures:
 - First, Learn discrimination rules for different classes (HN) under the same intermediate node (i.e., under a higher taxonomical category). For example, in figure 2.5, we may learn classification rules for class 1 and class 2 in the category G000A by removing all objects (instances) that are not in the category G000A. Suppose we obtain a classification rule for class 1 in the category G000A as:

"If an object has attribute A1
then it belongs to class 1."

◦ Second, we actually can write a more specific rule as:

"If an object is in category G000A
and has the attribute A1
then it is class 1."

The algorithm may be applied level by level, and the results will become hierarchical; i.e.,

$LN \Rightarrow IN_{level1} \Rightarrow IN_{level2} \Rightarrow \dots \Rightarrow HN$.

In the JAUNDICE domain, by applying this technique to 72 cases, we found five concepts (see table 2.2). Four symbols, after medical interpretation, were found to correspond to "hepatocellular injury", "cholestasis", "intrahepatic jaundice", and "extrahepatic jaundice". These four symbols had been part of JAUNDICE at one point but had actually been removed from the old vocabulary for the purpose of this test. A fifth term, "hemo-gilb", was found because two diseases, "hemolysis" and "congenital conjugation defect (e.g., Gilbert's disease)" are similar and under the same taxonomy point. Though clinically meaningful (negative bilirubinuria), the symbol "hemo-gilb" bears little pathophysiological meaning.

Table 2.2 New symbols created by the technique of symbolizing taxonomy point and their interpretations.

<u>Symbols</u>	<u>Medical Interpretation*</u>
Neosym1	Hepatocellular injury
Neosym2	Cholestasis
Neosym3	Intrahepatic jaundice
Neosym4	Extrahepatic jaundice
Neosym5	? Hemo-gilb

*: The interpretation depends on the diseases included by the symbol.

Note that this technique is intended to discover new intermediate concepts, but the concepts may have already been in the vocabulary. Hence, after new symbols are created, they should be checked whether they are equivalent to the old symbols semantically. For instance, if both a new symbol "NS000A" and a old symbol "OS000A" include disease A and disease B and cover all the same cases, they are equivalent. Of course, this depends on the size of the case library, so we may want to keep redundant concepts around for a while.

2.3.2.2. Technique of Symbolizing Switchover Point

The development of this technique is motivated by the observation that intermediate concepts often serve as switchover points in a reasoning network. One heuristic rule behind this technique is:

- HR:** If
- i) There are n LNs and m HNs,
 - ii) All n LNs have (confirming) links to all m HNs,
 - iii) $n > 1$ and $m > 1$ and $mn > 4$.

then it is worthwhile to define a common intermediate node.

This heuristic rule is also represented in figure 2.6.

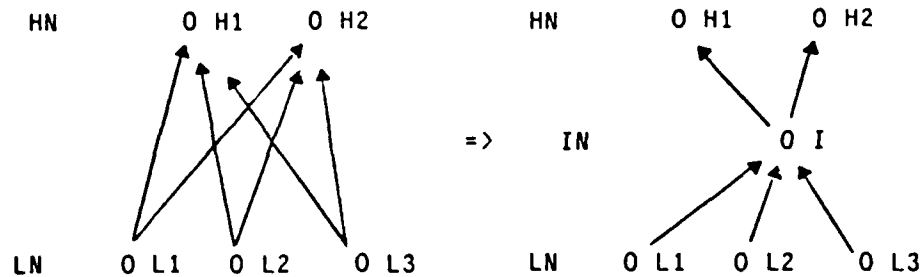


Figure 2.6 Creating new intermediate node at switchover point.

Since if one set of LNs (call it set L) and one set of HNs (call it set H) satisfy this rule, then any subset of set L and any subset of set H can also satisfy this rule, we determine that the intermediate node be defined on the basis of the largest sets (subsets or supersets of set L and set H) of LNs and HNs which satisfy this rule. The third condition of this heuristic rule is, in fact, the threshold of the complexity of the relationship between LN and HN for defining new symbols. We deliberately choose this threshold because of the fact that, for a given situation which satisfies this rule, descriptions of the inference behavior are simplified by adding a common intermediate node while all links from LNs to HNs are maintained via the intermediate node (i.e., no links from LN to HN are added or removed). For example, in figure 2.6, there are 6 links ($LN \Rightarrow HN$) initially and 5 links ($LN \Rightarrow IN$ and $IN \Rightarrow HN$) after introducing an intermediate node. Consider the case when there are 10 LNs and 10 HNs and $10 \times 10 = 100$ links initially; only 20 links are needed after introducing a common intermediate node. But if $n=1$ or $m=1$ or $nm=4$ (e.g., $n=2$ and $m=2$), the descriptions of inference behavior will not be simplified by

adding a common intermediate node. However, the descriptions of inference behavior may become complicated rather than simplified when so many intermediate nodes are introduced and there are overlaps of the associated sets of LNs and HNs among them (note that each newly defined intermediate node has one set of LNs and one set of HNs associated with it). Though complication is worthwhile if more understandability and better accuracy are gained, we might attempt to control the number of the newly defined intermediate nodes (concepts) by adjusting the threshold of complexity for defining them (the third condition of the described heuristic rule).

Creating a new intermediate node will face another problem if uncertainty is involved. For the example shown in figure 2.6, the final degree of certainty of H1 and H2 should remain approximately the same before and after introduction of intermediate concepts. The degrees of certainty (or CF's) are assigned to new links in such a way as to preserve these final degrees of certainty.

In learning from examples, this heuristic rule is applied to a set of rules ($LN \Rightarrow HN$) which are learned from training instances ($LN \rightarrow HN$), and can be applied recursively, as long as there are plausible switchover points, to form a multi-level network. By applying this technique to the jaundice domain, totally there are 9 symbols created, which are shown in table 2.3.

Table 2.3 New symbols created by the technique of symbolizing switchover point and their interpretations.

<u>Symbols</u>	<u>Medical Interpretation*</u>
Neosym1	Benign hepatic pathology ⁺
Neosym2	Cholestasis
= Neosym3	Chronic liver failure
Neosym4	Complete biliary obstruction ⁺
Neosym5	Extrahepatic jaundice
Neosym6	Hepatobiliary pathology
Neosym7	? Liver cachexia ⁺
Neosym8	Inflammation ⁺
Neosym9	Hepatocellular injury

*: The interpretation is made by observing the involved features (LN) and diseases (HN).

+: These symbols are outside the initial vocabulary.

Among these 9 created symbols, 4 symbols are outside the initial (old) vocabulary and 5 symbols are semantically equivalent to some old symbols. It is also noticed that there is some overlapping of the results from the technique of symbolizing taxonomy point and from the technique of symbolizing switchover point. The fact that most of the created symbols are medically meaningful is expected because an intermediate symbol is created when there is a complex but regular relationship between LN and HN (represented by the heuristic rule).

2.3.3. Comparison and Discussion

From the angle of creating new descriptors or concepts, the related work includes EURISKO [Lenat 83], BACON [Langley 83], and [Utgoff 82]. But the difference is our explicit attempt to discover the new intermediate concepts to construct a reasoning hierarchy. However, from the viewpoint of establishing a conceptual hierarchy, the most representative related work in AI is [Michalski 83b]. But it differs from our work in at

least two aspects. First, our work deals with not only conceptual clustering but finding the intermediate links. Because each training instance is also characterized by a high level concept besides low level descriptions, the search for the meaningful intermediate concepts is constrained bidirectionally (from LN and from HN) while this is not true for [Michalski 83b]. Second, though the technique of symbolizing taxonomy points (described previously) intends to discover conceptual clusters, the technique of symbolizing switchover point intends to find important reasoning islands which are more complex than what we call "clusters"; one LN or HN may link to more than one IN and vice versa.

We expect the methods described here can be easily extended to other non-medical domains. In learning intermediate knowledge, we use a general concept hierarchy; and the heuristics we use to discover intermediate knowledge are not specific to medicine. The major contribution of this idea is its capability of learning intermediate-level concepts from a set of training instances that are described only by low level features and high level concepts, and not by any intermediate concept.

2.4. Learning Disconfirming Rules

Disconfirming rules are rules which deny some facts. They can be traced back to MYCIN [Shortliffe 76], in which rules with negative CF are called disconfirming rules in contrast to confirming rules with positive CF. In our scheme, we use degree of certainty (extension from CF) to represent uncertainty. An example of a disconfirming rule is as follows:

$$\begin{array}{ccc} \text{-.7} & & .7 \\ \text{"P"} \Rightarrow \text{A"} & \text{or} & \text{"P"} \Rightarrow \text{-A"} \end{array}$$

This rule says if "P" exists then "A" is denied with the degree of certainty ".7".

There are two basic approaches to form disconfirming rules:

1. From high frequency evidence: If some piece of evidence (clinical manifestations in medicine) frequently appears in a hypothesis (clinical diagnosis in medicine), then the absence of that evidence tends to deny the mentioned hypothesis [Miller, Pople, and Meyers 82].
2. From mutual exclusiveness or incompatibility among facts: If some evidence supports a hypothesis X which is mutually exclusive with another hypothesis Y, then the mentioned evidence tends to disconfirm the hypothesis Y.

For the first approach, in an extreme case, if some evidence appears in a hypothesis under all circumstances, then the absence of that evidence definitely denies the mentioned hypothesis. These are called pathognomonic findings in medicine. This statement may be rephrased as:

$$\text{"P(e/h) = 1"} \quad \Leftrightarrow \quad \text{"P(-h/-e) = 1"}$$

But, if $P(e/h)$ is not "1", then it is not necessary that " $P(e/h) = P(-h/-e)$ "; and each conditional probability depends on the distribution of the evidence among the population of "h" and the population of "-h". It is dangerous to use only $P(e/h)$ to estimate $P(-h/-e)$ unless we know the distribution. It is noteworthy that in MYCIN, the CF, though related to probability, is, however, different from probability in some aspects [Buchanan and Shortliffe 84]. And, it is misleading to use probability to measure directly the degree of belief or disbelief. It is interesting to note that if we use high frequency evidence to form disconfirming rules, the assigned degree of certainty of denying a hypothesis, "-h", by

giving "-e" is parallel to "P(e/h)" rather than "P(-h/-e)". In clinical practice, it is often believed to be true that if a clinical manifestation frequently appears in one disease, then the absence of that manifestation tends to disconfirm that disease. As an example, "SGPT" is always elevated in the disease: "acute hepatitis", and the absence of "SGPT elevation" makes "acute hepatitis" unlikely. In JAUNDICE, another example of a disconfirming rule formed by the first approach is as follows:

"If there is no history of gall bladder disease,
then it is unlikely (-.5) that the disease is
calculous-jaundice"

This rule is derived from the observation that history of gall bladder disease always exists if the jaundice is caused by gall stone.

The issue of overdisconfirmation can be solved by assigning a lower degree of certainty to a disconfirming rule (unless $P(e/h) = 1$). For instance, in JAUNDICE, we use a simple mapping, such as this: if attribute A *always* (corresponding to the degree of certainty in the range: [.8 1)) appears in disease X, then the absence of attribute A *often* (corresponding to the degree of certainty around ".5") rules out disease X. By so doing, confirming rules usually override (to some extent) disconfirming rules to make conclusions if both succeed.

The second approach may be represented as a rule:

"If $e \Rightarrow h1$ and $h1 \Rightarrow -h2$,
then $e \Rightarrow -h2$ "

Sometimes, a disconfirming property can propagate along a relational chain (e.g., causal links), thus:

"If $e1 \Rightarrow e2$, $e2 \Rightarrow \neg e3$, and $\neg e3 \Rightarrow \neg h$,
then $e1 \Rightarrow \neg h$ "

The uncertainty may also propagate; the degree of certainty of a path is the product of the involved links.

Learning disconfirming rules can also focus on a specified case (focus mode). For example, if a case is misdiagnosed as disease B while it should be diagnosed as disease A, then, with the approach from high frequency evidence, disconfirming rules can be formed to disfavor disease B by using feature-values that are absent in the specified case but frequently present in the cases correctly diagnosed as disease B.

2.5. Constructing a Hierarchical Knowledge Base

As mentioned earlier, intermediate knowledge is important for accuracy and understandability in an expert system. This section describes the application of the RL program to constructing a hierarchical knowledge base.

The procedures are as follows:

- step 1. Learn the direct inference rules ($LN \Rightarrow HN$) from the given set of training instances. (learning confirming rules is described in Section 2.2; learning disconfirming rules is described in Section 2.4)
- step 2. Starting from partial or no intermediate knowledge, explore the intermediate knowledge by all methods that include bottom-up, top-down, bidirectional extension, symbolizing taxonomy point, and symbolizing

switchover point, as much as possible (described in Section 2.3). Two things that are expected are: first, some methods may not work because of incomplete knowledge, e.g., bottom-up can't be adopted when knowledge of the form " $LN \Leftarrow IN$ " is missing; second, the results from different methods may be redundant. The first problem is handled simply by abandoning the methods that can't apply. The second problem can be solved by removing redundancy. The symbols created by the techniques, such as symbolizing taxonomy point and symbolizing switchover point, must be interpreted first before checking redundancy with other old symbols and new symbols already created. The interpretation can be made automatically by observing the involved LN and HN (see tables 2.2 and 2.3). At this stage, the knowledge base under construction has the knowledge of three types: $LN \Rightarrow IN$, $IN \Rightarrow HN$, and $LN \Rightarrow HN$.

- step 3. Replace direct rules ($LN \Rightarrow HN$) by intermediate rules ($LN \Rightarrow IN$, and $IN \Rightarrow HN$) if they are equivalent. By "equivalent", we mean the same conclusion (HN) with the same strength (degree of certainty, allowing an error of ".15") can be reached, given a set of low-level features (LN). For instance, in the jaundice domain, a direct rule "negative bilirubinuria and elevated urobilinogen \Rightarrow hemolysis" can be replaced by the rule "negative bilirubinuria and elevated urobilinogen \Rightarrow overproduction of bilirubin" and the rule "overproduction of bilirubin \Rightarrow hemolysis". Note that one direct rule may be replaced by several intermediate rules.

After these procedures, the knowledge base contains hierarchical concepts, but will also contain some simple associations of the form " $LN \Rightarrow HN$ " which can't be explained by the intermediate concepts.

Chapter 3

Feature Condensation

3.1. Introduction

Attempting to build an efficient learning program, as implied by Simon when he pointed out the tediousness of human learning [Simon 83], is justified not only because we want to save time, space, and cost, as desired in all kinds of science, but also because we want to make computers smart enough to learn quickly. In a survey conducted in [Dietterich 83], efficiency is also listed as an important factor for comparing different learning methods. Notice, however, the choice of learning methods can't simply be based on efficiency since this concern may often sacrifice benefit in other aspects. In current AI research on machine learning, the only solution for improving efficiency seems to be employing appropriate heuristics to prune the search space, such as heuristics used in Meta-DENDRAL [Buchanan 78a], "beam width" used to prune hypotheses in INDUCE 1.2 [Dietterich 81], and "window" heuristics used to limit the amount of data to be processed [Quinlan 79].

The work described in this chapter is motivated by finding an efficiency-enhancing algorithm, which is independent of learning methods (i.e., it can be concatenated to any learning system). A program called "CONDENSER" is built to remove irrelevant features (red-herrings) or unrequired features dynamically during learning; this process is

called "feature condensation". By this technique, only the set of required features needs to be considered in learning, thus the dimensions of the search space, which is expanded by the features involved, can be reduced, and the efficiency is improved consequently.

Deciding on which features are relevant or required is a bias, which may have already been assumed to exist (provided by the designers) in some learning works. However, as the bias or ignorance of human designers may negatively influence machine learning, we would rather formulate this issue as a new problem; it may also provide us an understanding of determining the relevant features in learning a new concept. Furthermore, this is a pragmatic issue since we may have already observed that the features involved in certain concept descriptions or certain decision making are often a small subset of all features possibly used in the domain; that is, there do exist some features that are "relevant" in a certain context. Here, by "relevant features", we mean the features can adequately characterize one concept and discriminate it from other concepts. In a domain particularly dealing with decision making, in order to minimize the cost of making decisions, "relevant features" also imply the minimal features required and this implication is maintained in this chapter. Thus, so far as the learning is concerned, it is important to make distinction between "relevant descriptions" and "most general descriptions" since both terms seem to indicate features involved in the descriptions are minimal;²⁵ however, the difference is that "relevant descriptions" can still be as specialized as possible within the given set of relevant features by, for instance, choosing more specific values and not necessarily "most general".

²⁵ Recall the generalization rule #1 (i.e., dropping conditions) in Section 2.1.1, the most general description tends to have minimal features.

The task of CONDENSER may be defined as follows:

- Given:** 1. A set of training instances
(positive and negative instances).
2. Features or descriptors
which describe instances.
- Find:** Features required to describe the training
instances without causing ambiguity.

In other words, CONDENSER intends to remove unnecessary (or irrelevant) features without reducing the power of the learner to make distinctions between positive and negative instances. However, two basic assumptions (or conditions) are necessary to justify this task:

1. There exist adequate training instances, particularly negative instances,²⁶ to guide the condensation process.
2. The set of features required to describe an individual concept should be smaller than (of course, a subset of) the set of features used in the given domain.

Since we discern the relevant or required features by contrasting positive instances with negative instances, the first assumption is necessary. For instance, if a feature *F* is required to distinguish a positive instance *P* from a negative instance *N*, then the necessity of feature *F* wouldn't be recognized but for the negative instance *N*. One might ask what level of adequacy for negative instances is necessary to justify this technique. Since there are an infinite number of negative instances, we might only want to focus on those related negative instances. Thus, for example, in the JAUNDICE experiment, the case library

²⁶Since induction can also be done by using only positive instances [Dietterich 83], here we emphasize the adequacy of negative instances in order to avoid improper use of condensation.

only contains the cases with jaundice as the main manifestation; if one class of cases are labelled as positive instances, other classes of cases are labelled as negative instances; and we think we have adequate negative instances even though we don't have negative instances in other domains. Practically, "adequacy" can be assumed if there is a way to generate all (or most) possible instances and verify them or if there is knowledge (which may be transferred from other similar domains) to tell so.²⁷ Otherwise, it is safer not to assume "adequacy" and thus not to use condensation. The second assumption is to prevent the futile effort of this technique; under this assumption, CONDENSER is expected to remove at least some features. (Refer to Section 3.6.1 for further discussion.)

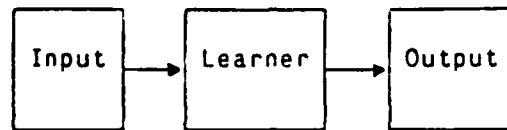
In this chapter, we will describe how, analyze why, and discuss when the CONDENSER program will work. In the terminology of this chapter, "feature base" means a collection of features in a given domain; "feature set" is a set of feature and value pairs representing each instance.

3.2. The Learning System

3.2.1. Structure and Behavior

A learning system is a problem solving system with its own input and output. The system can be either an open loop or a closed loop, depending on whether there is feedback pathway from the output to the learner (see figure 3.1, 3.2.).

²⁷ For example, in medicine, we may consider 30-40 cases as a reasonable sample size for studying a certain medical parameter.



Input: Positive and Negative training instances
Output: Learned concept which is consistent with training instances.

Figure 3.1 A simple open loop learning system.

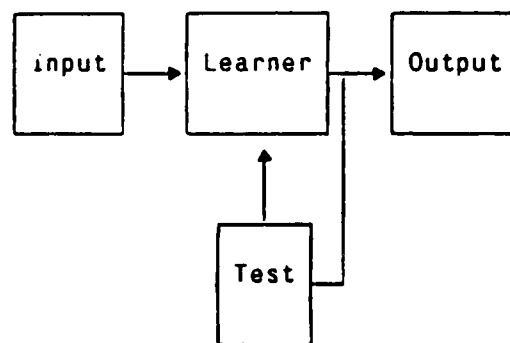


Figure 3.2 A simple closed loop learning system.

In the learning from example paradigm, the input is a set of training instances, which is constructed by some elaboration, including selecting good data and representing them properly. The learner can be model-driven or data-driven, and it seeks concept descriptions which are consistent with the training instances and human background knowledge. The output are the learned concept descriptions, which have been

transformed into human understandable forms. In a closed loop learning system, the learned concept descriptions undergo repeated tests to determine if most of the positive instances are covered and negative instances are rejected. A closed loop system because of these tests, which are part of the error-adjusting process (described in Chapter 4), will yield better results.

3.2.2. CONDENSER

We propose a more sophisticated learning system, diagramed in figure 3.3, which incorporates CONDENSER and the noise-filter; the former is described in this chapter; the latter will be explored in Chapter 4.

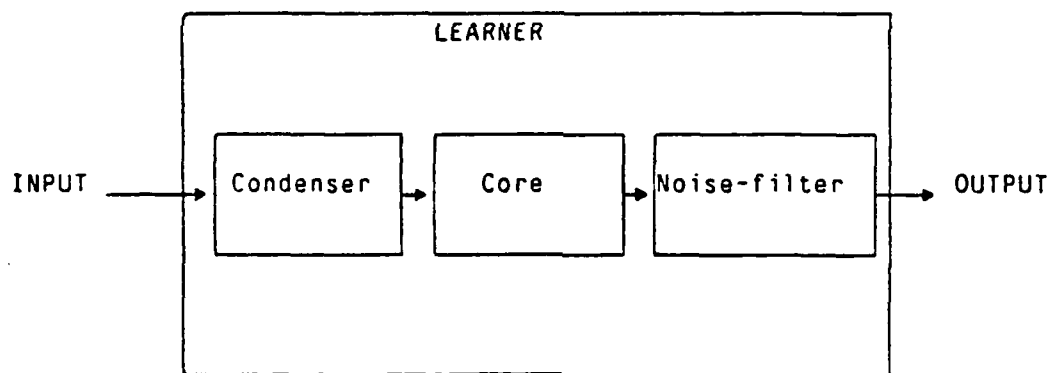


Figure 3.3 Diagram of an efficient and noise-resistant learning system with the condenser and the noise-filter.

In brief, the function of CONDENSER is to remove unnecessary descriptions. For instance, to distinguish dogs from cats, it is not necessary to mention that both are animals; however, to tell dogs from trees, it is useful to mention the former are animals and the latter are plants.

CONDENSER, a part of the learner, fulfills its role dynamically, depending on what class of concept to learn. One might wonder why the data have to be manipulated dynamically. The main justification is that the case library is a dynamic (time varying) structure; the statistics might shift with time; therefore determination on the set of required features for a certain class of concept on the basis of the case library at a specific point on time axis may deprive the learner of learning new knowledge and detecting the faults of old knowledge. Thus, from a long term perspective, it is warranted to do it dynamically.

3.3. The Rule of Condensation

Generally, the rule is the following: "while preserving the desired information, simplify the representation as much as possible". Specifically, the *condensation rule* can be stated in the following two aspects:

- The descriptions of a given concept can be condensed by removing some features (or descriptors) as long as the number of negative instances included because of this operation is less than the pre-set threshold. In this formulation, it implicitly assumes the given concept divides the instance space into positive and negative instances.
- Suppose the learner intends to learn about a new concept from already classified instances, then the descriptions about instances can be condensed by removing some features (or descriptors) as long as the number of negative instances made indistinguishable from positive instances by this operation is less than the pre-set threshold. In this formulation, condensation will proceed under the assumption of a given dichotomy (positive and negative instances). Note that an instance space might contain more than two mutually exclusive categories; if one category is treated as positive instances, then the other categories will be treated as negative instances. A different dichotomy will result in different condensation.

We can also represent concept descriptions by a feature set, a set of feature and value pairs that characterize the concept. For example, we may represent a red big round object by `{{(color red) (size big) (shape round)}}`. Then, the objective of condensation can again be formulated in two aspects:

- For a given concept description, find the required or relevant descriptions about the concept.
- For learning concept descriptions, find the set of features that adequately but not redundantly describe the training instances.

Example 1. Consider there are only three instances in an instance space.

```
pos.1: {(color red) (size big) (shape round)}
neg.1: {(color red) (size small) (shape cubic)}
neg.2: {(color black) (size small) (shape round)}
```

(Note: pos.= positive instance,
neg.= negative instance.)

Now, if the threshold is set to zero, i.e., no negative instances should be included or confused with positive instances by condensation, then,

Justified condensation:

```
pos.1: {(size big)}
neg.1: {(size small)}
neg.2: {(size small)}
```

Condensation is justified because positive instances are still distinguished from negative instances.

Unjustified condensation:

```
pos.1: {(color red)}
neg.1: {(color red)}
neg.2: {(color black)}
```

Condensation is unjustified because distinction is lost between pos.1 and neg.1.

In example 1, suppose we know the concept description is `{{(color red) (size big)}}`, then it can be condensed into `{{(size big)}}` by the condensation rule.

Now, we define the term "incompressible" as follows:

- An "incompressible concept description" is such that removal of any feature (descriptor) from it will cause more than allowed number of negative instances included: in other words, the condensation rule is violated.
- An "incompressible feature base" is such that removal of any feature from it will cause inadequacy in describing positive instances. Again, it depends on what we include as positive instances.

To what extent features can be removed depends on the presence and nature of negative instances. In the extreme case, there exist no negative instances, then all features can be removed, and the description become "null", i.e., all instances are positive instances. Therefore, one implicit assumption behind the "condensation rule" is that there should be adequate negative instances to justify "condensation".

Assertion 1.: The concept descriptions learned from the instances described by condensed features are still consistent with original instances.

<argument>: Adding more features is a kind of specialization. If concept descriptions can distinguish positive instances from negative instances with condensed features, it still can if all features are preserved. In Example 1, the concept description "{(size big)}" is consistent with instances after justified condensation; so it is still consistent with the original instances (before condensation).

3.4. Techniques of Condensation

Condensation can proceed by three strategies:

1. Exhaustive search: Test every possibility. So, if there are "m" features, the number of possibilities is " 2^m ". Because the search space is huge, it will not be used practically.

2. Search by Tree: Starting from the collection of all features, the tree is expanded by removing one feature at a time in all possible ways. In figure 3.4, assume there are 4 features to describe instances: the tree is expanded sequentially as described.

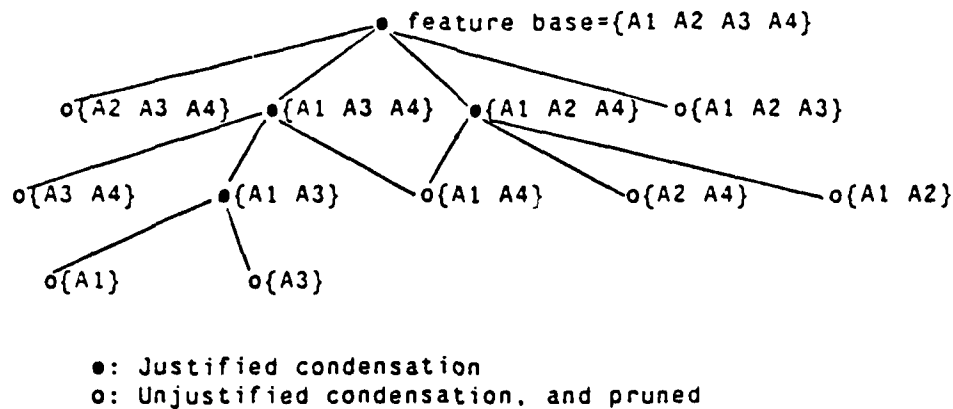


Figure 3.4 Condensation by search tree. In this illustration, there are four features in the original feature base.

Once the generated condensation is not justified, it is pruned, because subsequent condensations will also be unjustified, based on the consideration that dropping more features will cause negative instances even more undistinguished from positive instances. This strategy is a heuristic one in the sense that it can avoid exhaustive search. Nonetheless, it is still expensive. A somewhat similar strategy was applied to some works of pattern recognition [Becker 78].

3. Ordered Scanning Algorithm: See next section.

The program CONDENSER uses *only* "ordered scanning algorithm" for the reason of efficiency, as will be analyzed in Section 3.5.

3.4.1. Ordered Scanning Algorithm

Features are first ordered according to their priority (significance), which may be evaluated along different dimensions. In a domain like medicine, the cost of feature measurement (e.g., laboratory examination) is an important factor besides discriminating ability to determine the priority. The most desirable feature (with the highest priority) is the one with the highest discriminating ability (pathognomonic features in medicine) and with the lowest cost of its measurement. Then, features are scanned one by one to detect which features can be removed on the basis of the *condensation rule*. The priority of a feature can be determined by background knowledge or past experience or statistical techniques. If statistical techniques are used, instances are randomly sampled, and then, based on the distribution of feature values, we will be able to determine the discriminating ability of the feature. For a numerical feature, it is often true that the further apart the mean value (also the value with peak frequency in Gaussian distribution) of a feature between positive and negative instances, the better its discriminating ability.

The algorithm proceeds as follows.

- step 1. Order features according to increasing priority (significance) described above by placing a feature with higher priority behind a feature with lower priority, and thus build a list of features.
- step 2. Scan the ordered list, starting from the head, and proceeding down the list. Try to delete one feature at a time.
 - substep 1. If the deletion violates the condensation rule, then put the feature back on the list without breaking the ordering.
 - substep 2. Otherwise, delete it from the list.

In substep 1, in order to decide whether the condensation rule is violated, we set four

thresholds as follows: GATP (global ambiguity threshold for positive instances), LATP (local ambiguity threshold for positive instances), GATN (global ambiguity threshold for negative instances), LATN (local ambiguity threshold for negative instances). If one positive instance is indistinguishable from one negative instance (i.e., they share common or equivalent descriptions), they are "ambiguous". GATP is the maximal number of total positive instances made ambiguous because of this condensation operation. LATP is the maximal number of positive instances made ambiguous by deleting one individual feature. GATN and LATN are counterparts of GATP and LATP for negative instances. Therefore the operation is constrained both globally and locally in order to condense the feature base properly. Though ideally we may set all these four thresholds to be zero, practically this may not be the case owing to the imperfectness associated with the instances, e.g., false positive or negative training instances. Notice, however, GATP and LATP are somehow related to false negative predictions, so are GATN and LATN to false positive predictions: the extent depends on how the learner handles the ambiguity.

The program CONDENSER, which is our implementation of the ordered scanning program, receives (from the learner) input, which comprises a feature base to be condensed and a set of instances that have been labelled either positive or negative instances by the learner according to the class of concept to be learned; and the output is the condensed feature base. Then the learner will consider *only* features in the condensed feature base when it expands the search space (also refer to Section 2.2.1) and when the concept descriptions (or hypotheses) are matched to the training instances. Thus, during the learning cycle, training instances look as if they are represented only by the condensed features.

Example 2. Consider 3 instances in the instance space,
and there are 4 features to describe them.
Assume the four thresholds described are all set to zero.

```
pos.1: {(color red) (size big) (shape round) (weight heavy)}
neg.1: {(color red) (size small) (shape round) (weight heavy)}
neg.2: {(color red) (size big) (shape cubic) (weight heavy)}
```

If we consider the discriminating strength, feature
"size" and "shape" have higher priority than others.
So, we build a features list as the following:

```
(color weight size shape)
```

Then, the algorithm proceeds as follows:

```
removing "color" -> succeed! the condensation rule
                    is not violated.
removing "weight" -> succeed!
removing "size" -> fail! pos.1 and neg.1 are
                    indistinguishable. So, put
                    it back on the list.
removing "shape" -> fail! pos.1 and neg.2 are
                    indistinguishable. So, put
                    it back on the list.
```

As a result, the condensed feature base is as follows:

```
(size shape)
```

Assertion 2.: A feature base condensed by "ordered scanning algorithm"
is incompressible.

<argument>: Since, during the scanning procedure, all removable features (the removal
of which does not violate the condensation rule) are removed, further removal will cause
violation. Thus, the result is incompressible.

The results may or may not depend on the ordering; in the above example, the result is
independent of the ordering. Since there may be several different ways of ordering the
features, we might maintain several versions of the results; each version bears a specific

meaning. Thus, for instance, in medicine, one set of rules might involve safe but less accurate clinical features; another set might involve invasive but more accurate features; and so forth. However, in general, there are limited criteria based on the background knowledge to determine the ordering.

Another issue is "overcondensation", which results from inadequacy of negative training instances for constraining the condensation. Asking whether the training instances are adequate in learning is somewhat similar to asking whether the samples are adequate in statistics. Nevertheless, in learning, the training instances should be adequate not only in quantity but also in quality. *Near-miss*²⁸ negative instances are crucial to identify important features and thus are important for condensation. Near-miss instances can be generated by replacing values in a small number of features of the positive instances and then verified by experts or by visiting an instance library or by other methods. However, it is impossible to generate any near-miss instance we want unless it can be verified. Adding some background knowledge is another solution to relieve overcondensation. Though it is possible to create another program which can add more features in to remedy this problem, this is however not a desirable approach because the system may be trapped in the dilemma, deciding whether to delete or to add features. Rather, we would make sure whether we have already had adequate training instances in advance of applying CONDENSER: as a matter of fact, we have made this an assumption underlying this operation.

²⁸ Near-miss, as defined by [Winston 70], is a negative instance which differs from positive instances in only a small number of features.

3.5. Why Does CONDENSER Work?

The effect of introducing CONDENSER in a learning system is investigated on the basis of the cost and benefit as follows:

- *The cost is linear with the size of the feature base.* The ordered scanning algorithm performs one dimensional search instead of multiple dimensional search (e.g., tree search). For instance, if there are m features in the original features base, then, by means of "ordered scanning algorithm", m scans are required. In each scan, positives instances are matched against negative instances. So, if there are " p " positive instances and " n " negative instances, then in total " $m \times n \times p$ " times of matching are done. The time required for matching may be increased with the number of features, but the increment will be less than first order because of the following considerations. First, if two instances are not matched, the mismatch will be detected once one feature is not matched, and the matching process is terminated. Second, the matching will get simpler as more features are removed. Third, the sequence of features in the matching process is based on priority too (match more important features first); therefore mismatch is earlier to be detected. If condensation is based on an individual positive instance and its near-miss, and take the union as the result, then it will be even more economical, and the cost is reduced to " $m \times n$ " \leq cost $\ll m \times n \times p$ ". From this simple analysis, we know the cost spent in CONDENSER roughly increases linearly (though may be slightly higher than first order) with the number of features in the original features base. To demonstrate the computational near-linearity of CONDENSER, we labelled 11 acute hepatitis cases as positive instances and other cases as negative instances, and measured the time required for CONDENSER by means of ordered scanning algorithm to condense feature bases with different size as follows: 10, 20, 25, 30, 35, 40, 45, 50, 55, 60; despite their size, they all contain required features to describe positive instances (acute hepatitis). The result is shown in figure 3.5.
- *The benefit is nonlinear with the size of the feature base.* Before we analyze the benefit, we first estimate the size of the search space in learning. The search space is in fact the power set of the set formed by collecting all features (or their values) which are used to describe training instances. So, if the number of total feature values is " m ", the size of the search space is " 2^m ". To reduce

the number of features can thus reduce the search efforts greatly. But thanks to heuristics applied, the difficulty of learning will not grow exponentially with the number of features involved: still nonlinearity (beyond first order) is observed in general. To demonstrate computational nonlinearity of the learner, we measure the learning time by applying the learning method described in Section 2.2 to learn rules for diagnosing the disease of acute hepatitis from the case library with feature bases of different size. The result is shown in figure 3.6.

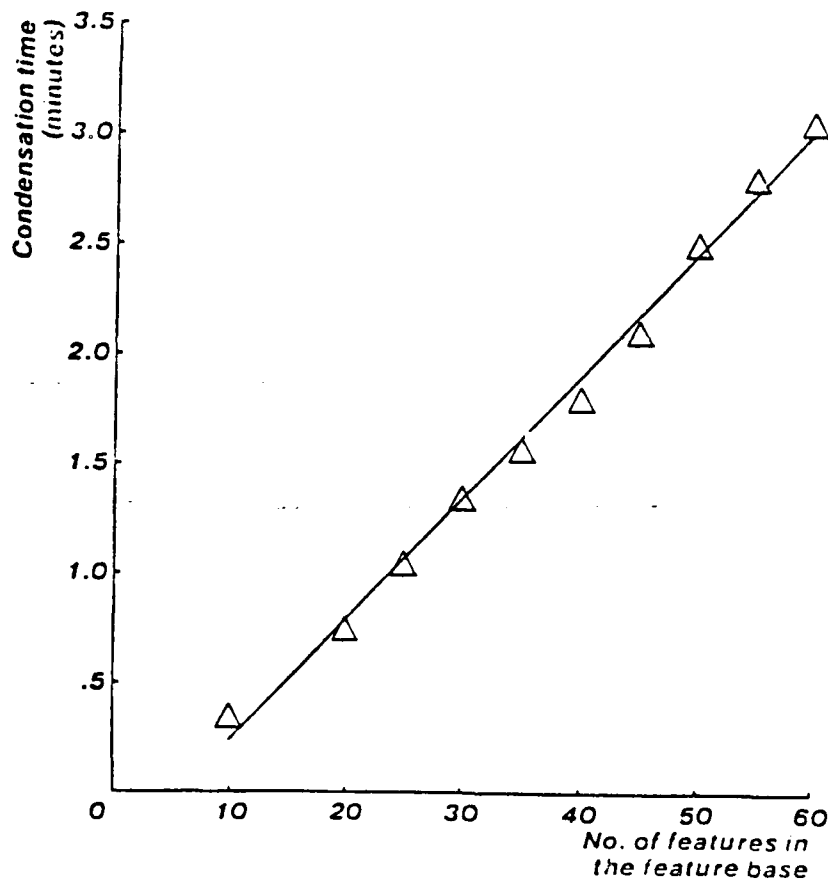


Figure 3.5 Linear effect of the size of the feature base on the time used in CONDENSER.

AD-A171 794

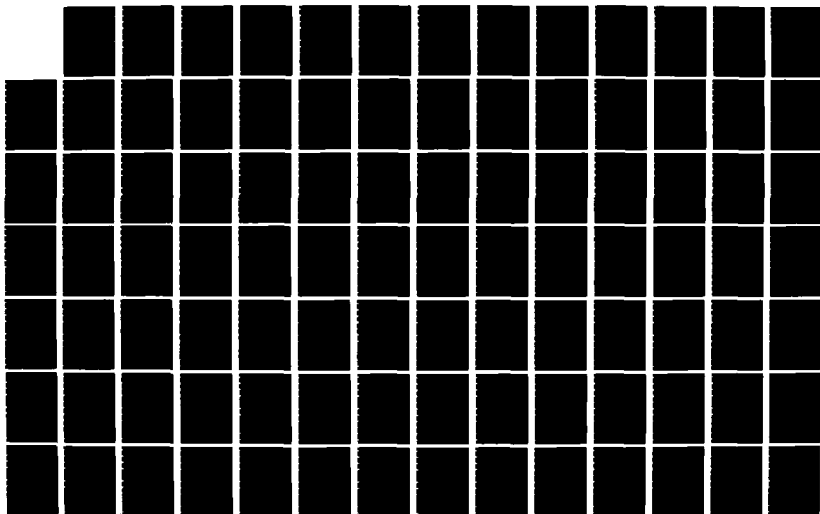
LEARNING OBJECT-LEVEL AND META-LEVEL KNOWLEDGE IN
EXPERT SYSTEMS(U) STANFORD UNIV CA DEPT OF COMPUTER
SCIENCE L FI NOV 85 STAN-CS-86-1091 N00039-83-C-0136

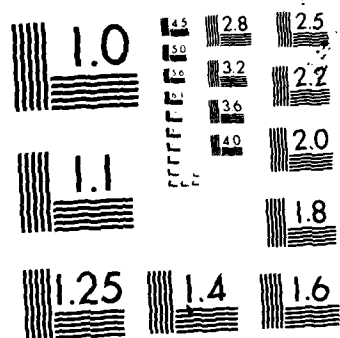
2/3

UNCLASSIFIED

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

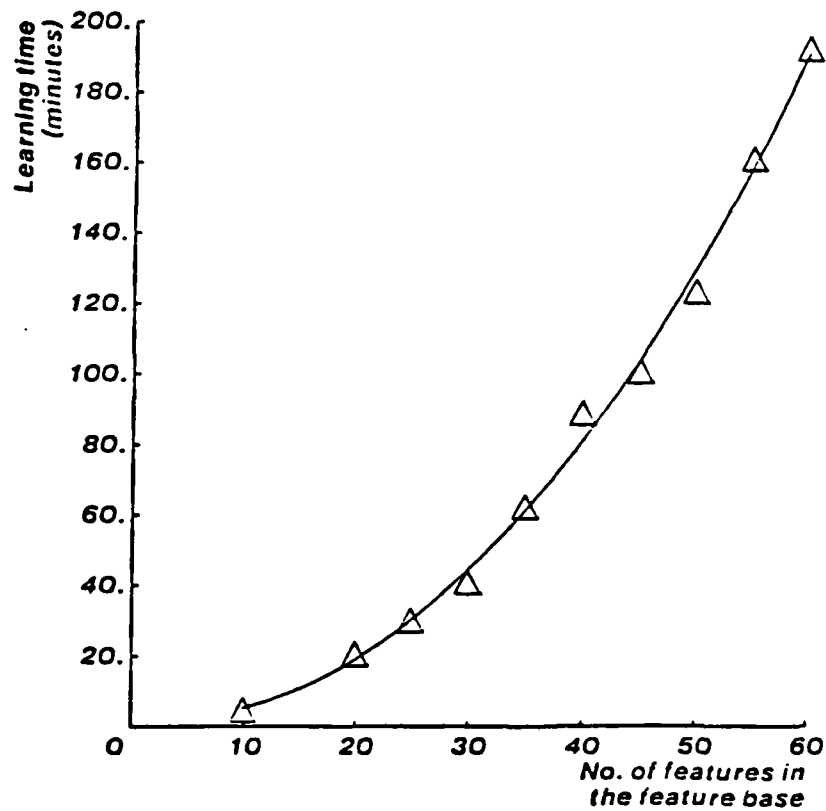


Figure 3.6 Nonlinear effect of the size of the feature base on the time used in the learner.

With respect to the number of training instances, the computation time spent in one single (non-disjunctive) concept learning task, according to Mitchell [Mitchell 78], is proportional to " $(p+n)^2$ " in depth-first search, " $p \times n$ " in breadth-first search, and " $p+n$ " in candidate elimination technique (where " p " is the number of positive instances, " n " is the number of negative instances). However, in multiple disjunctive concepts learning (a typical situation

in EMYCIN-based systems), another nonlinear factor because of combinatorics should be considered: for example, a factor related to maintaining multiple version spaces in [Mitchell 78]. In contrast, the time required for CONDENSER is proportional to, as described earlier in this section, somewhere between " n " and " pxn " in both single and multiple concept learning tasks.

Thus, the cost spent by CONDENSER is linear with the size of the feature base and comparatively small with the size of instance space while the benefit gained with respect to the size of feature base is nonlinear. It follows that incorporating CONDENSER can enhance the efficiency of learning. To demonstrate this analysis, we again applied the same learning method to learn rules for diagnosing the disease of acute hepatitis from one half of the case library with 72 cases, then we used another half of the case library to test the KB constructed by replacing the old rules associated with the diagnosis of acute hepatitis by the learned rules in the old KB with 141 rules. The result is shown in table 3.1 Without CONDENSER, constructing a new KB in JAUNDICE by learning all concepts (there are ten concepts in JAUNDICE) from the same 72 cases takes about 14 hours whereas, with CONDENSER, it takes only 45 minutes; and the qualities measured by the diagnostic accuracy over 42 clinically diagnosable cases (unrelated to the above 72 cases) which received liver biopsy in Stanford Medical Center in 1978 are the same (83.3%) (also refer to Table 7.2).

Table 3.1 Comparison of the performance of two learning networks with and without Condenser.

	time (min.)	No. of rules learned	Diagnostic accuracy* (36 cases)
with Condenser	condensation 2.5	29	31/36
	learning 5.4		
	total 7.9		
without Condenser	learning 192.2	30	31/36
	total 192.2		

*: Rules learned from one half of the case library with 72 cases are tested against the other half.

The result indicates CONDENSER can save a significant amount of learning time while preserving both quality and quantity of learned information. It also implies that a learning process can be decomposed into two parts: discovering the set of required or relevant features, and the process of generalization and specialization. Great efforts can be saved if we solve them separately (divide and conquer, so to speak). This result is compatible with the above analysis.

3.6. Application

3.6.1. Applicable Domains

We recapitulate the benefit gained from incorporating CONDENSER in learning systems as follows:

1. Efficiency of learning can be greatly improved.
2. Decision rules learned in such systems can be precluded from carrying unnecessary risk and cost since unnecessary features are removed.

From this perspective, any domain is a good candidate to be applied. However, CONDENSER might act adversely in the following circumstances:

- If the features used to describe the training instances are all relevant or required, CONDENSER will have no effect.
- If no adequate (see Section 3.1 for the discussion of "adequacy") negative training instances exist, then overcondensation will occur and make the learned concept descriptions too general and tending to cause false positive predictions which may also carry undesired risk and cost.

Considering the second circumstance, we seem to face a tradeoff between simplicity and fidelity, a issue which is also emphasized in data compression in engineering science (e.g., [Gray 74], and [Blasbalg 62]). Fortunately, the tradeoff will exist only under this imperfect condition: if we have adequate negative instances to monitor condensation, we may actually achieve both simplicity and fidelity. In the introductory remarks, we set forth two assumptions which are required to establish the practical value of CONDENSER; the assumptions are merely the opposite of the above adverse conditions. Despite these, there is still room for arguments; for example, how to verify whether the assumptions hold. However, we can still tell simply based on background knowledge or

intuition. For instance, in medicine, many routine examinations, which may not be necessary except possibly for legal reasons, create a large amount of data in patients' records, but the clinical features related to diagnosing a certain disease are often a very small set of all clinical features. Therefore we may expect the usefulness of CONDENSER in medicine.

Herein, we define CONDENSER Utility Index (abbreviated as CUI) as follows:

$$\text{CUI} = \frac{\text{No. of features in the feature base}}{\text{No. of features in target concepts}}$$

The higher the index, the better indicated the CONDENSER. The threshold of CUI such that CONDENSER will be beneficial is " ≥ 1 ", since the objective of CONDENSER is to remove some irrelevant features; the exact value needs to be calibrated in different domains.

In summary, the applicable domains are domains where the two assumptions hold; practically, we might anticipate such domains to bear the following features:

1. Each case or instance in the domain is rather complicatedly described. Even if we don't know whether the new concept is simple, it may still be worthwhile to try condensation. Remember that scientific rules or principles are often simple.
2. Training instances can be generated and verified; if not, then somehow there is knowledge to judge whether the instances are adequate.

3.6.2. Compatible Learning Systems

What types of learners are required for CONDENSER to work? Generally speaking, CONDENSER can be effective for any learner if the time used in the learner increases nonlinearly (higher than first order) with the number of features in the feature base. The rationale is again based on the fact that CONDENSER performs one dimensional scanning and its cost is about linear, as indicated in figure 3.5. In figure 3.6, it is demonstrated that the time used in the learner with the learning method described in Section 2.2 is about quadratic, and significant improvement is acquired by incorporating CONDENSER, as seen in table 3.1. In version space approach [Mitchell 78], reduction of the number of features will accelerate convergence upon the desired concept description. With such an approach, in a perfect learning environment, the learning time may not increase rapidly with the size of the feature base while, in imperfect situations (where inconsistency occurs), the time will become nonlinear owing to maintaining large boundary sets. In INDUCE 1.2 [Dietterich 81], the algorithm restrains the hypothesis space under a constant width ("beam width") during each expansion of the search space and results in an incomplete search. CONDENSER, by removing unnecessary features, can thus relatively broaden the "beam width" because of reduction of the search space and render the search more complete in such a system. ID3 [Quinlan 83] is similar to ordered scanning algorithm in that they both order features based on some criteria. The difference is that ID3 does not remove features, and each time a decision node is constructed, the system, performing best-first search, examines all remaining features to determine which feature can provide maximal information for classification, based on the decision tree so far constructed. So, CONDENSER has two possible applications in such a system: first,

it may condense the decision tree into a more compact form, secondly, since CONDENSER may discover more than one set of required features based on different criteria of priority, more than one decision tree associated with different meaning may be built.

In particular, the learner designed to learn multiple disjunctive concepts (e.g., in EMYCIN-based systems) will be greatly benefited from CONDENSER because of the horrendous combinatorics of features and instances.

3.7. Comparison and Discussion

From the idea of improving efficiency of learning, as described before, some learning programs employ heuristics to prune the search space. For example, in Meta-DENDRAL [Buchanan 78a], an "improvement criterion" is used to determine the relative plausibility between a parent chemical environment and its successors and thus guide pruning the search space. In INDUCE 1.2 [Dietterich 81], "beam width" is used to prune hypotheses, and search becomes incomplete. However, inasmuch as the feature condensation technique is intended to represent dynamically the training instances (a set of data) as simple as possible so long as not much information is lost, this work can actually be applied by the term "data compression". Thus, if we view from data compression, two learning programs may be related. The first one is again Meta-DENDRAL, in which the INTSUM program compresses data by the aid of "half order theory" during constructing the instance library; or, in other words, the half order theory is exploited to make the data interpretation more efficient and accurate. The second work is [Quinlan 79], in which a "window" is used to handle a large volume of data; only the

training instances in the window are processed. But no work has ever mentioned how to remove irrelevant features during learning, perhaps, because all other learning works assume either the provided features for the learning system are all relevant or the process of determining the relevant features is actually done during the process of generalization or specialization. Whatever assumptions are made, it is worthwhile to separate this process out and do it efficiently, as demonstrated in CONDENSER. Another point is that the condensation will generally not affect the completeness of the search in learning, as implicated in table 3.1, since the objective is to remove only the irrelevant features. Furthermore, the idea of feature condensation is one example of generating automatically the proper bias on the descriptive language, instead of being provided by human designers, to enhance the performance; this perspective also includes how to create a new language to relieve the bias imposed by a fixed language (as suggested in [Utgoff 82]).

In systems, such as communication, image processing, pattern recognition, and so on, there are works (e.g., [Gray 74], [Blasbalg 62], [Becker 78]) which, though related in the idea of data compression, bear little similarity to the herein developed method from a methodological viewpoint. First, instead of using mathematical techniques, the developed method employs a symbolic technique to match positive instances against negative instances to determine dynamically the relevant features with respect to the learning task. (Note that it is possible that all features are relevant if we consider all learning tasks; however, for a specific task, only some may be relevant.) Secondly, the one dimensional scan based on heuristics- or knowledge-based priority in the ordered scanning algorithm finds no counterparts in these areas.

3.8. Summary

The main goal of this chapter is to build an efficient learner. We solve this problem by developing a symbolic technique of feature condensation.

The role of the CONDENSER program is validated by some analysis and simple demonstrations. CONDENSER is designed for general domains, and should be able to be connected to any learning system. Though domain specific modifications are required, the principle will hold. The reason why CONDENSER will work may be boiled down to some simple facts that CONDENSER performs one dimensional search (by means of ordered scanning algorithm) while the learner performs multi-dimensional search.

Chapter 4

Learning in Noisy Environments

4.1. Introduction

In inductive concept learning (learning from examples), one might ask "what if the training instances are incorrect?" In this chapter, we will investigate possible error-causing factors (called "error-sources") and provide solutions to handling them.

The objective of learning is to find concept descriptions or rules that are consistent with all (or most) instances in a given domain. However, practically, instead of exhaustively using all instances (impossible anyway), we use a set of training instances and anticipate the results learned from this training set can be applied to all other instances as well. It seems logical to ascribe the errors associated with the results to either the training instances or the learning system, or both. In current AI research on learning, the error-sources which have been addressed include the following:

1. Incorrect training instances, including false positive or false negative instances ([Buchanan 78a], [Mitchell 78], and [Dietterich 83]).
2. Inappropriate bias embedded in the learning algorithm or the descriptive language ([Mitchell 78] and [Utgoff 82]).

It seems justifiable to stick to the division of error-sources into two main categories: input and the system, as diagramed in figure 4.1.

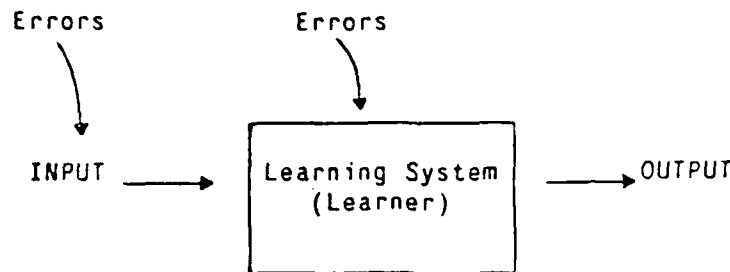


Figure 4.1 Diagram of learning with sources of errors shown.

All possible error-causing factors that are enumerated here are based on AI researchers' concern (as mentioned above), general background knowledge (e.g., sampling insufficiency may provide incorrect statistics), and the experience during our experiments. Some trivial factors, such as bookkeeping errors, however, are not considered.

The motivations of this chapter comprises the following two aspects:

- Error handling in AI learning is seldom addressed [Dietterich 83]. Though researcher have already begun to solve this issue, e.g., maintaining multiple version spaces [Mitchell 78], RULEMOD in Meta-DENDRAL [Buchanan 78a] (as will be described in more detail in Section 4.7), the solutions are discrete. This chapter is intended to establish an unified framework for handling errors in a noisy learning environment, the framework which not only generalizes but also amplifies the old approaches.
- We also intend to provide people a notion of "learning is a mixture of search and optimization under an imperfect learning environment". "Optimization" denotes "achieving the best result"; in AI learning we are concerned about, it means the learning results are maximally consistent with the instances in a given domain.

In order to achieve the best result, we further divide the task into two successive stages. The first stage is to find concept descriptions that are maximally consistent with the training instances; the second stage is to update the descriptions so that they can be maximally consistent with all new instances other than the training instances. However, we only aim at the first stage problem here, and leave the second stage problem in other areas of this thesis (refer to section 2.2.4 for "focusing" mode of learning and Chapter 5 for automated debugging). Notice, however, unless the result of the first stage learning problem is desirable, we might not even intend to solve the second stage problem. In this chapter, we first declare two basic assumptions which we think are necessary in some sense as follows:

- The basic framework in the learning system is maintained; i.e., we assume there is a proper representation and proper descriptive language because, at this stage of development of machine learning, it has not yet been possible to build or reorganize this basic framework by machine; though some work has begun to explore this issue, e.g., [Lenat 83].
- There is a set of training instances and most of them are correct and complete. With this assumption, the concept descriptions that are maximally consistent with the training instances are anticipated to be consistent with most of instances in the given domain, though a limited amount of editing is still required.

The maximally consistent state can be achieved by an optimization technique, which, in AI, is "hill-climbing search" under the assumption that we start from a plausible point.

In this chapter, we begin with descriptions about all possible error-causing factors in a reasonable depth to provide a general and adequate understanding of this issue. Then, a general method is developed to maximize the consistency of the learned concept descriptions or rules among the training instances. In terminology, "error-source" denotes

any imperfectness (not necessarily errors) which is associated with either the input (the set of training instances) or the learning system and may cause "error" in the output (the learned concept descriptions or rules).

4.2. Imperfect Training Instances

We classify the causes of imperfect training instances into two main categories: inconsistency and incompleteness. Since "incompleteness" may also lead to "inconsistency", to avoid redundancy or confusion, "inconsistency" denoted here excludes "incompleteness".

4.2.1. Inconsistency of Training Instances

Inconsistency can be further divided into "spontaneous" and "non-spontaneous (or artifactual)" inconsistency; the former connotes the inherent overlapping between positive and negative instances with respect to certain features; the latter denotes those human-responsible factors, and we only describe the most important one in induction: false positive and false negative training instances.

4.2.1.1. Spontaneous Inconsistency

Figure 4.2 shows the frequency distribution of positive and negative instances with respect to a certain numerical feature.

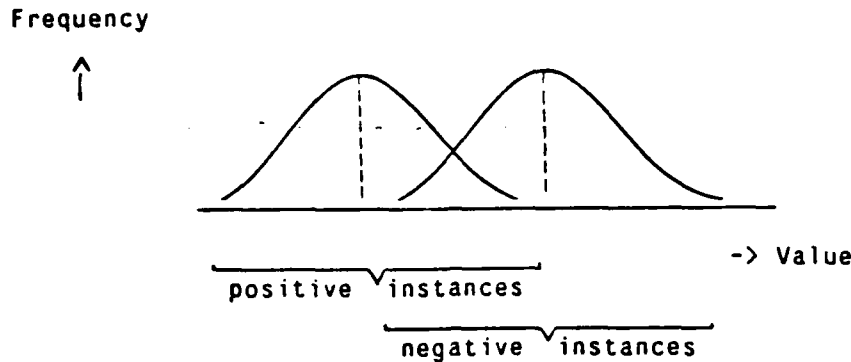


Figure 4.2 Overlapping of positive and negative instances.

Though the values with peak frequency of two distributions are separate, there is always some degree of overlapping. The wider the spread (owing to larger variance or standard deviation), the greater the overlapping. Because of this noise, uncertainty is involved in reasoning. Historically, ways of handling uncertainty include as follows: probabilistic reasoning, fuzzy sets theory [Zadeh 65], certainty factors [Shortliffe 76], etc. For instance, in medicine, the statement "95% of upper respiratory tract infection is caused by virus" includes a probabilistic factor "95%". As stated by [Hahn 29], "all knowledge originating in experience comes with a coefficient of *uncertainty* affixed to it". In electrical engineering, "stochastic" means "involvement of uncertainty".

In learning from examples, even if we have a perfect set of training instances and a perfect learning algorithm, we may still not find an ideal concept which is consistent with all instances because of this natural uncertainty involved in the domain. Consequently, the objective is to find concept descriptions which are consistent with as many instances as

possible. In figure 4.2, obviously there is no clear-cut boundary between positive and negative instances; if the chosen cut-off point shifts rightward, there will be more negative instances falsely believed to be positive instances (called false positive predictions); similarly, shifting the cut-off point leftward will cause more false negative predictions. In fact, there is a trade-off.

4.2.1.2. Incorrectly Classified Training Instances

False positive training instances are negative instances falsely classified as positive instances; false negative training instances are positive instances falsely classified as negative instances. Data-driven learning methods (e.g. Version space algorithm [Mitchell 78]) are particularly susceptible to this type of noise. One false positive instance will cause excessive generalization of the concept, as seen in figure 4.3. And one false negative instance will cause excessive specialization of the concept, as seen in figure 4.4.

Model-driven learning methods (e.g. Meta-DENDRAL [Buchanan 78a]) are superior in escaping this type of noise because there exist global criteria (which measure the consistency over the instances) for selecting hypotheses generated by the models, and the instances are not considered individually. Since the methods intend to find the most consistent concept descriptions or rules, falsely classified instances will somehow be ignored if they are the minority.

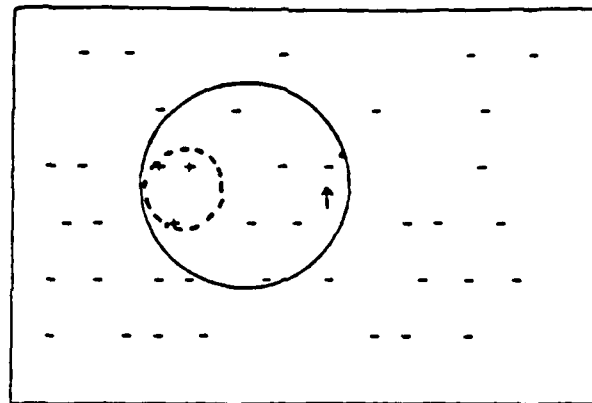


Figure 4.3 Excessive generalization caused by a false positive instance.

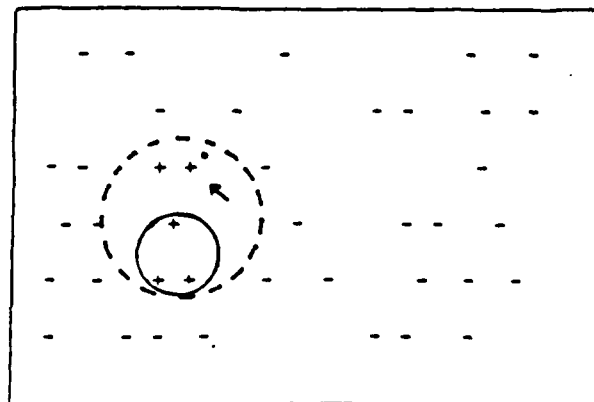


Figure 4.4 Excessive specialization caused by a false negative instance.

4.2.2. Inadequacy of Training Instances

4.2.2.1. Incompleteness of Data

Sometimes, ambiguity between a positive and a negative instances arises because of incomplete descriptions about the two instances. In medicine, incomplete data about a patient will cause confusion, and more data obtained may switch the disease diagnosis to a totally different one.

Learning under the condition of incomplete data, though not desirable, may sometimes be unavoidable because of the difficulties encountered in obtaining the missing part of the data. For example, there is no way to obtain some desirable laboratory data from an expired patient.

Learning based on incomplete data will also yield an inconsistent result. This is due to the fact that the incompleteness of data may cause inconsistency or ambiguity between positive and negative training instances. Once this happens, it is impossible to find consistent concept descriptions which are true for all positive training instances and false for all negative training instances.

Though the best way to solve this problem is to make the data complete after some considerations of cost and effectiveness of doing it, this may not often be possible. We may simply ignore the missing data if they are not important. The other alternative is filling the missing part of the data on the basis of constraints procured from common sense or domain-specific knowledge or heuristics (may be encoded into "half order theory" as in Meta-DENDRAL [Buchanan 78a]) with respect to the existing data. "Default rules" may

serve this purpose as well; they will be triggered if no other rules are available. The conclusions made by default rules will be accepted unless inconsistency is detected. Filling incomplete or missing data can thus be done by considering constraints and/or employing default knowledge. Filling missing data is also done in statistics (refer to [Madow, Nisselson, and Olkin 83]). Some important message may be lost by filling the data; it may even cause misleading results [Dempster 83]. It demands caution indeed. One paradox here is that suppose the deduction theory is strong enough to fill any missing datum, nothing can be learned. However, note that "deduction and induction are not antagonistic, but complementary" [Croxtton, Cowden, and Klein 67]. Deduction based on some old knowledge can help to discover new knowledge by induction.

Practically, we might not want to fill anything unless there is no other alternative or it is quite straightforward. As an example, if we want to investigate the association between a disease and sex, then it is fully justified to take a pregnant person as "woman" even though this fact is not included in the existing data.

4.2.2.2. Sampling Insufficiency

This indicates the following conditions:

1. The number of instances is too small.
2. The instances are atypical.

The objective of learning from examples is to find concept descriptions that are consistent not only with the training instances but also with all instances in the domain. Since the learning is strongly biased by the given set of training instances, a good result demands an adequate sampling.

A small set of training instances may not reflect the real distribution; this is particularly deleterious in domains where decisions rely on statistical knowledge. In multiple (disjunctive) concept learning, only a limited number of concept descriptions or rules can be learned from a small number of training instances. In single concept learning, the version space [Mitchell 78] won't converge upon the desired description if no sufficient instances are available.

To overcome this problem, more instances are required, and the atypical instances will be diluted. Careful selection of instances can make the result more precise (e.g., near-misses proposed by [Winston 70]) and make the desired concept description more rapidly converged upon [Mitchell 78]. However, on the other hand, if uncertainty is involved or there should be multiple disjunctive concepts, the selection should be random to average out those invisible factors underlying the instances or to avoid losing generality. Incomplete sampling is also an important topic in sample surveys, one solution is seen in [Sirken 83].

But what if more instances are not available? One philosophy of science, as stated by [Brillouin 62], is: "if we cannot observe them, let us admit that they have no reality"; he added "we must candidly admit that we do not know". That is, we should avoid over-interpreting what we observe. Two strategies have been adopted by researchers to cope with this issue. First, all consistent descriptions in the version space should be preserved until forced to be eliminated by new instances; this strategy is called "least commitment" and is adopted in candidate-elimination algorithm [Mitchell 78]. Secondly, if only positive instances are available, the generalization should be maximally specific (refer to [Dietterich 83]).

4.2.2.3. Unreliability and Inconsistency of Data

Since unreliability implies errors, learning based on unreliable data will be erroneous at least to some extent. It is hard to tackle this problem. If available, it is always desirable to replace the unreliable data with reliable ones. Otherwise, the data may be revised, based on domain-specific knowledge and common sense (encoded into half order theory). Not only inconsistency should be detected, but also it should be resolved. The complexity may demand an expert program; RULECRITIC [Haggerty 84] may be such an example.

Practically, if we do not want to distort the data, then what can be reasonably done is to determine how reliable the data are by checking the consistency among data and asking the source of the data and to assign a reliability index to the result. Again, this may require an expert program; REFEREE [Haggerty 84] may be such an example.

4.3. Imperfect Learning Systems

4.3.1. Insufficiency of the Descriptive Language

Inconsistency may be due to the incapability of the language fed to the learner. The cause of this problem is often ignorance rather than bias. That is to say even human experts don't know what features or descriptors are missing in the provided language, rather than they improperly choose features or descriptors because of their bias. Thus, it becomes a hard issue; though more useful features or descriptors may be discovered and exploited, as knowledge evolves.

To classify instances, we might start with one feature and add more features until proper classification is achieved. Suppose there are "n" features, the space of classification is

n -dimensional, and the decision boundary will be a hypersurface of less than n -dimensions. To seek a good feature is worthwhile because it may replace several features and reduce the dimensions of the classification space; and thereby the complexity of the problem can be greatly reduced. (Note that the complexity usually increases with the number of features nonlinearly or, in the worst case, exponentially. There are more discussions on this issue in Chapter 3.) Consider an example in medicine, CT (computerized tomography) may provide more information than several old examinations in the diagnosis of brain tumors.

This problem, however, can be solved at least partially by the following strategies:

1. Extend the initial language.
2. Change the representation (the style of the descriptive language) or add another representation (refer to Section 4.3.4).

Extending the initial language by either syntactically combining different features or analytically defining new features can relieve the bias imposed by the fixed language, as proposed by [Utgoff 82]. In Section 2.3, we develop some techniques which can define new useful intermediate symbols and thus augment the initial language. However, this issue is still largely unexplored.

It is always desirable that the system can interact with human experts and negotiate for new features or descriptors.

4.3.2. Insufficiency of Rules of Generalization or Specialization

In learning from examples, we generalize to cover positive instances and specialize to exclude negative instances. For example, rules of generalization include as follows: dropping conditions, variable replacement, climbing generalization tree, etc. Learning is a search in the space of all possible hypotheses. The search tree is expanded by the learning operators (rules of generalization or specialization) available. Inadequate operators will narrow the search space, and improper operators may mislead the expansion of the search tree. This issue can be illustrated by the following example. If we want to make induction from three positive instances: (2 5), (4 7), (8 11), unless we have the operator "subtraction" or "difference", it is hard to observe the regularity among these three instances.

Modifying or even creating new operators requires higher level knowledge and heuristics. In the EURISKO program [Lenat 83], a heuristic rule can be changed by meta-heuristic rules. Before we develop this far, the better way to cope with this problem is asking human experts for new operators.

4.3.3. Procedural Bias

In learning, heuristics are often used to avoid exhaustive search. In fact, if the hypothesis space (rule space) is huge, heuristic search (e.g., in Meta-DENDRAL [Buchanan 78a]) is the only way to make the learning feasible. Since the heuristics are not 100% correct, some important rules might be missed because of the incompleteness of the search. If the results of learning are not satisfactory, the heuristics or knowledge guiding the learning should be modified or altered.

Figure 4.5 shows the error caused by the generalization from disjunctive concepts. Data-driven learning algorithms are more susceptible to this type of error. However, this error can be avoided by testing the generalization against more negative instances; i.e., if many negative instances are included by the generalization, then disjunction is possible (other possibilities include "false positive instances" and "exceptional positive instances").

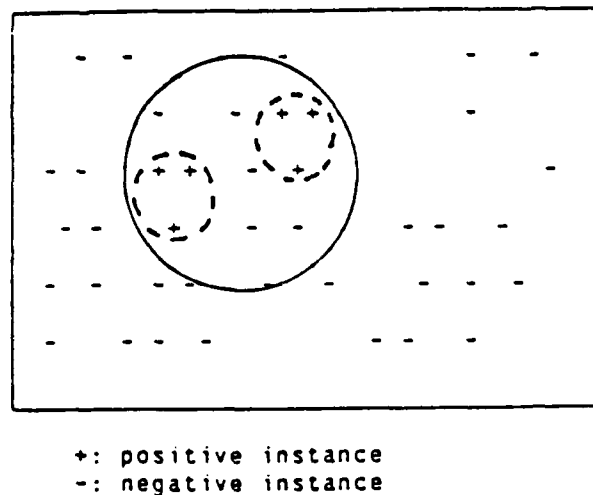


Figure 4.5 Excessive generalization caused by generalizing from disjunctive concepts.

Sometimes, because of an inappropriate bias, the learner may fail to generalize from non-disjunctive concepts. This is illustrated in figure 4.6. This type of error is suspected if the number of the learned concept descriptions or rules is more than expected or if strong similarity is observed among them.

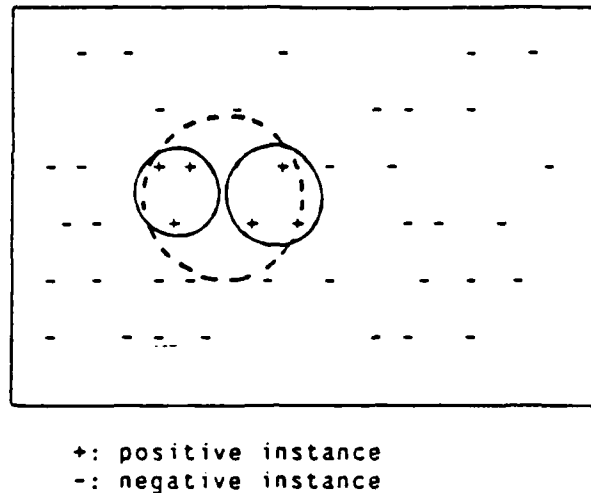


Figure 4.6 Failure to generalize from non-disjunctive concepts.

4.3.4. Representational Bias

Representation is a key issue in artificial intelligence; a good representation will facilitate the discovery of new and useful concepts or rules. The criteria for choosing representation include the following: representational adequacy and efficiency. "Representational adequacy" says: the representation should be able to characterize the instances or concepts and be flexible enough to be adapted to learning operators. For example, semantic net is superior in representing intricate relationship; and if we want to represent a statement "object A is on the top of the object B", then "(on-top A B)" is more flexible than "(on-top-of-A B)". "Representational efficiency" says: the representation should achieve operational advantages which include the computational and spatial

economy. However, there is sometimes a trade-off; an adequate representation may incur more computational efforts, and vice versa.

Inappropriate representation may render the training instances inadequately described and thus lessen or distort the learning results. Changing representation or adding another representation is the possible solution. This is because of the reason that insufficiency in one representation does not necessarily mean insufficiency in others. For a given domain, we often choose a representation scheme which can more adequately capture domain features. For example, in chemistry, the chemical bond language (analogical representation) is more natural and incorporates more semantics. However, to add another representation may sometimes be useful; for example, in chemistry, we may place some frames to describe the properties of atoms or molecules. Though, in current AI learning, the representation is always pre-determined by human designers, it is preferable that the system has the ability to modify or change the representation, as suggested by [Lenat 83].

4.4. Error Measurement

There are two criteria for measuring the quality of learning results: calibration and prediction error.

4.4.1. Calibration Error

Human experts may select proper instances for a given concept or may conclude a concept for a given set of instances; if these instances are fed to the system and the output from the system is compared with the known output (i.e., the given concept or the expert-concluded concept); the difference thus measured is called *calibration error*. The objective of this measurement is to tune the system to an ideal state (or best plausible state) in terms of the capability of generating some well-known facts before learning unknown facts.

The differences between the observed and the known output are determined by the following factors:

1. The ratio of the intersection of the known and the observed output to the observed output.
2. Whether there exist contradictions.

Ideally, the observed output should be exactly the same as the known output. The first factor is the measurement of the percentage of good quality results. If the percentage is low, the performance is not efficient (nor accurate). Contradictions to the known output will jeopardize the results: one bad rule may sometimes be worse than one hundred good rule in a risky domain. When the observed output is not syntactically the same but bears the same meaning (construed by experts) as the known output, they should be regarded equivalent.

The following aspects should be considered as well:

1. Since the objective of calibration is with respect to the system, the training instances should be as perfect as possible.

2. The calibration should be carried out in the same domain because the system usually incorporates domain-specific knowledge and heuristics. For instance, if the system is designed to learn medical rules, it is inappropriate to use chemical molecules for testing.

4.4.2. Prediction Error

The learned concepts or rules are used to predict instances which are already classified correctly. The predictions are then verified. The number of incorrect predictions is defined as *prediction error*. There are two types of prediction errors: "false positive predictions" and "false negative predictions". False positive predictions mean predicting negative instances as positive instances; false negative predictions mean predicting positive instances as negative instances. However, in an expert system with more than one diagnostic category, false positive predictions mean incorrect conclusions, and false negative predictions are defined as "cases which are not predicted to be any pre-defined category" in our scheme. If the predictions are very accurate, then both types of errors should be zero. Now, the magnitude of prediction error is defined as follows:

$$\begin{aligned} |\epsilon| &= \text{mispredictions} \\ &= \text{FP} + \text{FN} \end{aligned}$$

where $|\epsilon|$: magnitude of error
FP: false positive predictions
FN: false negative predictions

With respect to the prediction error, there are somewhat different interpretations between single concept learning and multiple concepts learning. If the output is a single concept description (rule), then the false positive prediction indicates the learned rule is overly general, and the false negative prediction indicates the learned rule is not sufficiently

general. If the output is multiple rules, then the false positive predictions indicate some of the rules are overly general, and the false negative predictions indicate some of the rules are not sufficiently general or some rules are missing. If the system has been calibrated to zero calibration error, then the prediction error will reflect mainly the noise associated with the input.

In EMYCIN-based systems [Van Melle 80], since the rules may either positively or negatively interact with one another and certainty factors can be combined, the conclusion is made by a set of rules rather than an individual rule. Since the assumption of independence underlying the combination of certainty factors is not always true, the conclusion made by the set of rules may still be incorrect even if all individual rules seem correct. In such systems, it seems indicated to make a distinction between *global* and *local* (or individual) errors; the former denotes the error with respect to the set of rules, and the latter denotes the error with respect to individual rules. The global error is defined as before as follows: false positive predictions denote incorrect predictions; false negative predictions denote cases which are not predicted to be any pre-defined category. The local error for an individual rule is defined as follows: false positive predictions denote cases are predicted by the rule to be the class indicated by the RHS of the rule whereas they are not in this class; false negative predictions denote cases in the class indicated by the RHS of the rule are not predicted by the rule to be in this class. It is quite straightforward to compute false positive and negative predictions for global error, based on the predictions made by the set of rules. On the contrary, it is somewhat obscure to compute the false predictions based on an individual rule, because of the following facts: first, for disjunctive concepts, a rule does not necessarily cover all positive instances; secondly, if

uncertainty is involved, a rule does not necessarily exclude all negative instances. However, if we take an ideal assumption that a rule should cover all positive instances (the instances indicated by the RHS) and exclude all negative instances, then the prediction error computed under this assumption, though maybe overly idealized, can reflect the performance or quality of a rule; the lower the error, the better the quality, and vice versa. We may also define the minimal generality as the minimal coverage of the positive instances and define minimal specificity as the maximal coverage of the negative instances. Then if a rule breaks either of the constraints, its error is defined to be "infinity".

So far as a learning system is concerned, we also distinguish between *intrinsic* and *extrinsic* errors; the former denotes the error with respect to the training instances in the input, and the latter denotes the error with respect to the instances other than the input. The concept descriptions or rules learned from the input instances will tend to be more consistent with them than with other instances. Thus, in general, the intrinsic error is smaller than the extrinsic error. As described in the introductory remarks in this chapter, the method developed here is intended to minimize the intrinsic error; and we leave the task of minimizing extrinsic error to other chapters (see focusing mode of learning in Section 2.2.4, and automated debugging in Chapter 5).

4.5. Error Handling

Figure 4.7 shows an efficient and noise resistant learning network. The role of CONDENSER is discussed in chapter 3. Here, we focus on the noise filters. Depending on the stage of intervention, we name the following: pre-filter, mid-filter, and post-filter. As will be described in more detail, pre-filter and mid-filter deal with detecting and

removing the error-causing factors directly while post-filter deals with optimizing the result by minimizing its error while disregarding the error-causing factors during optimization.

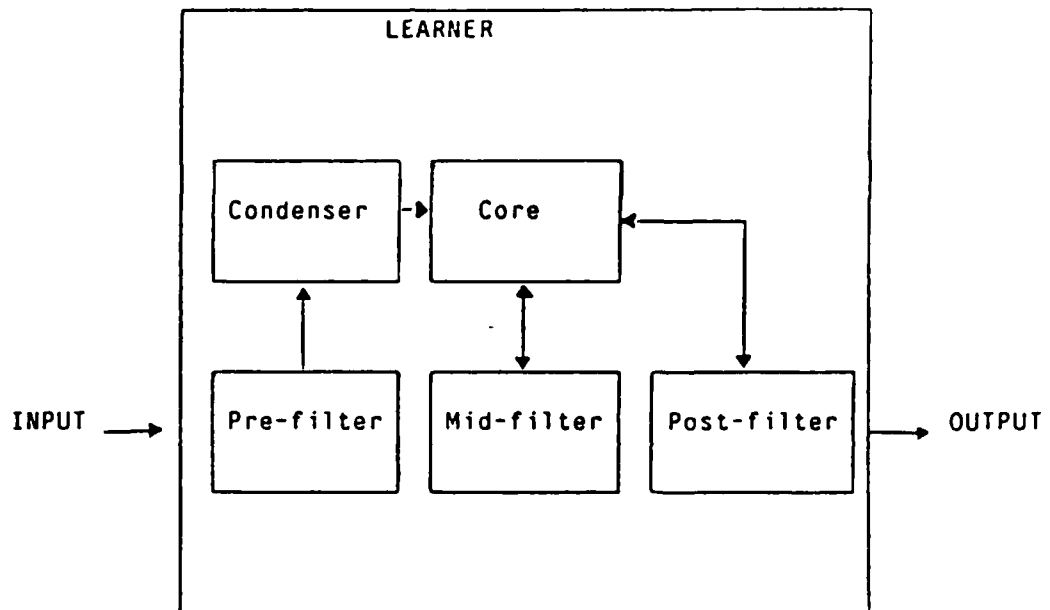


Figure 4.7 Diagram of an efficient and noise-resistant learning system.

4.5.1. Pre-Filter

The role of the pre-filter is to choose a proper language and learning algorithm, and to remove imperfect instances. Imperfect instances include: falsely classified instances, ambiguous instances, instances with incomplete or unreliable data. The choice of a proper language or algorithm is often domain-dependent; for example, chemical bond language is chosen in Meta-DENDRAL [Buchanan 78a], and the learning method described in Chapter 2 is designed for multiple concept learning. It is a difficult task to detect false positive or false negative instances. One way is to measure similarity or dissimilarity. If a positive instance is found to be dissimilar (refer to Section 2.3.2.1 for measuring dissimilarity) to all other positive instances, it is labelled as a potentially false positive instance (thought it may be an exceptional instance) (see also figure 4.3). If a negative instance is found to be very close to positive instances and dissimilar to other negative instances, it is labelled as a potentially false-negative instance (see also figure 4.4). The potentially falsely classified instances are considered in the last resort. Ambiguous instances are simply removed. The current implementation of pre-filter is only limited to the above descriptions.

The INTSUM program in Meta-DENDRAL [Buchanan 78a], which is able to remove inconsistent or erroneous data by the aid of a half-order theory, is one example of the pre-filter.

4.5.2. Mid-Filter

This filter is designed for data-driven learning methods, with which the current maintained hypotheses will be modified to accommodate new instances in the following manner. If the new instance is a positive instance, minimally generalize the current hypothesis to cover it; if the new instance is a negative instance, specialize the hypothesis to exclude it or abandon some hypotheses which become inconsistent, depending on the search strategy.

The mid-filter monitors the learning process. It detects and removes the noise by the following rules:

"If a positive instance causes excessive generalization,
then it may be a false positive instance."
(see figure 4.3)

"If a negative instance causes excessive specialization,
then it may be a false negative instance."
(see figure 4.4)

The suspected instances are stored in the list of the last resort. Unless the learning result is not satisfactory, they will not be reconsidered. Some thresholds are required to determine excessive generalization or specialization. If they are not properly chosen, more than one iteration may be required to achieve a good result. Moreover, the initiation of the current hypotheses is critical; if this is based on a false instance, the result will diverge rather than converge, and the learning has to be reinstituted. This filter finds no value in model-driven learning methods since instances will not be considered individually with them.

4.5.3. Post-Filter: Optimizer

4.5.3.1. Minimal Error Principle

A "minimal error principle" is used in the post-filter. It is that the desired information content can be more accurately estimated by minimizing the error, a principle widely applied in signal processing (refer to [Papoulis 65] and [Balakrishnan 84]).²⁹ In inductive concept learning, we have described two techniques for measuring the error associated with the output of a learning system (see Section 4.4). Here, we particularly focus on minimizing the prediction error since this is the ultimate goal.

False positive predictions (also called over-predictions) and false negative predictions (also called under-predictions) may carry different levels of risk. This is particularly true in medicine. For example, over-prediction (over-decision) of a patient's requirement for chemotherapy is dangerous while under-prediction of acute appendicitis will delay the operation and incur mortality. Hence, it is justified to assign a weighting factor for each type of error. And *weighted prediction error* is defined as follows.

$$wpe = w_p FP + w_n FN$$

where, wpe: weighted prediction error
 w_p : weighting factor for FP
 w_n : weighting factor for FN
 FP: false positive predictions
 FN: false negative predictions

The weighting factors, standing for the domain attitude toward two types of errors, will vary with different concepts to be learned. Practically, the weighting factors can be

²⁹For filters, such as Wiener's filter, the mean square error is minimized.

acquired by asking the expert the costs incurred by these two types of false predictions. The default is $w_p = w_n = 1$. Since there may be a trade-off between FP and FN, as implied in figure 4.2, the only alternative is to minimize the weighted sum if they cannot be minimized independently. Sometimes, people may define a function to evaluate the performance, but it is noted that, with respect to the prediction power, "minimizing the error" is the dual of "maximizing the performance". In EMYCIN-based systems, as described before, we should make a distinction between global and local prediction errors. If the local prediction errors for all individual rules are zero, then the global error is also zero (i.e., if every rule can cover all positive instances without including any negative instance, then the global conclusion based on the combination of all individual conclusions is also impeccable), but not vice versa (i.e., zero global error does not necessarily mean every individual rule can cover all positive instances and exclude all negative instances). As discussed previously, the local prediction error is not necessarily the "error" (e.g., if disjunction occurs); however it can measure the performance of an individual rule: that is, we tend to believe a rule will be more powerful if it can cover more positive instances and exclude more negative instances (in other words, if it is associated with a lower prediction error, as defined). Our main goal is to achieve minimal global errors rather than minimal local errors. Therefore, global tuning comes before local tuning. The system parameters are first adjusted, then the output is subjected to local optimization. The systems will be tuned until the global error is minimized. The procedures will be described next.

4.5.3.2. Procedures

As described in Section 2.2, there are several constraints which define either the minimal generality or minimal specificity; all these constraints are called "system parameters" here. The parameters defining the minimal generality are somehow more related to false negative predictions, so are those defining the minimal specificity to false positive predictions. Because of the minimal generality, rules cannot be too specialized and are expected to predict more positive instances; thus the possibility of false negative prediction can be reduced. Similarly, setting a threshold for minimal specificity can prevent a rule becoming overly generalized and thus the possibility of false positive prediction can be reduced. Note that we implicitly assume the threshold for minimal generality is more specific than the threshold for minimal specificity (assume partial ordering in the version space); otherwise the above argument will not hold. Based on this analysis, tuning these thresholds can have impact on the global prediction error. Though the best way to determine these parameters is based on domain-specific knowledge or heuristics, sometimes they are not available. If there are four parameters, the search space is huge. Our strategy to cope with this issue is as follows:

- step 1. Initialize the parameters with ideal numbers; i.e., we may first assume the minimal generality is 100% (a rule should cover all positive instances), and the minimal specificity is "0" (i.e., a rule should not include any negative instance).
- step 2. If available, calibrate the system with some typical rules (well-established knowledge in textbooks); i.e., adjust the system parameters such that these typical rules will come out. In comparison with the typical known rules, if the output is overly general, reset the threshold for minimal specificity; if the output is overly specific, reset the threshold for minimal generality.

- step 3. Minor adjustments of system parameters based on the adjustments in step 2 (i.e., try several finite states around the state determined in step 2) are done until the optimum (i.e., minimal global weighted prediction error) is achieved.

In our experiments, we found that calibrations in step 2 are crucial for an efficient optimization. As seen in figure 4.7, the learner core receives feedback from the post-filter (optimizer).

The local optimization for individual rules proceeds as follows:

- step 1. Set the variable `OUTPUT` := input rule.
- step 2. Make one step transformation of `OUTPUT` by either one step generalization or one step specialization of `OUTPUT`. But the transformation should not be repeated. If the weighted prediction error of `OUTPUT` is smaller than that of all transformations derived from it, then do nothing, go to exit, and the output is the value of `OUTPUT`; otherwise, reset the variable `OUTPUT` := transformation with minimal weighted prediction error. The number of possible transformations is controlled by heuristics. For example, one step specialization is done by adding one feature value that appears in positive instances with relatively high frequency. One step generalization is done by removal of one conjunct from the LHS of `OUTPUT` or replacing one feature value in the LHS with minimally more general value.
- step 3. Go to step 2.

Therefore, if the variable `OUTPUT` reaches a local minimum with respect to the weighted prediction error, i.e., all possible minor changes (transformations) could not be better, then the procedure is terminated, and the final value of the variable `OUTPUT` is the output. Thus, the local optimization is done by performing a hill-climbing search with respect to the local prediction error. In EMYCIN-based systems, since there is uncertainty involved, one additional constraint during optimization is that the certainty factor or degree of

certainty should be reasonably maintained (in JAUNDICE, the difference should be less than ".15"); this comes from the argument that different ranges of certainty factors bear different meaning. In the JAUNDICE experiment, we use the learning method developed in Section 2.2, which searches for maximally specific rules first and optimizes the results by generalization operators.

4.6. One Example

The following illustration shows how the post-filter optimizes a rule. Recall that there are four main steps in the learning method described in Section 2.2.1; the post-filter corresponds to step 3.

Example. Assume the weighting factors in the "wpe" (weighted prediction error) is as follows:

$$w_p = 2, w_n = 1$$

There are 20 cases diagnosed as "cancer" in the case library.

One rule, before optimization, is as follows:

"If 1. Serum bilirubin is elevated.
 2. Body weight loss is greater than 15 lb.
 3. Disease course is progressive.
 4. Ascites is present.
 5. Liver is enlarged.
 6. Liver is hard.
 Then probably (.7) cancer."

Because it covers two negative instances (non-cancer) and covers only four positive instances (cancer),

$$wpe = 2 \times 2 + 1 \times (20 - 4) = 20$$

After optimization:

"If 1. Serum bilirubin is elevated.
 2. Body weight loss is greater than 15 lb.
 3. Liver is enlarged.
 4. Liver is hard.
 Then likely (.65) cancer."

Now, it still covers two negative instances but covers nine positive instances, so,

$$wpe = 2 \times 2 + 1 \times (20 - 9) = 15$$

4.7. Comparison and Discussion

In model-driven learning systems, because there exists a certain criterion to test the model-generated hypotheses, the result of learning will tend to be more noise-resistant [Dietterich 83]. Basically, the criterion is to maximize the covered positive instances and minimize the included negative instances. For example, in Meta-DENDRAL [Buchanan 78a], the criterion to rank rules is defined in an *ad hoc* fashion as " $I \times (P + U - 2N)$ " (where I is average intensity of positively predicted peaks, P is the number of correctly predicted peaks, U is the number of uniquely predicted peaks, and N is the number of incorrectly predicted peaks), and the RULEMOD program is to optimize with respect to this score by specialization and generalization on the basis of the "seeds" generated by the RULEGEN program. However, the herein developed method bears the following distinct features:

1. We generalize the criterion by defining a "prediction error", and the objective is to minimize the weighted prediction error.
2. Both global and local optimizations are considered because the method is designed in EMYCIN-like frameworks where strength of a conclusion from different rules can be combined. In contrast, other learning systems only consider local optimization.

In data-driven learning systems, because individual cases are considered equally and there exist no global criteria to constrain the process of generalization or specialization, the

systems are more susceptible to noise associated with the data. In single concept learning, one solution, proposed by [Mitchell 78], is to maintain multiple version spaces: if the current most desirable version space is collapsed, the algorithm backtracks to the next less desirable version space. The weakness of this solution is the potentially huge storage space for the multiple boundary sets (assuming there are no heuristics to prune the boundary sets), the storage for which becomes nonlinear with the number of instances under inconsistency. In contrast, the method developed here sets up a global criterion to test the hypotheses generated by data (e.g., generalization from two positive instances, or specialization to reject negative instances) and thereby to detect the possible incorrect or incomplete instances, which are deleted or considered last. The result is further optimized. As the degree of inconsistency grows, this approach, as an alternative to maintaining multiple version spaces, becomes more economical because the storage space for instances will not expand as that for boundary sets. Notice that an unified optimization technique is actually developed here to handle the noise in learning, whether the learning system is model-driven or data-driven. However, this optimization technique calls for an initial set of training instances: recall as well that our learning model starts from a set of training instances rather than a single instance.

Issues, such as small sample size, incomplete data, and bias, also occur in statistics. "Confidence interval" is used to measure the quality of a statistical result: the larger the sample size, the narrower the interval, and thus the better the estimate (refer to [Croxtan, Cowden, and Klein 67]). Special modifications to statistical techniques are required to deal with small sample size, e.g., Yate's correction [Croxtan, Cowden, and Klein 67]. In sample surveys, the methods used to tackle incomplete data include the following:

network sampling (i.e., the information about one node can be obtained from its neighbors through the network if the information cannot be obtained directly) [Sirken 83] and imputation (i.e., "replacing the missing data by estimates of the missing items") [Madow, Nisselson, and Olkin 83]. Bias in a statistical test can be minimized by "randomization" and "blindness" of the test. Although the knowledge of data handling in statistics can be transferred to inductive concept learning, it is still hard to apply the ideas, such as "confidence interval", to learning; for example, it is hard to tell how far a learned rule from the truth is by simply looking at the size and variance of the sample. The fact that statistics is "the collection, presentation, analysis, and interpretation of *numerical* data", as defined by [Croxtton, Cowden, and Klein 67], makes statistical techniques fall short in AI, where symbolic reasoning dominates.

Errors may occur in all kinds of empirical science. A scientific discovery relies on careful and patient observations, as suggested by [Hahn 30]. As the new technology emerges, a scientific theory, which is once true, may be subjected to modifications to accommodate exceptions or even be overthrown. That is why we think incremental learning is essential. In physics, as said by [Brillouin 62], "with Heisenberg uncertainty principle, the fundamental role of experimental errors becomes a basic feature of physics"; he thought, despite the classical ideal view that the error can be made as small as possible and ultimately negligible by careful instrumentation, "errors are an essential part of the world's picture and must be included in the theory". In electrical engineering, filters, such as Wiener's and Kalman's (refer to [Papoulis 65] and [Balakrishnan 84]), are designed to remove noise (e.g., white noise). Mathematically, there are many analytical or numerical techniques for optimization, some of which are widely applied in economics, engineering

science, etc. However, the optimization in AI is more or less a search and relies on symbolic techniques, such as generalization or specialization. Regardless of methodological differences, the common underlying principle of error handling in all kinds of science is to minimize the error or cost.

4.8. Summary

Considered in this chapter is learning in an imperfect environment. Human efforts will be involved to make the environment as perfect as possible; however, because of human bias or some factors beyond human consideration, errors will remain. Seeking a state which is maximally consistent with the training instances is the strategy used in this work. We define "weighted prediction error", which is a general criterion in inductive concept learning. By careful applying the available operators and adjusting the system parameters to minimize the weighted prediction error, the desirable result can be achieved; and the perturbation in the data or the system will largely be ignored during optimization under the assumption that the perturbation is a relatively small fraction. Better accuracy is thus purchased at the cost of optimization.

Chapter 5

Automated Knowledge Base Updating

5.1. Introduction

Our realistic goal is to build a complete model of inductive concept learning in expert systems. In the previous chapters, we have solved the first stage problem: constructing a knowledge base (KB) from a set of training instances. The subsequent use of this KB in concluding new cases may face another issue when an incorrect conclusion occurs: an issue related to tracking down the faults and correcting them, which is formulated as follows:

Given: 1. A knowledge base (KB).
2. An incorrect conclusion based on the KB.

Find: Corrections to the KB such that the conclusion can be rectified.

Recall that, in Section 4.4.2, we define "intrinsic error" (with respect to the training set used to construct the initial KB) and "extrinsic error" (with respect to instances other than the training set), and we have developed a solution to minimizing the intrinsic error; this chapter, as a continuation, is devoted to improving the KB by minimizing extrinsic errors. The task defined above resembles the focusing mode of learning described in Section 2.2.4. In fact, we apply the learning technique to debugging the knowledge base; and the descriptions in this chapter primarily deal with how to integrate those learned rules based on the incorrectly concluded case into the old KB, which was constructed previously.

This problem, knowledge base debugging,³⁰ is explored in other AI work, such as TEIRESIAS [Davis 79], EMYCIN editor [Van Melle 80], SEEK [Politakis 82], and [Waterman 68]. In a generalized model proposed by [Buchanan 78b], the "critic" deals with the so-called "credit and blame assignments"³¹ and recommending changes to the performance element via a source which can provide new knowledge.

The motivations of this chapter are the following considerations:

- A simplified view of knowledge base debugging is as follows: generalize overly specialized rules, specialize overly generalized rules, add missing rules, delete erroneous rules, and resolve conflicting rules. But this view is somewhat oversimplified in EMYCIN-like systems where evidence can be combined and uncertainty is involved; these facts render the debugging inexact. This work is intended to embody the expert's thought which is relied on in programs, such as TEIRESIAS, to debug the KB. The automation demands some considerations which are raised mainly because of lacking empirical knowledge which experts use to debug the KB. The credibility of automated debugging is also a related concern.
- As mentioned, our ultimate goal is to establish a complete model of inductive concept learning, a model which starts from a set of training instances and incrementally updates the KB. Again, we emphasize the notion of "optimization" in inexact domains. We try to minimize the errors in a period from "t=0" when the KB is initially constructed to now.

³⁰ But, within the framework developed in this thesis, we think *updating* is a better term than *debugging* since our learning model is initially based on a set of training instances, which may provide only partial statistics, and as the database accrues, statistics shift to better closeness to the real distribution of the population, and the old knowledge, which may be right in one time and wrong in another, is updated or obsoleted. The difference between these two terms, if any, perhaps is a more constructive connotation associated with the former (updating) than the latter (debugging). However, we still preserve the term "debugging" because it is often used in other works, and the debugging program described in this chapter can be detached from our learning model and applied to another KB which may be constructed by human experts.

³¹ The credit assignment problem is first raised by Minsky [Minsky 63].

The assumption made in this chapter is that there exists a database, based on whose initial form, a KB is constructed as its starting point.³² This database serves as an important reference (and actually imposes a considerable constraint) for updating the KB because we don't want to purchase the accuracy with respect to a single new case at the cost of the accuracy over the old reference cases. Furthermore, to achieve a rapid convergence of the KB, the initial set of training instances should be adequate for providing a good starting basis (i.e., representative of the whole population). Otherwise, the learning may proceed back and forth.

In our scheme, we take advantage of a strategy we call "retrospective inspection after learning". This strategy is also employed in other work, such as [Waterman 68]; the difference will be analyzed in Section 5.5. The "knowledge base updating" in this work comprises the following steps: learning, proposing experiments, and verifications; the last two steps are required because the source that indicates the faults in the system conclusion is not always reliable, and even if it is reliable, the KB will remain in its old version if the modifications proposed to accommodate the new case are not favored by the old reference cases.

We first describe the possible faults in the KB and their ordinary rectifying operators. Then we explore the issue of updating the KB when a faulty conclusion emerges, emphasizing learning in EMYCIN-like systems.

³²Though it is not necessary that the KB is built automatically, we assume so in this thesis. The database can be controlled to a reasonable size by constantly removing the overflow without disturbing its statistics, but the original training set used to build the KB and the cases which are incorrectly concluded are maintained.

5.2. Faults in the Knowledge Base

In expert systems, the errors of performance can be traced back to errors in the KB or sometimes to errors of the inference engine. But, because of the nature of inconsistency in the domain (or say uncertainty) and the incompleteness of data, some degree of errors can be allowed. A standard must exist for evaluating the performance of the system. The debugger will be triggered to debug the KB only if the performance is judged as "bad". The basic assumption is that the chosen standard is right; otherwise it will be nonsense to debug the KB. For example, in TEIRESIAS [Davis 79], the standard comes from the expert. In this chapter, we use *weighted prediction error* (defined in Section 4.5.3.1) as an additional performance standard to guide debugging the KB.

The "faults" described in this section designate either *true error* or *improperness* which impairs the system performance with respect to a certain standard. "True error" means the associated semantics conflicts with real observations; for example, the statement "all mammals are plants". "Improperness" means the associated semantics is right but not optimal; for example, the statement "men at the age of 50 are mammals". In an expert system, improperness of rules, such as overly generalized or specialized rules, may cause false predictions. For example, the rule "men at the age of 50 are mammals" will make men at the age of 20 unconcluded (false negative prediction) if there is only one such rule dealing with "men" in the KB of an expert system designed to conclude whether an animal is a mammal. And a misconception made by the system may reflect "true error" or "improperness" of individual rules. In EMYCIN-based systems, the assumption of independence for combining certainty factors doesn't always hold; therefore even if all individual rules seem right, the global conclusion may still be wrong. Although it is easy

to define "true error", it is hard to delineate "improperness" particularly if uncertainty is involved.

5.2.1. In Domains without Uncertainty

5.2.1.1. Overly Generalized Rules

An overly generalized rule is a rule which causes false positive predictions because the conditions (or descriptions) in the LHS (left hand side) of the rule are too general. For example,

```
instance1 with attributes A1, A2, classified as class A
instance2 with attributes A1, A3, classified as class B
```

Then, the rule "A1 -> class A" is overly generalized, because instance2 is falsely classified as class A by this rule.

5.2.1.2. Overly Specialized Rules

An overly specialized rule is a rule which rarely succeeds because the LHS of the rule is too specific (overly constrained).

```
instance1 with attributes A1, A2, classified as class A
instance3 with attributes A1, A4, classified as class A
.....
.....
```

Then, the rule "A1 & A2 & A4 -> class A" will be too specific for both instance1 and instance3. If only few instances in class A can satisfy this rule, it is overly specialized.

In a domain with disjunctive concepts, instances in a given class may be covered by different rules, and we can't expect a rule can cover all instances. But, if a rule can cover no or few instances only, it is regarded overly specialized. False negative predictions may be ascribed to the overly specialized rules (or missing rules) in the knowledge base.

5.2.1.3. Erroneous Rules

An erroneous rule is a rule which contradicts the truth (or the currently recognized knowledge). Even if the KB is built by a group of experts, there is no guarantee whatsoever that all the rules in the KB will be 100% consistent and accurate. Factors causing erroneous rules include the incorrect knowledge of the KB builders and the knowledge shift (today's knowledge may not be tomorrow's knowledge). Subsequent tests after building the KB are important to detect errors.

The following two rules contradict each other (if A and B are mutually exclusive):

```
"A1 -> class A"
and,
"A1 -> class B"
```

Thus, if one rule represents the truth, the other will be erroneous. However, if uncertainty is involved (see also Section 5.2.2.3), two rules with the same LHS but with mutually exclusive RHS may be compatible unless at least one of them is assigned a degree of certainty "1". For example, the following two rules are compatible:

```
.6
"A1 -> class A"
and, .4
"A1 -> class B"
```

5.2.1.4. Missing Rules

For a given instance, if no rules can correctly classify it, then it is possible that some rule is missing (or some rule is overly specialized, or the data are incomplete).

5.2.1.5. Subsumption

Subsumption occurs if two rules have the same conclusion but the premise of one rule subsumes that of another. If both are true, keep the more general one. In the following example, the premise of rule R1 subsumes that of rule R2:

```
R1: A1 & A2 -> class A
R2: A1 -> class A
```

5.2.1.6. Redundancy

Redundancy occurs if two rules share a common premise and conclusion. Only one rule should be kept.

5.2.2. In Domains with Uncertainty

5.2.2.1. Overly Generalized Rules

By an overly generalized rule, we mean a rule whose degree of certainty is below some threshold or which covers more than a threshold number of negative instances. A rule with low degree of certainty implies its LHS is not very specific for concluding its RHS. In JAUNDICE, we choose ".4" as the threshold. Thus, a rule with degree of certainty below .4 is treated as an overly generalized rule. Also we define that a rule, whatever the degree of certainty is, should not cover more than 10% of negative instances (in JAUNDICE); otherwise it is overly generalized. Though it seems reasonable to obtain a piece of certain information by accumulating several pieces of uncertain information, we are still reluctant

to accept a piece of very uncertain information. Moreover, the accumulation is under the assumption of independency; since this assumption is not always proper, the accumulation may lead to an erroneous result. Therefore, it is justified to remove the rules with low degree of certainty.

5.2.2.2. Overly Specialized Rules

By an overly specialized rule, we mean a rule rarely succeeds because there are too many conditions (or too many constraints) in the LHS of the rule. In JAUNDICE, if the number of conditions in the LHS of a rule exceeds 6, the rule will generally be considered as an overly specialized rule. An overly specialized rule may be right individually, but it is globally improper (from the viewpoint of the global system performance) since it may cause false negative predictions.

5.2.2.3. Erroneous Rules (or Erroneous Degree of Certainty)

It makes little difference whether we say a rule is erroneous or the degree of certainty assigned to it is erroneous. The rationale behind this is briefly analyzed as follows. Consider a rule:

$$R1: P \overset{d}{\rightarrow} C$$

If "R1" is to confirm "C" (i.e., "P" is positive evidence for "C"), then the degree of certainty "d" should be a positive number; if "d" is not a positive number, then "R1" is wrong. On the contrary, if "R1" is to disconfirm "C" (i.e., "P" is negative evidence for "C"), then "d" should be a negative number; if "d" is not a negative number, then "R1" is wrong. If "P" has nothing to do with "C", then "d" should be zero; if "d" is not zero, then "R1" is wrong or "R1" should not exist.

Incorrect degrees of certainty may be due to a small or an atypical case library which is used to construct the KB or the bias of the KB builders. In our experience, degree of certainty allows an error of about ".15" (also refer to [Buchanan and Shortliffe 84]). Therefore, the following two rules are in accord:

```

      .7
"A1 -> class A"
      .6
"A1 -> class A"

```

The following two rules contradict each other:

```

      .7
"A1 -> class A"
      -.5
"A1 -> class A"

```

If one rule represents the truth, the other is erroneous. As described before, that two rules differ in their conclusions but overlap in their premises is not a real conflict.

5.2.2.4. Missing Rules

Either no conclusions or incorrect conclusions may imply some rules are missing. Had these missing rules been applied, errors would not have occurred.

5.2.2.5. Subsumption

One solution is to write rules in a mutually exclusive way so that they won't succeed simultaneously [Shortliffe 76]. In the following example, rule R1 subsumes rule R2:

```

      f1
R1: A1 & A2 -> class A
      f2
R2: A1 -> class A

```

The solution is modifying R2 into R3 as follows:

f3
R3: A1 & -A2 -> class A

5.2.2.6. Redundancy

Redundancy occurs if two rules are identical in their premises and conclusions, and the difference of degree of certainty is trivial ($< .15$).

5.3. Fault Corrections

There are several operators to correct faults in the knowledge base. The types of faults and their corresponding correction operators are summarized as follows:

<u>Faults:</u>	<u>Correction operators</u>
1. Overly generalized rules	<ul style="list-style-type: none"> * Adding conditions * Replacing conditions * Closing interval (in JAUNDICE)
2. Overly specialized rules	<ul style="list-style-type: none"> * Replacing conditions * Deleting conditions * Splitting rules * Taking minimum or maximum (in JAUNDICE)
3. Erroneous rules	<ul style="list-style-type: none"> * Deleting rules * Changing degree of certainty
4. Missing rules	<ul style="list-style-type: none"> * Adding rules

In this section, we only describe how these operators generally work; the actual implementation in JAUNDICE is described in Section 5.4.

"Adding conditions" operator, one of specialization operators, searches for the most appropriate conditions to add in the LHS of the rule. By increasing the number of constraints, the rule becomes more specialized (i.e., harder to be satisfied). The conditions

chosen to add should be consistent with some positive instances and inconsistent with (most) negative instances. For example,

```
instance1 with attributes A1, A2, classified as class A
instance2 with attributes A1, A3, classified as class B
```

Then, the rule "A1 -> class A" is overly generalized, and causes false prediction of instance2.

And, this rule may be specialized into "A1 & A2 -> class A".

"Maximally general specialization" can prevent specialization operators from excessive specialization.

"Deleting conditions" operator, one of generalization operators, searches for the most appropriate conditions to delete in the LHS of the rule. By deleting conditions, the rule becomes more general (i.e., easier to be satisfied). The LHS of the rule after applying this operator should be more consistent with the positive instances (namely, more positive instances can satisfy), and still be inconsistent with (most) negative instances. For example,

```
instance1 with attributes A1, A2, classified as class A
instance2 with attributes A1, A3, classified as class B
```

Then, a rule "A1 & A2 & A3 -> class A" will be too specific for instance1.

And, the rule may be generalized into "A1 & A2 -> class A".

"Maximally specific generalization" can avoid excessive generalization by generalization operators.

"Replacing conditions" operator replaces some conditions in the LHS of the rule by more specific or more general conditions in order to make the rule more specific or more

general. This can be conducted by climbing down or up the generalization hierarchical tree.

"Splitting rules" operator is a special case of generalization operators. For instance, a rule "A1 & A2 -> class A" can be split (or say generalized) into "A1 -> class A" and "A2 -> class A".

"Turning conjunction into disjunction" is equivalent to "splitting rules". "Turning disjunction into conjunction" is equivalent to "adding conditions". "Closing interval" and "taking minimum or maximum" are described in Section 2.1.1.

The general procedure for correcting a misconception is summarized as follows (as will be described in more detail in next section):

1. Find the relevant rules in the knowledge base.
2. Correct the faults in the KB by the following steps:
 - Generalize those rules which should succeed but fail by "deleting conditions" or "replacing conditions" operator.
 - Specialize those rules which should fail but succeed by "adding conditions" or "replacing conditions" operator.
 - Delete or change the degree of certainty of erroneous rules.
 - Add rules if they are missing in the knowledge base.

In fact, each fault correction operator is a search operator, searching for the most plausible solutions. Consideration of the potential huge search space created by applying all possible operators to all possible rules motivates the development of the strategy of

"retrospective inspection after learning". With this strategy, the rules which can rectify the misconception are first found, and the comparison of the learned rules with the old rules will provide hints of knowledge base modifications. Since the goal is to correct the misconception, learning rules on the basis of the misconcluded case can be viewed as a goal-oriented approach. If we apply the learning method of "search from the most general hypothesis", which actually successively applies specialization operators (described in Section 2.2), there will be only one search space. In contrast, if we start from many old rules and try to modify them by expanding a search space for each rule, the cost is expected to be much higher unless there are only a small number of rules involved. Furthermore, missing rules can be found only by learning. Thus, this approach becomes even more useful in the incipient stage of knowledge base construction, when there are lots of missing rules.

5.4. Automated Debugging

Figure 5.1 shows the data and knowledge flow in the process of automated debugging.

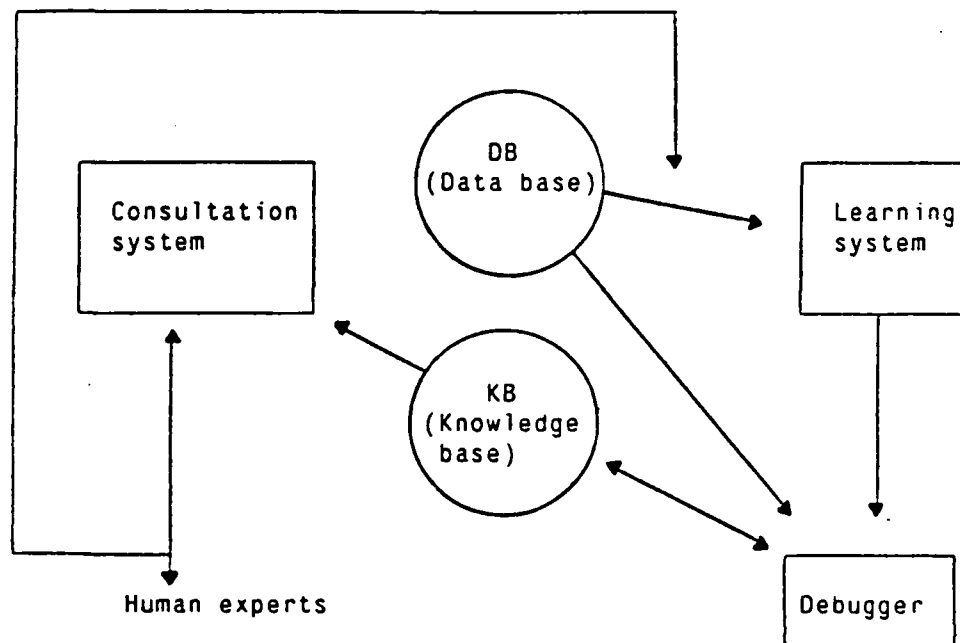


Figure 5.1 Overview of the automated debugging mechanism in the JAUNDICE program.

When a misconception occurs, as indicated by a knowledge source (e.g., experts), the learner will first be triggered to learn rules with respect to the misconcluded case, then the debugger proposes experiments for modifying the KB, based on the comparison between the learned rules and the old rules, and the proposal will be accepted or rejected, depending on the result of verification over the old reference cases and on whether the misconception can be rectified. There may be more than one iteration if the first proposal is rejected. In each iteration, the learning system may adjust its selection criteria and thus change the quality of learned rules; however, the debugger only inflexibly applies some rules (described later) to propose modifications.

There are two sources of reference: the knowledge source which indicates the misconception and provides a correct answer, and the old cases in the DB. The acceptable result is the consistency between these sources: i.e., the modified KB can rectify the misconception without degrading the performance with respect to the old reference cases. This double-check can make the debugging result more reliable. Sometimes, if the data of the misconcluded case are incomplete or the expert instead of the system gives a wrong conclusion, the learner may find no good rules to send the debugger.

5.4.1. Fault Analysis

The diagnosis given by the consultation system is called "system's diagnosis", and that given by the expert (assume the user is an expert; otherwise the user will be forbidden to give his diagnosis) is called "expert's diagnosis". Since uncertainty is involved, the system will actually return a list of disease diagnoses which are ranked according to the corresponding degree of certainty; and only those diagnoses with significant degree of certainty are returned to the user (the expert). The expert often gives one disease diagnosis that he believes most. Though, sometimes, the expert may give more than one diagnosis if he can not make further distinctions; however, this is not a good case from the learner's point of view; in learning from examples, each training instance is labelled as either positive or negative instance, but not both. If the expert doesn't agree to the system's conclusion, the automated debugging process will be initiated. The system's diagnosis is said to match the expert's diagnosis if and only if both of the following are true (assume the expert gives only one disease diagnosis):

1. The system's top diagnosis is the same as the expert's diagnosis.
2. The system's top diagnosis is as certain as the expert's belief.

Example 1. The system's diagnosis:

Disease A	.6 (degree of certainty)
Disease B	.3
Disease C	.1

The expert's diagnosis:

Disease A	.9
-----------	----

<comment:>

In this case, the system's top diagnosis, though the same as the expert's diagnosis, is less certain than expert's belief (the difference is greater than ".15", the precisional error).

<possible remedy:>

Raise the degree of certainty of Disease A by the means described later.

Example 2. The system's diagnosis:

Disease A	.6
Disease B	.3
Disease C	.1

The expert's diagnosis:

Disease B	
-----------	--

<comment:>

In this case, Disease B is the most certain diagnosis given by the expert, though he didn't give his belief about it. And the system top diagnosis does not match the expert's diagnosis.

<possible remedy:>

Raise the degree of certainty of Disease B, Reduce the degree of certainty of Disease A, such that Disease B can override Disease A, by the means described later.

In a given case, for a given diagnosis, the degree of certainty can be raised by the following ways:

1. Generalize the partially satisfied confirming rules which conclude the diagnosis so that the rules can be satisfied and the diagnosis can be more confirmed.
2. Specialize the satisfied disconfirming rules which deny the diagnosis so that the rules won't be satisfied and the diagnosis can be less disconfirmed.
3. Raise the degree of certainty of the satisfied confirming rules so that the diagnosis can be more confirmed.
4. Reduce the degree of certainty of the satisfied disconfirming rules so that the diagnosis can be less disconfirmed.
5. Add new rules (or missing rules) which conclude the diagnosis and can be satisfied by the given case so that the diagnosis can be more confirmed.
6. Delete the satisfied erroneous disconfirming rules which deny the diagnosis so that the diagnosis can be less disconfirmed.

On the contrary, the degree of certainty can be reduced by the following ways (opposite to those ways for raising degree of certainty):

1. Generalize the partially satisfied disconfirming rules which deny the diagnosis so that the rules can be satisfied and the diagnosis can be more disconfirmed.
2. Specialize the satisfied confirming rules which conclude the diagnosis so that the rules won't be satisfied and the diagnosis can be less confirmed.
3. Raise the degree of certainty of the satisfied disconfirming rules so that the diagnosis can be more disconfirmed.
4. Reduce the degree of certainty of the satisfied confirming rules so that the diagnosis can be less confirmed.
5. Add new rules (or missing rules) which deny the diagnosis and can be satisfied by the given case so that the diagnosis can be more disconfirmed.
6. Delete the satisfied erroneous confirming rules which conclude the diagnosis so that the diagnosis can be less confirmed.

However, as will be described next, the method developed here will not exhaustively exploit the above operators.

5.4.2. Application of Machine Learning

The combinations of possible fault corrections described in last section can be great. However, the complexity can be reduced if we handle those invoked and non-invoked rules separately. Since the invoked rules are often a small subset of the KB, we may examine them exhaustively. In contrast, exhaustively examining the non-invoked rules is inefficient because they are often so many and there is no way to examine missing rules. Instead, we apply the learning technique (focusing mode of learning, described in Section 2.2.4) to learn rules, based on the misconcluded case, and compare the learned rules with the old rules to determine the modifications of the KB.

The procedure is described as follows:

- step 1. Examine the invoked rules: check the degree of certainty and check whether they break the pre-defined constraints for minimal generality and specificity.
 - If they are sound, do nothing.
 - Otherwise, optimize the potentially erroneous rules according to the procedure described in Section 4.5.3.2. Note that, after optimization, the rules which are initially satisfied by the misconcluded case may become unsatisfied. Sometimes, an activated rule is desired to be inactivated (usually by specialization) in order to remedy the misconception, but this is allowed only if favored by the old reference cases; consequently, we use optimization instead of simply specialization to tackle this problem.
- step 2. Apply focusing mode of learning, based on the misconcluded case.
 - Learn confirming rules to support the expert conclusion (see Section

2.2.4), and learn disconfirming rules to disfavor the system misconception (see Section 2.4).

- Compare the learned rules with old rules, propose experiments for modifying the KB, and verify them (see next section).

In our model, the KB is constructed initially on the basis of a set of training instances; as the database accrue, the statistics may shift, and a rule, sound in one time, may become ill in another. The step 1 is intended to adapt the KB to this temporal change. However, as the statistics converges, this possibility will decline.

5.4.3. Retrospective Inspection after Learning

After learning rules based on the misconcluded case, the subsequent stage is to determine how to integrate those rules into the KB. In our model, the newly learned rules are based on the current case library; they may not necessarily be consistent with the old rules, which are learned based on the initial case library. Therefore, it is necessary to check the consistency between the newly learned rules and the old rules. If one newly found rule is incompatible with one old rule, it is plausible to replace the old rule by the new one because the new rule is consistent with more cases than the old one. The experimentations and verifications described in the following sections are designed to assure the replacement of old rules by new rules or adding new rules if they are missing is proper. The experiments are proposed on the basis of the comparison between the new and the old rules. One newly learned rule will be compared with one old rule if and only if they share a common RHS.

5.4.3.1. Experimentations

There are four experimental rules. The first experimental rule is used for correcting overly specialized rules in the knowledge base:

- ER1: If
1. One learned rule is more general than one rule in the knowledge base.
 2. The difference of degree of certainty is trivial (i.e., $<.15$).

Propose: Replace the rule in the KB by the learned rule.

Since a learned rule has been "optimized" within the same range of degree of certainty with respect to some criterion (i.e., weighted prediction error) in the current database, the old rule should be updated if indicated. If the degree of certainty differs, it is incomparable; in EMYCIN-based systems, more specialized rules are often associated with higher degree of certainty. That "rule R1 is more general than or as general as rule R2" means "whenever the LHS of R2 is true (or satisfied), the LHS of R1 will also be true (or satisfied)". This is detected when the following two conditions exist:

1. The features in the LHS of R1 are a subset of features in the LHS of R2.
2. For each feature in the LHS of R1, its value is the same as or more general than the value of the same feature in R2.

R1 will be more general than R2 if the above conditions are satisfied and R1 is not the same as R2. One example of applying the rule ER1 is as follows:

```

One learned rule,      LR1: A1 .7 -> class A
One rule in the KB,   R1: A1 & A2 .6 -> class A
Then, R1 might be replaced by LR1.
```

The second experimental rule is used for correcting overly generalized rules:

- ER2: If**
1. One learned rule is more specific than one rule in the knowledge base.
 2. The difference of degree of certainty is trivial (i.e., $< .15$).

Propose: Replace the rule in the KB by the learned rule.

Again, because a learned rule has been optimized, it is plausible to update the old rule according to the learned rule. "More specific" is the opposite of "more general". "R1 is more general than R2" is equivalent to "R2 is more specific than R1"; therefore, we can detect "more specific" by the same means we detect "more general" as above. One example of applying ER2 is as follows:

One learned rule,	LR2: A1 & A4 \rightarrow class A	.7
One rule in the KB,	R2: A4 \rightarrow class A	.6

Then, R2 might be replaced by LR2.

The third experimental rule is used to delete erroneous rules:

- ER3: If**
1. One learned rule is contradictory to one rule in the knowledge base.

Propose: Replace the rule in the KB by the learned rule.

As mentioned earlier, the newly learned rule is consistent with more cases than the old rule, it is plausible to apply this rule. For two given rules whose LHS and RHS are both the same, if the difference of their degree of certainty is trivial (less than .15), then they are "redundant" with respect to the other; otherwise they are "contradictory". If two rules share a common LHS, their RHS are mutually exclusive, and the degree of certainty of at least one of them is "1", then they are contradictory. This rule is exemplified as follows:

One learned rule,	LR1: A1 \rightarrow class A	.7
-------------------	-------------------------------	----

One rule in the KB, $R3: A1 \rightarrow \text{class } A$

Then, $R3$ might be replaced by $LR1$.

The last experimental rule is used to treat missing rule:

- ER4:** If
1. One learned rule can't be applied by $ER1$, $ER2$, or $ER3$.
 2. The learned rule is not redundant with respect to any rule in the KB.

Propose: Add the learned rule in the KB

If the learned rule is redundant, do nothing to the KB. Finally, if subsumption occurs, it can be handled in a way suggested by [Shortliffe 76]. This is illustrated in Section 5.2.2.5.

5.4.3.2. Verifications

The proposed experiments are then verified to see whether the performance is still maintained (or even improved) when applied to the old reference cases and whether the misconception is rectified. The reliability of the knowledge source which indicates the misconception and provides the correct conclusion is an important concern. In medicine, the source can be regarded reliable if it is based on a pathognomonic study or the advice of senior experts.

The experiments are verified as follows:

- If the source is definitely reliable,
 - If the modifications to the KB can rectify the misconception and maintain or improve the performance reflected by the weighted prediction error (defined in Section 4.5.3.1) with respect to the old reference cases, accept the modifications.
 - Otherwise, reject the modifications.

- #### 5.4.4. One Example

NR1: If 1. serum bilirubin is elevated.
2. Colicky rt. upper abd. pain
is present.
then likely (.6) Calculous jaundice.

{Then, the debugger moves to the next step.}

step 3. The debugger finds NR1 is more general than rule "R114" in the KB:

R114: If 1. Serum bilirubin is elevated.
2. Colicky rt. upper abd. pain
is present.
3. Course is recurrent.
then likely (.7) Calculous jaundice

First experimental rule "ER1" is triggered, the result is as follows:

"R114 might be generalized into NR1"

{Then the debugger returns the experimentally modified KB to the performer.}

step 4. The performer (the consultation system) reruns the case.

system's diagnosis: Calculous jaundice .68
Acute hepatitis .4

step 5. Check the weighted prediction error of the old reference cases.

(Assume $w_p=2$, $w_n=1$)

Before changes: $wpe = 2 \times 2 + 1 \times 0 = 4$
(i.e., there are two incorrectly concluded cases and one unconcluded case.)
After changes: $wpe = 2 \times 2 + 1 \times 0 = 4$

Succeed!

{The changes in the KB are accepted.}

5.5. Comparison and Discussion

The main features of the work developed here are summarized as follows:

1. Because the updating is completely automated (except that another knowledge source is required to point out the incorrect conclusion made by the expert system), experimentations and verifications are necessary.

2. It is a part of the complete learning model we develop.
3. It is intended to achieve an optimum with respect to both old reference cases and the new case with incorrect conclusion.
4. It can cope with uncertainty and is designed primarily for EMYCIN-based systems where evidence can be combined.

Based on these features, we choose several related AI programs for comparison in order to reveal the underlying significance.

TEIRESIAS [Davis 79] is the most related work in the sense that it is also designed in MYCIN-like systems and includes a similar bug tracking strategy by examining the relevant rules with respect to a certain case is used. However, because of different assumptions made, the approaches to modifying the KB are quite different. TEIRESIAS debugs the KB, based on the empirical knowledge from a knowledge source (i.e., the human expert) which is assumed to be reliable while in the herein developed method, we don't assume there is a knowledge source which can debug the KB (but we assume there is a knowledge source which can detect the incorrect conclusion made by the system), and the debugging requires some experimentations and verifications to increase its credibility. EMYCIN [Van Melle 80], including a TEIRESIAS-like environment, also provides a similar verification mechanism as our program does. However, the fundamental difference is man- vs. machine-oriented approach. For example, TEIRESIAS relies on expert knowledge to specialize a rule so that it will not succeed, while the automated technique relies on an optimization search to decide whether to specialize, and, if so, how to specialize. If one rule should be generalized so that it can succeed, TEIRESIAS again relies expert knowledge while the automated approach takes advantage of the inductive

concept learning technique to learn rules first and compare them with the old rules to decide which rules should be generalized. If new rules should be added, TEIRESIAS assists the expert to transfer his knowledge while the automated approach again exploits machine learning technique. Moreover, in an inexact domain, automated debugging may achieve a better result than expert-oriented approach in dealing with compromise among correcting multiple faults.

The SEEK program [Politakis 82] is designed to refine the KB by generating experiments and asking the expert's advice. Instead of machine learning employed in our work, the SEEK program generates experiments based on domain-dependent heuristics. Its great limitation is the incapability of finding missing rules. Our approach is comparatively more general and powerful.

The rule checker program [Suwa 84] conducts a systematic and exhaustive checking, which is certainly not the case in our work, in the KB. The apparent limitations of this approach include the following: the search space can't be too large, and most of the possible combinations should be meaningful; otherwise it is very inefficient. The program also doesn't mention handling of uncertainty.

The poker player [Waterman 68] also uses the strategy of "retrospective inspection after learning" as our method does. Perhaps because he assumes there is no inconsistency and uncertainty, his program is intended to maximize the performance of each individual play without re-examining the old plays. In contrast, our method is intended to maximize the performance under considering both old reference cases and the new case. In addition, the analytical technique used by Waterman (neglect the advice taking technique here since

our approach doesn't rely on it) to acquire training rules rests with the proof by backward-chaining via an axiom system. In contrast, we use forward-chaining to handle those activated rules (i.e., try to optimize those activated rules if they are found ill) and backward-chaining (or goal-oriented) to learn rules for the misconcluded case for reasons of efficiency. In more detail, when we apply focusing mode of learning, we treat the misconcluded case as "positive instance" (indicated by the correct conclusion) and the learner is intended to learn rules for it. However, instead of labelling the misconcluded case as "negative instance" to learn rules for classes other than the class indicated by the correct conclusion of the misconcluded case, we simply focus on those activated rules to see whether they accidentally cover the misconcluded case as classes other than the class indicated by the correct conclusion of the misconcluded case. For example, if the misconcluded case should be "Disease A" but it is falsely concluded by "rule R" as "Disease B", then we don't learn rules for diagnosing Disease B by treating the misconcluded case as a negative instance; instead, we simply check whether the rule R is optimal, and, if not, decide how to optimize it, based on the consideration that the falsely activated rules are very limited (often no more than a few).

5.6. Summary

From a long term perspective, the necessity of updating a KB is clear. However, incremental updating should not degrade the performance standard maintained by the old reference cases. Consider various sources of errors and the imprecise nature of reasoning in EMYCIN-applicable domains; it is often hard to add new knowledge monotonically without examining its impact on the old truth value reflected by the weighted prediction error over the old reference cases. Thus, we develop a method which optimizes the KB at

" $t=t_0$ " by minimizing the error in the period from " $t=0$ " when the KB is initially constructed to " $t=t_0$ " when a new case with incorrect conclusion appears: the method will repeatedly be applied every time a misconception occurs. The task is accomplished by learning, proposing experiments based on the four experimental rules described, and verifications.

Chapter 6

Discovery of Meta-Rules

6.1. Introduction

The value of meta-level knowledge for guiding the invocation, construction, and explanation of object-level rules in an expert system has been demonstrated by Davis [Davis 76]. In this chapter, we explore the use of machine learning methods for formulating new meta-level knowledge, extending Davis' ideas about learning rule models. The research reported here aims at learning meta-rules that will guide the rule-based diagnostic system by pruning and reordering the diagnostic rules, as in MYCIN [Davis and Buchanan 77].

We are strongly motivated by the fact that meta-rules are important in systems with large knowledge bases to avoid exhaustive search. Yet, human experts should not be concerned with control issues as much as with domain knowledge, so it is desirable to automate the formulation of control rules.

This is a second-order learning problem (and not a first-order problem of learning new object-level rules) which is defined as:

³³ This chapter is based on the article of [Fu 84].

Given:

A set of object-level rules including:
inferential, causal, and taxonomic knowledge of
the domain.

Find:

Useful meta-rules that improve system efficiency
by guiding the invocation of object-level rules.

This chapter first discusses design considerations and then presents implementation details of the second-order learning system with demonstrations in JAUNDICE with 141 diagnostic rules that diagnoses likely causes of jaundice, and 80 non-diagnostic rules that are linked in a network reflecting a taxonomy among diseases and causal links among events as suggested in [Patil 82], [Wallis 82] and [Clancey and Letsinger 81].

6.2. Learning Meta-Rules: Design Considerations

6.2.1. Format of Meta-Rules

As in [Davis 76], we use two syntactic forms of meta-rules: pruning and reordering forms (see Figures 6.1 and 6.2). In fact, the distinction between these two forms is often blurred semantically. For instance, if we say "Do Rule-Set1 before Rule-Set2", and we succeed in our goal by invoking only Rule-Set1, then Rule-Set2 is pruned anyway. As seen in figure 6.1 or 6.2, the premise of a meta-rule has three parts:

1. The first part is the goal description which can be global or local, and, in our system, the global goal is disease-entity, and local goals include: syndrome, pathophysiological mechanism, etiology, etc.
2. The second part is a conjunction in which each conjunct is a predicate with a triplet of attribute, object, value.
3. The third part is the description of concluded rule sets.

Meta-rule001

If 1. The goal is to conclude the disease of Jaundice.
 2. The indirect type bilirubin is not dominant.
 3. There are rules which mention in their premise "overproduction of bilirubin".
 Then it is definite(1.) that each of these rules is not going to be useful.

Premise:

(\$AND (SAME JAUNDICE GOAL DISEASE-ENTITY)
 (DIFFER LFT DOMINANT-BILIRUBIN INDIRECT)
 (THEREARE OBRULES (\$AND (MENTIONS PREMISE
 OVERPRODUCTION-OF-BILIRUBIN)) SET1))

Action:

(CONCLUDE SET1 UTILITY NO 1.)

Figure 6.1 Example of a meta-rule in pruning form created by META-RULEGEN. Upper part is it's English translation; lower half is it's code in INTERLISP.

Meta-rule079

If 1. The goal is to conclude the disease mechanism of Jaundice.
 2. The Alkaline Phosphatase level in serum is greater than 15 B.U.
 3. There are rules which mention in their action "Cholestasis".
 4. There are rules which mention in their action "Parenchymal-dysfunction".
 Then it is probable(.8) that the former should be invoked before the latter.

Premise:

(\$AND (SAME JAUNDICE GOAL DISEASE-MECHANISM)
 (SAME LFT ALKALINE-PHOSPHATASE >15B.U.)
 (THEREARE OBRULES (\$AND (MENTIONS ACTION
 CHOLESTASIS)) SET1)
 (THEREARE O2RULES (\$AND (MENTIONS ACTION
 PARENCHYMAL-DYSFUNCTION)) SET2))

Action:

(CONCLUDE SET1 DOBEFORE SET2 .8)

Figure 6.2 Example of a meta-rule in reordering form created by META-RULEGEN.

The description of the rule sets in the third parts of meta-rules may be by content or by name. Indirect referencing (by content) is important to maintain flexibility and understandability. The learning program also expands indirect references into a so-called name-referred form with an explicit list of rule names, as seen in figure 6.7.

6.2.2. Utility Consideration of Meta-Rules

The main reason for incorporating meta-rules in a performance system is efficiency.³⁴

Theoretically, it is difficult to say how to measure the efficacy of meta-rules unless we delineate the concept of Utility Value for the meta-rules, which is also important in generating them.

6.2.2.1. Utility Value for Meta-Rules

The Utility Value of meta-rules is based on an analysis of costs and benefits. Intuitively, high Utility Value is associated with high benefit and low cost. We define an absolute utility value based on estimated savings in CPU time and then a relative value to normalize absolute values over object rule sets of different sizes.

Absolute Utility Value of Meta-Rules

The cost of a meta-rule is the estimated CPU time to evaluate its premise. If the premise is true, then its benefit is how much CPU time might be saved by pruning or

³⁴We are not concerned here with the use of meta-rules to guide a dialogue, although human engineering issues are also important. If the meta-rules are effective in pruning unnecessary questions, however, the dialogue will also appear to be better focused. Note also that the term "utility" has different technical meanings in different technical areas. We are concerned here with a measure of importance based on estimated costs and benefits.

reordering object level rules under the guidance of the meta-rule. For simplicity, we first define unit cost to be "average CPU time to evaluate one conjunct (clause) in the premise." Suppose there are "n" conjuncts in the premise of the meta-rule, we define the cost of the meta-rule to be "n" units. (The performance program may cease evaluation once one conjunct appears to be false, but our estimate will assume the worst case.) The benefit will be how many object level rules are pruned out, if the meta-rule succeeds (i.e., if its premise is true). The cost of using one object level rule will include CPU time for evaluating its premise, and, if the premise is true, making a conclusion and doing the bookkeeping. But, considering the least condition (i.e., first conjunct is found to be false, and evaluation is stopped), if there are "m" object level rules which are pruned away by the meta-rule, then the least benefit will be "m" units. Again, our estimate assumes the worst case (i.e., least benefit). Thus, if a meta-rule makes a successful and accurate prediction, then the gain (the ratio of benefit to cost) will be "m/n". Now, the Absolute Utility Value (abbreviated as AUV) is defined for a meta-rule as follows:

$$AUV = b/c_o \times \text{Freq.}(\text{prem.}) \times C_{mR}.$$

where,

b = Number of object rules pruned.

c_o = Number of conjuncts in the premise.

Freq.(prem.) = the estimated frequency with
which the premise is true.

C_{mR}³⁵ = degree of certainty of meta-rule.

This definition of AUV takes into account not only the cost and benefit, but also the frequency with which the premise is true over reference cases in a case library (see figure 6.6) and the degree of certainty of the meta-rule. If the meta-rule rarely succeeds, it will

³⁵C_{mR} ranges from 0 to 1. "Zero" means "unknown", while "one" means "definitely yes". It is obtained from human experts, or can be computed as described in Section 6.3.2.

have low utility. And, even if it succeeds (i.e., the premise is true), the prediction (conclusion) will be very uncertain if the degree of certainty of the conclusion is very low. When the frequency information is incomplete, it can be estimated from a Frequency table, that stores the number of cases (in the case library) for which each single premise clause is true. For instance, if a premise mentions the presence of Attribute1 and Attribute 2, then we can estimate the frequency of the conjunction by multiplying the frequency of Attribute1 and Attribute2 in the reference cases under the assumption (default) of independence. An example of calculating AUV is shown in Section 6.4.

Theorem The threshold value for AUV such that the expected benefit under the worst assumption³⁶ will be greater than zero is:

$$AUV_{\text{threshold}} = 1 + c_p f(1 - C_{mR})/c_o$$

c_p : Penalty owing to the incorrect prediction by the meta-rule.³⁷

c_o : Number of conjuncts in the premise. This is the required cost to evaluate a meta-rule.

b : Number of object rules pruned. This is the benefit when a meta-rule succeeds.

f : Freq.(prem.)

b_{exp} : Expected benefit of the meta-rule.

(proof): C_{mR} can be viewed as the estimated rate of correct predictions out of all predictions. Therefore, if the meta-rule succeeds (i.e., the premise is true) and the prediction is correct, then the net benefit will be " $(b - c_o)$ ", and if the meta-rule succeeds and the prediction is wrong, then the net benefit will be " $(-c_o - c_p)$ ". Otherwise, the net benefit is " $-c_o$ ". Hence, the expected benefit is:

³⁶ As discussed above, the estimate is under the worst assumption, a conservative estimate, so to speak.

³⁷ If C_{mR} is not 1, then the meta-rule may predict wrong sometimes. And, the wrong predictions may cause penalty which depends on the extent the system undoes and re-executes, or the extent of improper reordering.

$$b_{exp} = [(b - c_o)C_{mR} - (c_o + c_p)(1 - C_{mR})]f - c_o(1 - f)$$

Let $b_{exp} = 0$, we get $AUV = bfC_{mR}/c_o = 1 + c_p f(1 - C_{mR})/c_o$, and this is the threshold value $AUV_{threshold}$.

Corollary 1 The minimal threshold, $\min.(AUV_{threshold})=1$.

Corollary 2 $AUV_{threshold}=1$ when $C_{mR}=1$ or $c_p=0$.

Corollary 3 If $AUV \gg AUV_{threshold}$, then AUV can estimate the lower bound of expected benefit.

(proof): from proof of the above Theorem.

$$b_{exp}/c_o = AUV - AUV_{threshold}$$

If $AUV \gg AUV_{threshold}$,

then $b_{exp}/c_o \simeq AUV$.

or $b_{exp} \simeq AUV \times c_o$.

Since $c_o \geq 1$, AUV can estimate the lower bound of b_{exp} .

Relative Utility Value of Meta-Rules

We also define Relative Utility Value (abbreviated as RUV) as follows.

$$RUV = \frac{AUV}{\text{Number of total object rules under the global goal}} \times 100$$

Three important aspects of RUV are:

1. If $AUV \gg AUV_{threshold}$, RUV can estimate the relative improvement (in

percentage) of the overall system performance. Because the overall cost for system execution is parallel to the number of total object rules, and, from corollary 3, AUV can estimate the expected benefit of a meta-rule, the ratio of the two can estimate the relative improvement of performance by a meta-rule.

2. RUV is good for comparison of meta-rules from different systems, which have different numbers of object rules.
3. Because the object rule set usually expands quickly as a performance program is being constructed, RUV is a more important index to maintain useful meta-rules. For instance, if we assume the utility value can estimate the system performance, we might say "keep those meta-rules with RUV more than 10" rather than "keep those meta-rules with AUV more than 10" since "AUV more than 10" may not be significant if there are, say, 1000 object rules.

6.2.2.2. Selecting Useful Meta-Rules

Our heuristics for selecting useful meta-rules are as follows.

- step 1. we use $AUV_{threshold} = 1$. That is, we first retain those meta-rules with AUV greater than "1". From corollary 1, we know if a meta-rule with its AUV less than or equal to "1", it will be useless (i.e., its $b_{exp} \leq 0$); however, if its AUV is greater than "1", it may be useful but not necessarily since $AUV_{threshold}$ is not necessarily "1". Therefore, step 2 is required.
- step 2. we use RUV as a reference and experimental simulations to do further pruning.

Finally, we select a useful set of meta-rules by removing redundant meta-rules³⁸ and from experimental simulations.

³⁸If two meta-rules often succeed simultaneously and their conclusions severely overlap, then they are redundant with each other. Both the frequency threshold and the extent of overlapping are defined heuristically.

6.2.3. Overview of Two Approaches to Learning Meta-Rules

6.2.3.1. From Object Rules

Starting from each object rule, the program uses information about three well-known medical strategies [Miller, Pople, and Meyers 82] to determine if there could be useful meta-rules covering this object rule and related ones.

- 1) Rule-out mechanism: If there exists enough evidence contradicting a fact, then we don't bother trying to confirm it or deduce other facts from it. If some evidence is against a fact, then we attempt to form a hypothesis: "That fact and all possibly associated facts with it may be ruled out based on that evidence", and if this hypothesis is justified by some evaluation criteria (e.g., Utility Value), then we succeed in our attempt.
- 2) Rule-in mechanism: This is the inverse of the Rule-out mechanism. It says we should consider certain facts first if some evidence implies doing so and this consideration is valuable with respect to some evaluation criteria.
- 3) Differential mechanism: Physicians are often involved with the issue of differential diagnosis, which is basically confirming one diagnosis from a set of possibilities to the exclusion of the others. In part, this is the consideration that when there is more evidence suggesting Disease1 than Disease2, it will be reasonable to confirm Disease1 before Disease2.

6.2.3.2. From Attributes

If one conjunct appears many times in the premise of object level rules, then it might be worthwhile to evaluate this conjunct first. If this consideration proves valuable, then we keep it. A similar syntactic approach can be found in [Davis 76] and [Van Melle 80]; however, the difference is our explicit consideration of utility, as described in Section 6.3.

- b. Affirming-tree (see figure 6.4): a network of confirming associations. If one node is suggested, then, by following the links in the affirming tree we know the degree of implication of other nodes. The affirming tree is

constructed by recording pairs of individual facts that are positively linked in the object-level rules. For example, overproduction of bilirubin confirms (.95) hemolysis.

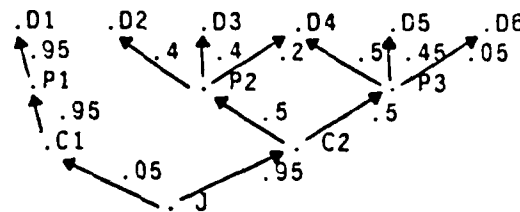


Figure 6.4 Part of AFFIRMING-tree in META-RULGEN. same notations as in Figure 6.3.

- c. List of differential pairs (or groups) and mutually exclusive pairs (or groups) (see figure 6.5): lists of incompatible diagnoses and findings. This list is constructed by recording pairs of individual facts that are incompatible (and often easily confused) with each other as reflected by the object-level rules. For example, hemolysis should be differentiated from Gilbert's disease (congenital conjugation defect), because they are similar in the aspect that urine bilirubin is negative.

```

((Hemolysis   Congenital-conjugation-defect)
 (Hepatitis   Calculous-jaundice)
 (Primary-biliary-cirrhosis Calculous-jaundice)
 .....
 ..... )
  
```

Figure 6.5 List of differential or mutually exclusive pairs (or groups).

In these three structures, the current implementation allows no conjunction or disjunction in each node, i.e., each node represents a single fact.

3. Heuristics, which underlies the whole procedure for learning meta-rules (see Section 6.2.3). The frequency table which is constructed by recording frequencies of attributes over the cases in the case library (see figure 6.6) is also used to guide the program.


```
( (Acute-course .5)
  (Malaise .7)
  (Chills .08)
  (Marked-body-weight-loss .1)
  .....
  ..... )
```

Figure 6.6 Frequency Table. Each sublist contains an attribute with its frequency in the case library.

6.3.2. Algorithm

The search space of all possible meta-rules is roughly equal to the number of combinations of legal attribute-value pairs in the existing set of object rules. If all attributes were binary, this is the power set over n attributes. In the JAUNDICE program, n is 56 attributes in the 141 diagnostic rules (some attributes have binary values, and others have multiple values). The search is greatly constrained from the start by setting up the affirming tree and denying tree, which represent positively and negatively confirming links among attributes (and values) already noticed in the set of object rules.

There are two different parts of the algorithm, both of which are exercised. In our experiments we have taken the union of the two sets of meta-rules as the result. The algorithm is described for the separate approaches. Refer to Section 6.2.1 for descriptions of the three parts of a meta-rule.

6.3.2.1. Approach from Object Rules

Form meta-rules on the basis of each individual object rule (of 141 diagnostic rules) as follows. Collect all object rules and form a set S. Take out the first element from S and do the following procedures; also delete this element from S.

- Step 1. Form the second part of the premise (conjunction of predicate with attribute- object- value triplet) on the basis of the premise part of the object rule. From the discussion in Section 6.2.3, we note that a piece of evidence (premise in the object rule) is a plausible starting point to generate meta-rules.
- Step 2. For different conditions:
 - a) Formation of pruning form meta-rules
 - i) If the object rule confirms some fact, by climbing the affirming tree, we know what other facts also are implied, with degrees of certainty calculated by propagating the uncertainty along the tree.³⁹ Thus, the third part of the premise will be those rule sets mentioning "presence of these facts", and these rule sets will be concluded to be useful in the action part of the meta-rule with degree of certainty calculated as described (see "Rule-in mechanism" in Section 6.2.3.1).
 - ii) If the object rule disconfirms some fact, by climbing the denying tree, we know what other facts are also denied, with degrees of certainty calculated by propagating the uncertainty along the tree. Thus the third part of the premise will be those rule sets mentioning these facts, and these rule sets will be concluded to be useless with degree of certainty calculated as described. (see "Rule-out mechanism" in Section 6.2.3.1)
 - Note that in both (i) and (ii), more than one meta-rule will be formed. There is a merging process in step 5.
 - b) Formation of reordering form By looking at the list of differential pairs, we

³⁹ For example, if A implies B with degree of certainty .4 and B implies C with degree of certainty .6, then A implies C with degree of certainty $.4 \times .6 = .24$

know, for instance, Fact1 should be differentiated from Fact2. If the object rule confirms Fact1, then under the premise of this object rule, Fact1 should be pursued before Fact2. Thus, the third part of the premise will be rule sets mentioning Fact1, called Set1, or Fact2, called Set2, and it is concluded in the action part that Set1 should be invoked before Set2 with degree of certainty approximately equal to the degree of certainty of the object rule. Similarly, we can figure out the process if the object rule disconfirms Fact1 (see "Differential mechanism" in Section 6.2.3.1).

- Step 3. Form the first part of the premise (the goal description) on the basis of the mentioned facts in the third part of the premise (the description of rule sets). For example, in Meta-rule079 (figure 6.2), the reordering of two facts: "cholestasis" and "parenchymal dysfunction" has to do with the conclusion of the subgoal "disease mechanism".
- Step 4. Calculate the Utility Value for each newly formed meta-rule, and filter out those with AUV being less than 1.
- Repeat the whole procedure until the set S is empty.
- Step 5. Meta-rules are further selected, based on RUV and experimental simulation results. That is, we first set a cut-off point with respect to RUV to select meta-rules; those selected rules are then tested by experiments (see Section 6.4). Then, merge the meta-rules. If two meta-rules have the same first and second parts of the premise and their conclusions have no (or little) overlapping with respect to the concluded rule sets, then (a). add their descriptions of concluded rule sets to form a description of a new rule set and (b). add their conclusions (actions) to form a new conclusion (action). The Utility Value of the merged meta-rules is defined as the sum of the Utility Value of the individual rules before merging. Also, the redundant meta-rules are removed.

Example 1.

In the following descriptions, the notation "-" means "absent" or "denied".

A set of object-rules:

.7
R1: A1 -> -S1

.6
R2: A2 -> S2

.....

.....

A part of denying tree:

1.
-S1 -> -M1
1.
-S1 -> -M2

Form potential meta-rules on the basis of R1:
By propagating implications,

.7
A1 -> -M1

.7
MR1: A1 -> Rules mentioning M1 are useless
Similarly,

.7
MR2: A1 -> Rules mentioning M2 are useless

(Note that "MR1" can be interpreted as:
"If attribute A1 is present,
then any rule mentioning presence
of M1 will be useless,
with degree of certainty .7",
since M1 is denied by A1.)

Calculate the AUV as shown in Section 6.4.

If both MR1 and MR2 are determined to
be retained after selections and their
conclusions are little overlapped,
then they are merged as:

.7
"A1 -> Rules mentioning M1 are useless
.7
-> Rules mentioning M2 are useless."

<Note:>

If a part of the differential list is:

"((M1 M3))",

then a potential reordering meta-rule can be
formed:

.7
MR3: A1 -> Rules mentioning M3 DOBEFORE
Rules mentioning M1.

since M1 is denied by A1.

6.3.2.2. Approach from Attributes

Collect all parameters (attributes) to form a set S. Take out the first element and do the following procedures; also delete this element from S.

- Step 1. Form the first part of the premise. This is usually the main goal of the system, if there is only one main goal.
- Step 2. Form the second part of the premise by "presence of the parameter", and collect all object rules whose premise fails immediately because of "presence of the parameter". Thus, the third part will be those rules mentioning the presence of the parameter in their premise, and it is concluded in the action part that these rules will be useless with degree of certainty 1.
- Step 2'. Form the second part of the premise by "absence of the parameter", and do the similar procedure as in Step 2. (Note: we try to form two meta-rules for each attribute with this approach.)
- Step 3. Calculate the Utility Value of each newly formed meta-rule, and filter out those with AUV being less than 1.
- Repeat the whole procedures until the set S is empty.
- Step 4. Meta-rules are further selected by RUV and verified by experimental simulations.

Example 2.

A set of object-rules:

R1: A1 & A2 -> S1

R2: A1 & A3 -> S2

.....

.....

Form potential meta-rules on the basis A1

MR1: -A1 -> R1, R2, ... useless.

Calculate AUV

.....

.....

6.4. Results

Sixty-three rules were created by META-RULEGEN. About ten were formed by the approach from attributes and the remainder by the approach from the 141 rules in a preliminary version of the JAUNDICE system. About fifteen rules were reordering rules (all formed by the approach from object rules) and the remainder were pruning rules. (Figures 6.1 and 6.2 show one pruning rule and one reordering rule produced by the program.) We expand the rule sets into explicit lists of rules (figure 6.7) to compute Utility Values of MR01 in figure 6.1 as follows.

$$\text{AUV as: } \frac{20}{1} \times .97 \times 1 = 19.4 \text{ and}$$

$$\text{RUV} = 19.4 \times \frac{100}{141} = 13.76$$

Premise:
 (SAND (DIFFER LFT DOMINANT-BILIRUBIN INDIRECT))
 Action:
 (CONCLUDE (R79 R78 R61 R20 R19 R18 R17 R16 R13
 R12 R11 R10 R9 R8 R7 R6 R3 R109
 R117 R118))
 UTILITY NO 1.)

Figure 6.7 Name-referred form of meta-rule in Figure 6.1, in which the intensive definition of the concluded rule set is replaced by an extensive definition.

Table 6.1 shows the distribution of Utility Values among the 63 meta-rules. We informally confirmed that Utility Values are a reasonable standard, by looking at the medical significance of meta-rules (as found in the literature). We found that meta-rules with high Utility Values usually have high medical significance and conversely.

Table 6.1 Distribution of R.U.V. of 63 meta-rules created by META-RULEGEN.

	RUV			Total
	< 5	5-10	> 10	
Number of meta-rules	48	8	7	63

In addition, we performed a simple experiment to determine the effect of using some of these meta-rules in the JAUNDICE program. We selected 20 representative cases (non-randomly, but preserving the relative frequencies of diagnoses) among 72 cases collected from the literature, and ran them in batch mode in the JAUNDICE program. We measured the efficiency before and after incorporating different meta-rules. Table 6.2 shows selected portions of the outcome from which we see that the predicted Utility Value generally parallels the observed enhancement.

Table 6.2 Relationship between Utility Value and enhancement of system efficiency as determined from 20 cases run in JAUNDICE program (batch mode) with selected meta-rules.
(Lisp time: Lisp interpretation time, not compiled)

	A.U.V.	R.U.V.	LISP time (sec.)	enhancement (percentage)
with MR01	19.4	13.76	271	15.3%
with MR34	13.07	9.27	293	8.4%
with MR40	11.5	8.1	298	6.9%
with MR07	1.94	1.38	330	-3.1%
with MR01 & MR34			256	20%
with MR01 & MR26 & MR37 & MR40 & MR45 & MR46			121	62%
without metarules			320	NA

The most important result suggested in table 6.2 is the *additivity*⁴⁰ of two non-overlapping (non-overlapping of their pruned rule sets) meta-rules (e.g., MR01 & MR34). This important property can be proved-formally (yet, we neglect the proof here), and it indicates that, by carefully selecting a set of useful meta-rules, overall system performance can be improved greatly (e.g., MR01 & MR26 & MR37 & MR40 & MR45 & MR46). However, there seems to be a limitation of the enhancement by combining several meta-rules.

⁴⁰That is, the benefit of using two meta-rules A and B is (approximately) equal to the benefit of using A plus that of using B.

Improved efficiency is only desirable in our system if there is no significant loss in performance. Thus, we compared the quality of the performance with and without meta-rules by asking whether the top disease diagnosis given by the system was the same as the expert's diagnosis and whether the associated degree of certainty was "close" (i.e., within .15) to the expert's confidence if the top diagnoses given by the system and the expert are matched.⁴¹ The results show nearly complete coincidence: the only one imperfect match exists in an ambiguous case, in which two top diagnoses are given without meta-rules and only one top diagnosis is given with meta-rules. These results are expected because the meta-rules simply reorder the invocation of object-rules. If no certain conclusions are made, all object-rules will be activated anyway.

6.5. Conclusion

Intelligent control of inferences is important in knowledge-based systems for reasons of efficiency and human engineering, especially when knowledge bases become very large. In both cases, focusing the attention of the performance program can be accomplished by reordering and pruning elements of the knowledge base before invocation.

We have presented a general method for discovering meta-level knowledge that can be used to control inferences of an underlying performance program. The demonstration of the method is in terms of a rule-based representation of both object-level and meta-level knowledge, but we believe there is nothing specific to a rule-based representation in the method itself. The method depends only on representing the elements of the knowledge

⁴¹In the SEEK program [Politakis 82], the top model conclusion was compared with the expert's conclusion; however, the experts' confidence is not considered in comparison.

base in a network, with each node representing a fact and each link representing an inferential (or evidential) relationship between facts. In the case of a MYCIN-like rule base this means constructing three additional knowledge structures from the rules themselves: an affirming tree, a denying tree and a table of differentials. These are explicit networks derived from the object-level rules showing facts that are positive evidence for other facts, negative evidence for other facts, or means of discriminating between two facts.

These three knowledge structures are analyzed in order to determine sets of nodes and links in the whole inference network that can be safely ignored in some contexts because they are seen to be irrelevant (i.e., false in those contexts). Similarly, the analysis can show parts of the network (sets of rules) that should be examined before other parts because that will increase efficiency.

From another viewpoint, the analysis mechanisms (Rule-in, Rule-out and Differential) are conditional search strategies because they help to select from a large stored knowledge source the best knowledge to apply. Meta-rules are just heuristics to select good and useful object rules, and Utility Value is one criterion for weeding out heuristics. Although we have designed our system to use degrees of certainty, an exact system without uncertainty can also be handled. It is an extreme case of a system with uncertainty in which uncertainty is quantized into two levels: True and False.

In summary, the concepts described in this chapter can be easily extended to other AI systems because:

- Mechanisms such as, Rule-in and Rule-out, partition the reasoning network

into smaller sets of knowledge and remove the useless ones. They can serve as the basis of forming both *control* and *search* strategy.

- Additional knowledge structures separate confirming and disconfirming links in the inference network. The nodes in these structures can be any fact in the world.
- The rules are not necessarily written in a MYCIN-like format. Moreover, knowledge can be represented in other ways, for instance, in a semantic net.
- The method can be extended to domains without uncertainty by using only two levels "Yes" and "No" to measure the certainty.

Chapter 7

Results and Conclusions

A learning model has been developed that is capable of constructing a knowledge base of rules from a case library and continuously updating it to accommodate new facts. This model is particularly designed for a domain with considerable complexity, reflected by the necessity of expertise for solving problems. Reasoning in such a domain often involves uncertainty and complex evidential resolutions; or in rule-based systems, it implies multiple interacting rules assigned with different levels of uncertainty (or partial certainty). Some practical considerations, such as efficiency and error handling, are also explored as much as possible.

We further develop a method of learning meta-rules from object-rules: this is what we call "hierarchical learning". The learning model thus implicates completeness not only along the time axis (i.e., the KB can continuously be updated) but also along the knowledge hierarchy.

The experiments with the system called JAUNDICE, which embodied all the developed ideas, can serve as a good demonstration of the significance behind this model. This chapter describes the experimental results, the issue of validation, some lessons learned during the experiments, and the implicated future work.

7.1. Results of Learning in JAUNDICE

In the jaundice experiment, we constructed a hierarchical knowledge base by the RL program described in Chapter 2 from a training set of 72 jaundice cases collected from the medical literature. The automatically constructed knowledge base has 232 rules including 112 intermediate rules (rules involved with intermediate concepts). We then compared this new knowledge base with an old knowledge base of 141 rules, that was built by encoding medical knowledge from textbooks and journals and is also hierarchically structured. The comparison is done by using part of the program of automated debugging described in Chapter 5. The result is shown in table 7.1.

Table 7.1 Classification of 232 new rules learned from a case library of 72 Jaundice cases with the learning method developed in Chapter 2.

	No. of Rules
worth keeping	163
conflicting with old rules	1
more general than old rules	5
more specific than old rules	2
exactly same as old rules	33
not acceptable for medical reasons	28
Total	232

From the above table, it is seen that 33 rules in the original knowledge base are exactly rediscovered by learning. It should be stressed here that it is not necessary to rediscover all expert-generated rules by machine learning in medicine because even two experts may write down different sets of rules with equally diagnostic power. The determination of which rules should be retained is not a simple issue. It is not only a matter of looking up the textbook or literature but a matter of human judgement. It is often difficult to find a piece of textbook description that is identical with a rule learned by machine; the rule can still be true after integrating all the related knowledge.

We then conduct a comparison with respect to the prediction power, which we think is a much better quality measurement. First, we tested each knowledge base by the original 72 cases; the diagnostic accuracy of the new vs. the old knowledge base is 97.2% vs. 84.7%. But since the new knowledge base is based on these 72 cases, its better performance is somewhat expected. Therefore, we further tested the knowledge base by 68 other cases obtained from Stanford Medical Center; these cases received liver biopsy in 1978 and were *not all* diagnosable from clinical parameters alone. The diagnostic accuracy of the new vs. old knowledge base is 72.1% vs. 76.5%. If we remove all non-diagnosable cases among these 68 cases, we get 42 diagnosable cases (by "diagnosable", we mean the pre-biopsy diagnosis made by the physician who sent the biopsy coincides with the biopsy diagnosis. Note that not every clinical case is clinically diagnosable because a disease may be in its incipient stage without full manifestation); and the diagnostic accuracy is 83.3%

vs. 88.1%⁴² (see table 7.2). The results indicate the new KB is comparable with the old KB. The discrepancy may be ascribed to the fact that many more cases than 72 are needed to learn rules for even a well-circumscribed domain. Textbooks, after all, encode summaries of considerably more experience.

Table 7.2 Diagnostic accuracy of automatically learned rules.

	Old KB (141 rules-manually encoded from textbooks)	New KB (232 rules-automatically learned)
Training set for automatic learning (72 cases)	84.7%	97.2%
Test set (68 cases)	76.5%	72.1%
Test set with clinically diagnosable* cases only (42 cases)	88.1%	83.3%

*: Among the 68 test cases, 42 cases are diagnosable clinically (refer to text descriptions).

If we turn off the intermediate knowledge learner and learn only direct rules (i.e., only

⁴²If we assign a correct conclusion a quantity "1" and an incorrect conclusion or non-conclusion a quantity "0" for each test case and we use a statistical technique called "paired t test" (refer to [Croston, Cowden, and Klein 67]) to determine whether there is a significant difference between the old and the new KB for making conclusions, the result is " $t = 1.434$ ", which indicates the *null hypothesis* is accepted, or there is no significant difference.

step 1 described in Section 2.5 is turned on), we obtain a knowledge base of 185 (direct) rules; this knowledge base without intermediate knowledge can save execution time⁴³ to some extent if compared with the knowledge base of 232 rules (recall that the average system execution time is roughly proportional to the number of rules in the knowledge base), but the diagnostic accuracy tested by the 42 diagnosable liver biopsy cases drops to 61.9% (vs. 83.3% if intermediate knowledge is added). Here, we may notice there is a tradeoff between execution time and quality of performance. We further notice that cases which can be diagnosed correctly by the knowledge base with intermediate knowledge and cannot be diagnosed correctly without intermediate knowledge are cases with incomplete data. It seems clear that intermediate knowledge can improve the system prediction power particularly if only partial information is available. Moreover, intermediate knowledge provides much better understandability and explanation capability. For instance, in our experimental domain, the incorporation of intermediate knowledge can explain the underlying pathological and anatomical mechanisms of jaundice and make the diagnosis more convincing. Based on the above considerations and discussions, learning intermediate knowledge is justified and desirable in expert systems.

Shown in figure 7.1 are two well-known medical rules related to jaundice actually rediscovered by the program.

⁴³ The execution time is closely related to the acceptability of an expert system. In medicine, physicians are often impatient to get answers; in real-time situations, the decision making must be quick.

AD-A171 794

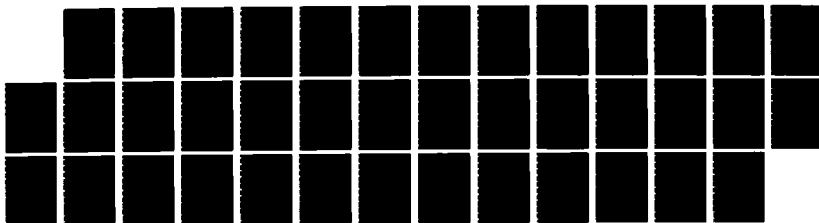
LEARNING OBJECT-LEVEL AND META-LEVEL KNOWLEDGE IN
EXPERT SYSTEMS(U) STAFORD UNIV CA DEPT OF COMPUTER
SCIENCE L FI NOV 85 STAN-CS-86-1091 N00039-83-C-0136

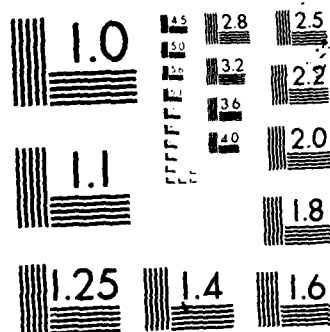
3/3

UNCLASSIFIED

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Charcot's triad:

- "If 1. serum bilirubin is elevated.
2. one of symptoms is shaking-chill.
3. one of symptoms is rt. upper colicky abd. pain.

then it is probable (.9) that the disease is
Calculous-Jaundice.

Courvoisier's law:

- "If 1. one of signs is palpable-gall-bladder.
2. Gall-bladder is not tender.

then it is probable (.8) that the disease is
Neoplasm.

Figure 7.1 Rediscovery of two well-known medical rules
by machine learning.

7.2. A Sample Dialogue of Interactive Mode in JAUNDICE

As described in Chapter 1 (also refer to figure 1.2), there are four main subprograms, each written in INTERLISP and running on a DEC 2060: the performance program, RL (the object-level rule learning program), the debugging program, and META-RULEGEN (the meta-rules learning program). Programs occupy a memory space of about 125 disk-pages⁴⁴; the knowledge base and the database occupy about 135 pages.

The performance program has two modes: interactive and batch modes. Interactive mode receives the user's input interactively, handles cases one at a time, and is designed to be a consultant. In contrast, batch mode handles multiple cases at one time; its purpose in

⁴⁴ A disk-page has 512 computer words.

JAUNDICE is to test the system performance by running multiple cases stored in the database under the following situations:

1. The knowledge base is edited. The modifications to the knowledge base should be tested by the old reference cases. Refer to Chapter 5.
2. The effect of meta-rules is to be tested. The meta-rules selected by utility value are further verified by running cases. Refer to Chapter 6.

Running interactive program for a consultation, exploiting all available facilities in the program (e.g., explanation, debugging the KB, etc.), takes about 10-20 minutes (clock time) under ordinary conditions⁴⁵, depending on the complexity of the case entered. The main portion of time is used for input/output and machine learning. If invoked, learning may become the bottleneck in such a program. Although learning and debugging may be carried out after the consultation, our ambitious goal is to develop a program which can learn fast enough to exchange ideas with the expert "on-line". In order to improve its practical value, CONDENSER (described in chapter 3) is invented; and it proves to be useful in conducting a consultation involved with learning and debugging at a reasonably good speed.

The following subsections show an annotated sample dialogue which is organized stage by stage. The demonstrated case is initially misdiagnosed (mismatch between the expert diagnosis and the system top diagnosis). The system finally manages to make a correct diagnosis after automatically debugging the knowledge base. This demonstration is not for showing how the user can communicate with the computer through manipulations of the built-in commands, but rather, it intends to show the ability of learning in such an

⁴⁵ The computer facility is ordinarily loaded.

expert system. Notice, however, the learning and debugging with respect to a certain case, as demonstrated here, can also be done non-interactively.

{In the following annotated sample dialogue, the user input appears *only* after the arrow head ">".}

7.2.1. Gathering Information

The program starts with collecting information by interacting with the user who enters an answer for each question given by the program. The answer can be of YES-NO type or numerical. The user can enter "?" if he doesn't know, or he may enter a real number from -1 to 1 to indicate the degree of certainty (confidence) for YES-NO type questions. Since every new case for consultation will be recorded in the database, the program intends to collect all possible information that it thinks is worthwhile. For example, if the patient for consultation has hepatomegaly, the program will continue to ask the degree of hepatomegaly. The JAUNDICE program is designed to scan jaundice patients preliminarily before invasive or advanced laboratory tests have been done. And the data entered are assumed to be the most significant data so far as the recent episode of jaundice is concerned.

1. What's the patient's name?
>FRED-SMITH

2. What's the sex?
>M

3. What's the age?
>65

4. What is the serum level of TOTAL-BILIRUBIN (mg/dl) ?
>3.2

5. What is the serum level of GOT (I.U.) ?
>210

6. What is the serum level of GPT (I.U.) ?
>280

7. What is the serum level of ALKALINE-PHOSPHATASE (B.U.) ?
>37

8. Is the COURSE of disease STATIONARY ?
>YES

9. Is the ONSET of disease ABRUPT ?
>NO

10. Is one of SYMPTOMS HEMATEMESIS ?
>NO

11. Is one of SIGNS SPLENOMEGALY ?
>NO

12. Is one of SIGNS ASCITES ?
>NO

13. Is serum ALBUMIN DECREASED ?
>NO

14. Is serum GLOBULIN ELEVATED ?
>NO

15. Is one of SYMPTOMS MALAISE ?
>NO

16. Is one of SYMPTOMS ANOREXIA ?
>NO

17. Is URINE-UROBILINOGEN ELEVATED ?
>NO

18. Is one of SIGNS HEMOGLOBINURIA ?
>NO

19. Is BILIRUBINURIA PRESENT ?
>YES

20. Is RETICULOCYTOSIS PRESENT ?
>NO

21. Is ANEMIA PRESENT ?
>NO

22. Is RATIO-OF-INDIRECT-OVER-DIRECT-BILIRUBIN >3 ?
>NO

23. Is one of SYMPTOMS INTENSE-PRURITUS ?
>NO
24. Is one of SYMPTOMS COLICKY-RT.-UPPER-ABDOMINAL-PAIN ?
>YES
25. Is one of SYMPTOMS FEVER ?
>NO
26. Is one of SYMPTOMS SHAKING-CHILL ?
>NO
27. Is one of SYMPTOMS CLAY-COLORED-STOOL ?
>NO
28. Is one of HISTORIES DAILY-FLUCTUATION-OF-JAUNDICE ?
>NO
29. Is the COURSE of disease RECURRENT ?
>NO
30. Is one of SIGNS PALPABLE-GALL-BLADDER ?
>YES
31. Is GALL-BLADDER TENDER ?
>NO
32. Is GALL-BLADDER NODULAR ?
>NO
33. Is one of SIGNS HEPATOMEGALY ?
>YES
34. What is the degree of HEPATOMEGALY ?
1). MILD 2). MODERATE 3). MARKED
>1
35. Is LIVER COARSELY-NODULAR ?
>NO
36. Is OCCULT-BLOOD of STOOL PRESENT ?
>NO
37. Is one of SIGNS LOWER-ABDOMINAL-MASS ?
>NO
38. Is one of SIGNS EPIGASTRIC-MASS ?
>NO

39. Is one of SYMPTOMS B.W.LOSS ?

>YES

40. What is the degree of B.W.LOSS ?

1). MILD 2). MODERATE 3). MARKED

>1

41. Is one of HISTORIES RECENT-TRAUMA ?

>NO

42. Is one of HISTORIES RECENT-SURGERY ?

>NO

43. Is serum ANTI-MITOCHONDRIAL-ANTIBODY ELEVATED ?

>NO

44. Is one of SYMPTOMS ABD.-PAIN-RADIATING-TO-RT.-SHOULDER ?

>NO

45. Is one of SYMPTOMS VOMITTING ?

>NO

46. Is the ONSET of disease SINCE-CHILDHOOD ?

>NO

47. Is one of HISTORIES RECENT-EXPOSURE-TO-HEPATOTOXIC-AGENT ?

>NO

48. Are there any associated diseases or complications?

>NO

7.2.2. Providing Interpretations

From the gathered information (that was also recorded into the database), the performance program draws inferences about the likely diseases and underlying mechanisms, based on the knowledge base. During processing, a dynamic database is constructed which records all deduced facts (intermediate and final conclusions), which are then printed out.

INTERPRETATION:

The Mechanism of Jaundice: Cholestasis, decreased bile flow due to
obstruction of biliary tract.

Pathological conditions: Cholestasis, Inflammation.

Anatomical diagnosis: Gall-bladder disease

Disease diagnosis:

CALCULOUS-JAUNDICE with degree of certainty .58
NEOPLASM with degree of certainty .41

{The performance program of JAUNDICE provides interpretation for the currently entered case, which includes: disease, mechanism of jaundice, pathological and anatomical conditions. Performer (the performance program) is goal-oriented type Reasoner. Concluding disease entity that causes jaundice is the main goal; concluding mechanism and pathological states of jaundice is the subgoal. The conclusions are made by invoking rules stored in the knowledge base. Because of the sophisticated nature of medicine, the interpretation will be more credible only if it incorporates the underlying pathological mechanisms besides the disease entity. In this scheme, the disease diagnosis will fall in a ten disease category. Diseases are generally assumed to be mutually exclusive. So, if one disease is definitely concluded, other diseases will be denied. However, there might be several diseases concluded if they all are not definitely concluded. In the final returned list of disease diagnoses, it only contains the diseases whose degree of certainty is greater than ".2". Though the general assumption is "mutually exclusive" among diseases, the implication of coexistence of diseases may be made if more than one disease reach the top degree of certainty simultaneously.}

7.2.3. Explanation

JAUNDICE only provides an overall explanation for the conclusions made. That is, the program summarizes all the knowledge used for making conclusions. This is quite similar to the way an expert analyzes a case. Because of explanation, the user gains a better understanding about the reasoning basis of the system. Therefore, we may view explanation as an error-checking mechanism with respect to both the system and the user.

Do you want an explanation for above diagnoses? (Please enter Y or N)
>Y

Bilirubinuria, elevated ALKALINE-PHOSPHATASE more than 15 B.U. and absence of URINE UROBILINOGEN ELEVATION are evidences suggesting the mechanism of jaundice is "Cholestasis". Elevation of both GOT and GPT indicates pathology of "Inflammation". The sign of PALPABLE-GALL-BLADDER and the symptom of COLICKY-RT.-UPPER-ABDOMINAL-PAIN indicate anatomy of disease is: "Gall-bladder". Thus, "CALCULOUS-JAUNDICE" is the most likely diagnosis. However, in this case, this diagnosis should be differentiated from "NEOPLASM". Though it can't be ruled out, the evidence tending to deny "NEOPLASM" includes the fact that the course is STATIONARY.

{Since JAUNDICE is a rule-based expert program, the provided explanation is based on the involved rules. If a fact is concluded, by tracing down the rules for conclusion, the program can collect all the relevant evidences that support this conclusion. Similarly, the program is able to gather disconfirming evidence if a fact is denied. A good explanation can be arrived at by this means if the knowledge base is constructed hierarchically. Because of the complexity of medicine, a logical explanation should be given level by level, starting from discussing pathological and anatomical mechanisms or states and ending up with concluding the disease entity. In order to augment the explanation capability, more knowledge about causal links and taxonomy is required, which is coded into 80 non-diagnostic rules in the knowledge base.}

7.2.4. Asking for the Expert's Diagnosis

The program may receive the feedback from the user if he is an expert. If the system conclusion matches the expert's, the consultation will move to the next case; otherwise the knowledge base will be debugged.

Are you satisfied with the result? (please enter Y or N)

>N

Why are you not satisfied?

- 1 Because the top diagnosis is not your diagnosis!
- 2 Because the top diagnosis is not certain enough as expected!

(please enter number.)

>1

What is your diagnosis?

- 1 Acute hepatocellular hepatitis
- 2 Chronic non-cirrhotic hepatitis
- 3 Hepatocellular cirrhosis
- 4 Primary biliary cirrhosis
- 5 Calculous jaundice
- 6 Neoplasm
- 7 Cholestatic hepatitis
- 8 Hemolysis
- 9 Congenital conjugation defect
- 10 Congenital excretion defect

(please enter number)

>6

{The program allows the user to enter his opinion about the diagnosis. Though not indicated here, only when the user is an expert, he is allowed to do so. The expert may hold a different view either because he thinks the diagnosis should be a different disease or because the top system diagnosis is not as certain as he expects. In the former case, the program will continue to ask the

expert diagnosis and then the learning program will be triggered to find missing rules or correct rules in the knowledge base so that the expert diagnosis will become the top diagnosis; in the latter case, the learning program will also be triggered in order to make the diagnosis more certain. In the example here, the expert diagnosis is NEOPLASM, and the top system diagnosis CALCULOUS-JAUNDICE. The objective of the program at this point is to make the diagnosis NEOPLASM become the top system diagnosis. For more detail, please refer to chapter 5.}

7.2.5. Debugging the Knowledge Base

In this example, since the expert disagrees with the system on the diagnosis, it is likely that something is wrong in the knowledge base. The program may first ask the expert to edit the knowledge base if he can. Suppose the expert doesn't know how to do it either because he is not familiar with the procedures or because he can only tell the diagnosis from the overall clinical picture but cannot abstract some medical rules from it, the program will take over the debugging task. The following computer output shows how the knowledge base is debugged automatically.

Would you like to create new rules or edit old rules?
>N

{If the answer is YES, a program like TEIRESIAS [Davis, 1979] or Emycin KB editor [Van melle, 1980] will step in, which allows the expert to enter new knowledge or edit old knowledge. However, in JAUNDICE, this part is not implemented mainly because it is not the focus of this thesis.}

Based on your diagnosis, let me figure out if there are missing rules or bugs in old rules.

<Please wait!>

I have scanned the conditions of rules for making conclusion, which include: R38, R45, R93, R113, R135, R153. Since they are sound and compatible with the case library, I suspect there might be some rules missing or some rules which should be corrected if your diagnosis is right. Now, I will first create new rules to support the diagnosis "NEOPLASM" and disfavor "CALCULOUS-JAUNDICE" on the basis of the data you entered; then I will inspect old rules to see where the bugs are.

<Please wait!>

The following are newly learned rules:

NR1: If
 1 COURSE of DISEASE is not RECURRENT
Then
 It is possible (.4) that DISEASE-ENTITY of PATIENT-JAUNDICE is not
CALCULOUS-JAUNDICE !

NR2: If
 1 ONE of SIGNS is PALPABLE-GALL-BLADDER
 2 GALL-BLADDER is not TENDER
Then
 It is probable (.8) that DISEASE-ENTITY of PATIENT-JAUNDICE is
NEOPLASM !

Now, I try to debug the old rules by comparing them with new rules created.

<Please wait!>

I found NR1 is redundant with R135, so NR1 is not necessary. From a definition rule R45, LHS of NR2 is "COURVOISIER-LAW", then NR2 becomes a generalization of R93. Conclusively, there are no missing rules; the only bug is: R93 should be generalized into NR2.

{At this point, since the expert doesn't know how to edit the knowledge base (though he knows what the correct diagnosis should be), the task is taken over by the so called automatic knowledge base debugger which essentially has two main components: the learner (the learning program) and the debugger (the debugging program). First, the program checks all invoked rules to make sure whether they are right. If those used rules are not right, then they are corrected and the case is re-run to see whether a correct diagnosis can be reached. In this example, however, the invoked rules are all right: thus the fault may be attributed to some missing rules or some rules that should be invoked but were not. If each rule in the knowledge base has been checked constantly with respect to the current case library, then it is not necessary to recheck them when a misdiagnosis occurs. In our scheme, we assume there is a database, from which a knowledge base is constructed: the knowledge base is then constantly updated on the basis of the continuously growing database: though the database may be pruned to a reasonable size without disturbing its statistics. Suppose the initial knowledge base is built by human experts, the knowledge base needs to be tested by the database, and somehow there should be compatibility between the knowledge base and database. Here, a strategy called "retrospective inspection after learning" is employed. Ignore temporarily the old rules in the knowledge base: all rules related to this case are found by learning. Then, by comparing old rules with new rules, bugs are found. In this example, R93 is found to be too specific and causes the misdiagnosis. For more detail, please see chapter 5.}

After knowledge base has been edited, interpretation becomes:

The Mechanism of Jaundice: Cholestasis, decreased bile flow due to
obstruction of biliary tract.

Pathological conditions: Cholestasis, Inflammation.

Anatomical diagnosis: Extrahepatic biliary obstruction

Disease diagnosis:

NEOPLASM with degree of certainty .6

CALCULOUS-JAUNDICE with degree of certainty .39

{After the knowledge base has been edited, the performance program re-runs the case and prints out the result which may give a feedback to the expert. If the expert diagnosis is correct and the knowledge base is defective, then a correct diagnosis can be obtained after this debugging. If the diagnosis is the same after changing the knowledge base as before or no bugs are found, it might imply that the expert diagnosis is wrong or the data are incomplete. Not indicated here is that the modified knowledge base should be tested against the old reference cases to see whether the old performances are still maintained.}

Do you think NR2 is a right rule?
>Y

{Even though the expert may not know how to edit the knowledge base, he might know whether the new rules are correct or not. To criticize is easier than to create.}

The knowledge base has been debugged, but I still have to confirm it by consulting my advisory experts.

{This message indicates that the changes of the knowledge base in an expert system should be very cautious!}

Do you have more cases for consultation?
>N

THANK-YOU!

7.3. Validation

The main focus of the JAUNDICE program is its learning ability. In this section, we only describe how to evaluate a learning system. (Refer to [Buchanan and Shortliffe 84] for the methods of evaluation of a performance program.)

7.3.1. Rediscovery of Well-Known Concepts

Rediscovery of some well-known concepts has been used by researchers as a way to validate the learning methods; for example, rediscovery of prime numbers by AM [Lenat 83], and rediscovery of Ohm's law by BACON [Langley 83]. Perhaps, this is one way to build confidence in machine learning. As seen in figure 7.1, our learning model is also capable of rediscovering some well-known medical rules for diagnosing jaundice.

7.3.2. Testing Generality in the Same Domain

In inductive concept learning, we desire to find concept descriptions (or rules) that are consistent not only with the given set of training instances but also with all instances in the given domain. Accordingly, the soundness of a learning method can be reflected from the generality of the result it produces. For this reason, we apply the learning result not only to the original training set but also to another set of test cases collected from the Stanford Medical Center. The result, as shown in table 7.2, indicates sufficient generality in the jaundice domain and this in turn implies the soundness of the learning method. Notice, however, if the training set is poor, the result will be poor, despite a perfect learning system.

7.3.3. Testing Generality in Other Domains

Domain-independence or generality has been emphasized in designing AI programs. The development of our learning model is partially in response to this consideration. Here, the generality of the developed RL program, which has been employed in JAUNDICE, is assessed by applying it to another domain named REFEREE.

REFEREE is an expert program written in EMYCIN for evaluating the quality of medical papers (refer to [Haggerty 84] for REFEREE). The conclusion is based on the parameters which deal with the reputation of author, journal, and institution, and the execution scheme, and the statistical analysis. For example, the parameter "Placebo-used" denotes a placebo was used in the experiment. Figure 7.2 shows an example of a rule in REFEREE.

```

Rule061: If 1. The quality of planning is unknown.
           2. A biostatistician was sufficiently involved.

           then it is possible (.3) that planning is good.

(RULE061 (($AND (SAME CNTXT PLANNING-UNKNOWN)
                (SAME CNTXT BIOSTAT)))
          (CONCLUDE CNTXT PLANNING-GOOD YES TALLY 300))

```

Figure 7.2 One example of a rule in REFEREE.

We first construct a set of training instances as follows. Each training instance is generated by a heuristics-based case generator and then concluded by the REFEREE program.⁴⁶ the heuristics used are based on the KB in REFEREE; e.g., the relative weighting for each parameter in making conclusions. Thus, each training instance has data descriptions and a correct classification. Since we believe the cross-prediction experiment (i.e., the result learned from one case library is applied to another) is the best demonstration of the validity of a learning method in a given domain, as illustrated by JAUNDICE, we apply this strategy to testing the generality in REFEREE. Sixty cases are

⁴⁶We didn't collect the real cases because the team working on this system has only a limited number of cases in storage; and another difficulty is that the parameters used by the program, such as the reputation of the authors, are often hard to obtain by the people outside the field.

generated and then divided into two parts; the rules learned from one part are tested by another, and vice versa. The results, as shown in table 7.3, support our assertion of domain-independence: i.e., we have demonstrated the value of the developed learning model in a medical domain (JAUNDICE) and a non-medical domain (REFEREE).

Table 7.3 Experiments of learning in REFEREE expert system. Rules learned from one case library are tested by itself and another case library, and vice versa. The diagnostic accuracy of every test is shown here.

	Library A (30 cases)	Library B (30 cases)
New rules (A)	96.7%	93.3%
New rules (B)	80.0%	90.0%

New rules (A): Rules learned from library A.

New rules (B): Rules learned from library B.

7.4.1. Basic Assumptions

The assumptions are the following: most of the training instances should be correct and the descriptive language should be adequate (i.e., without causing inconsistency) to describe the concepts to be learned. Although we use optimization techniques described in Chapter 4 to make the learned rules consistent with most of the training instances, unless most of them are correct, no good results will yield. Inadequate descriptive language will cause inconsistent learning problems (i.e., no perfect solutions) [Mitchell 78]. Some work has begun to explore the problem of creating new language, e.g., [Utgoff 82]; however, this is still a difficult issue.

7.4.2. Requirement of Domain-dependent Knowledge and Heuristics

The success of Meta-DENDRAL [Buchanan 78a] is a demonstration that the initial domain-dependent knowledge (coded into a "half order theory") is crucial for learning. However, our experiment proves that it is possible to build a new KB with adequate performance without initial domain-dependent knowledge.⁴⁷ This seeming improvement, perhaps, can be explicated by the relative adequacy of the selected training instances and tractableness in the domain we choose. Even so, the success of our learning model still relies on some domain-dependent heuristics; i.e., we define minimal generality and specificity. Then, what's the difference between the initial knowledge (or half order theory) and heuristics? In fact, they both represent "constraints". In our view, learning is a heuristic search, the more constraints, the narrower the search space; if the constraints are accurate, the learning is made efficient; otherwise the result may be distorted or

⁴⁷ However, correct classification of cases by outside experts requires a lot of domain knowledge; Meta-DENDRAL did not require this.

incomplete. Accordingly, we often choose the constraints that represent high level abstraction and are well-established and thus bear broader and more meaningful implications. The program CONDENSER (see Chapter 3) also demonstrates that machine can generate a proper bias and displace in part the requirement of the initial domain-dependent knowledge.

7.4.3. Case Selection

In learning from examples, the importance of this issue is never overemphasized. Near-miss negative instances [Winston 70] are important to discover the discriminant features and are also important in the theory of condensation (described in Chapter 3). Careful selection of instances can expedite the convergence of a concept [Mitchell 78]. It is desirable that machine can generate any instance that it thinks can help learning; the restrictions often come from the want of a technique for verifying the generated instances.

7.4.4. Domain-Dependent Rules of Generalization or Specialization

The rules of generalization used in the learner should be tailored to domain characteristics. Were it not for those domain-specific rules of generalization, the results of learning may be of poor quality. For example, the "take minimum" rule is designed for medical domains (see Section 2.1.1); so, a common generalization of "(GOT 200)" and "(GOT 400)" is "(GOT ≥ 200)". On the contrary, if we use the "changing constant to variable" rule (not used in JAUNDICE) for the above example, the result is "(GOT x)", which means "GOT can be any value" and is, in fact, less medically meaningful.

7.4.5. Representational Adequacy

Here, instead of discussing general representational schemes, we focus on how to represent appropriately a given fact within a given representational framework. A good representation denotes a representation that is "simple" but "meaningful". The CONDENSER program (described in Chapter 3) is an example of dynamically representing the instances during learning in order to achieve operational efficiency. Still, the representation of feature values is important. For example, we use values: "mild", "moderate", and "marked" for a clinical feature "HEPATOMEGALY", instead of the real measurement of the liver size. Here, we illustrate how an improper representation can hurt learning. Consider two different representations for two positive instances: "pos1" and "pos2", and one negative instance: "neg1", with only one clinical parameter "GOT" as follows:

Representation I:

```
pos1: (GOT 200)
pos2: (GOT 400)
neg1: (GOT 100)
```

Representation II:

```
pos1: (GOT mildly-elevated)
pos2: (GOT moderately-elevated)
neg1: (GOT mildly-elevated)
```

It is straightforward that representation II causes inconsistency while with representation I. we can find a description "(GOT ≥ 200)" which is consistent with the three instances.

7.4.6. Rule Redundancy

Since, in EMYCIN-based systems, evidence can be combined, redundancy will cause the same piece of evidence to be reconsidered more than once and make the conclusion imprecise. Too many redundant rules may also jeopardize the efficiency. Syntactic redundancy is easier to detect than semantic redundancy, by which we mean two different rules look different but imply each other. We may further make a distinction between the following conditions: redundancy between rules and redundancy between rule sets; by the latter, we mean two sets of rules make the same prediction for every case. The RL program is able to remove syntactical redundancy; however, it is often hard to determine semantical redundancy particularly between two sets of rules except definitional redundancy. In our experiment, we found the influence of semantical redundancy is negligible; moreover this redundancy is useful when data are incomplete and only one of the redundant rules is fired [Buchanan and Shortliffe 84].

7.5. Comparison with Related Work

Among the related work,⁴⁸ Meta-DENDRAL [Buchanan 78a] may be the most closely related work since it also employs a heuristic search from the most general hypothesis and can discover multiple disjunctive rules. The sophistication of Meta-DENDRAL may be due to its task-oriented approach; it also implies some critical issues in learning, such as efficiency and noisy data. Meta-DENDRAL, successful as it is, however, lacks the ability

⁴⁸In the aspect of acquiring new knowledge by machine learning, particularly learning from examples, related works include: [Hayes-roth 76], [Hayes-roth 78], [Hunt 66], [Hunt 75], [Smith 77], [Larson 77], [Michalski 77], [Michalski 78], [Michalski 83a], [Quinlan 79], [Quinlan 83], [Samuel 67], [Vere 75], [Pawlak 81], [Holland 80], [Anderberg 73], [Buchanan 78a]. See [Cohen and Feigenbaum 82] and [Buchanan 78b] for more discussions of these systems and other references.

of incremental learning. As the evolution based on the previous works on inductive concept learning, the learning model developed in this thesis is intended to theorize on several issues, some of which may have already been noticed in other works, and provide unified solutions for them: for example, feature condensation to improve efficiency, optimization to handle noisy data, and incremental updating the knowledge base to make it complete. Indeed, all these considerations contribute to the significance behind this model. Here, we neglect all detailed comparisons, which can be referred to in the previous chapters.

7.6. Future Extensions

We propose four possible future extensions as the long term goals.

7.6.1. An Expert System with the Ability of Discussion

The current implementation of JAUNDICE can receive simple feedback from the user but it lacks the ability of engaging in a sophisticated discussion⁴⁹ with the user. The requirements (or difficulties) of developing such an intelligent system that can discuss with the user and even make comment on his thought include as follows:

1. Understanding natural language.
2. A huge knowledge base that comprises common sense and domain specific knowledge.

The system can understand the user's thought well only if the system can accept natural

⁴⁹ Here, we mean the discussion involves active learning rather than learning by being told. That is, the machine can learn or discover something instantly by taking the feedback from the human. This ability is lacking in some programs, such as [Clancey 79], which also have the ability of discussion.

language as a way of communication. And only if the system has a rather complete knowledge background, the system is capable of discussing and commenting.

A part of the man-machine dialogue may look like the following:

(H: human, M: machine)

M: My diagnosis is X, what is yours?

H: But my diagnosis is Y! Could you give me your explanation?

M: My explanation is, how about yours?

H: My explanation is

M: I think something is not right in your explanation,
which is
Would you like to reconsider your diagnosis?

H:

7.6.2. Unsupervised Learning

In our scheme, we assume there is a teacher to classify the training instances correctly, and the learning is based on this classification. The task will become more difficult if there is no teacher; this is called "unsupervised learning". Under this situation, the system has to discover the classification by some observation and experiment and select instances by itself. Some work has begun to explore this issue, e.g., [Buchanan 78a], [Lenat 83] and [Michalski 83b].

7.6.3. Training Instances with Multiple Classifications

In our scheme, we generally assume the classifications are mutually exclusive and we select those cases with a single classification as the training instances. The task will be harder if one case may have more than one classification, which may be due to the fact that the expert cannot further distinguish or, in medicine, it may be due to the coexistence of more than one disease.

7.7. Conclusion

The theories and methods developed in this thesis can serve as a framework for inductive concept learning. The model can not only become a general knowledge acquisition tool, because of our consideration of generality, efficiency, noise tolerance, and incremental learning ability, but can also lend itself to constructing a high performance expert system which can continuously grow and adjust its own knowledge base.

Appendix A

Degree of Certainty

Approaches to inexact reasoning (reasoning under uncertainty) include probabilistic methods (e.g., Bayesian statistical approach used in [Warner 64]), fuzzy set theory [Zadeh 65], CF model [Shortliffe 76], Dempster-Shafer theory ([Shafer 76] and [Barnett 81]). Most of them have been applied to medical decision making. Another approach introduced here, called "degree of certainty", stems from CF model used in MYCIN [Shortliffe 76].

Degree of certainty is used to represent uncertainty in this thesis. It is a real number, ranging from -1 to 1. Each statement is assigned a degree of certainty: "1" means "definitely yes"; "-1" means "definitely no"; "0" means "not knowing at all"; any number between 0 and 1 denotes that we tend to believe it and the number represents the estimated chance of "yes"; and any number between -1 and 0 denotes that we tend not to believe it and the number represents the estimated chance of "no". For example, the statement: "if the dark cloud appears, it rains with degree of certainty .5", means "if the dark cloud appears, the estimated chance of raining is 50% and we tend to believe it will rain".

In an expert system, the assignment of a degree of certainty to a rule in the knowledge base is based on the expert estimate, which is the integration of many factors, including

probabilistic knowledge and attitude (bias). Mathematically, degree of certainty is defined as follows:

$$\text{degree of certainty (h, e)} = P(h/e)S(h, e)$$

where,

h: hypothesis

e: evidence or pattern

P(h/e): probability of h, given e

S(h, e): strength of e for h

S(h, e) is a strength factor and may be viewed as a weighting factor assigned to an evidence "e" for a given hypothesis "h".

An evidence is in fact a pattern composed of one or more than one feature (or attribute). We assign a predictive value, ranging from 0 to 6 and reflecting statistical knowledge, to each feature for a given hypothesis. Then the predictive value of an evidence (or a pattern) is defined as the sum of the predictive values of all features in that evidence (or pattern).⁵⁰ Then, the "predictive value(h, e)" is mapped to the strength factor "S(h, e)" in an ad hoc fashion as follows:

For a confirming evidence,

Mapping:	<u>Predictive value(h, e)</u>	<u>S(h, e)</u>
	<2	.4
	[2 3)	.6
	[3 4)	.8
	[4 5)	.9
	[5 6)	.95
	≥6	1

⁵⁰The assignment of a predictive value for a pattern, based on the predictive values of individual features, is similar to the decision making in Rheumatology where combinations of different number of major and minor criteria (each criterion is a symptom or sign or laboratory test) may yield different conclusions. For instance, rheumatic fever can be diagnosed with a pattern of two major criteria or a pattern of one major plus two minor criteria.

There is another (ad hoc) mapping for disconfirming evidence. Assume the disconfirming rules are formed from high frequency evidence (refer to Section 2.4); the mapping is as follows:

<u>P(e/h)</u>	<u>degree of certainty(-h, -e)</u>
1	1
[.8 1)	.5
[.6 .8)	.3
<.6	0

Degree of certainty of different pieces of evidence can be combined according to CF combining function in MYCIN [Buchanan and Shortliffe 84] or Dempster-Shafer theory as in [Gordon 84]. Moreover, degree of certainty and CF are related in the following ways. For a confirming evidence "e", if $P(h) \neq 0$ and we assign $S(h, e) = 1$, then:

$$P(h/e) = CF(h, e) = \text{degree of certainty}(h, e)$$

If $P(h) \neq 0$, by properly choosing $S(h, e)$, degree of certainty can still approximate CF. The main feature of "degree of certainty" is the incorporation of a strength factor which can reflect expert attitude toward evidence (e.g., conservative or aggressive) in a specific domain.

Practically, using CF will face one problem, which is as follows: for a confirming evidence "e", because there often exist many hypotheses, the prior probability for each hypothesis $P(h)$ is close to zero, and the CF becomes simply conditional probability $P(h/e)$ as follows:

$$CF(h, e) = (P(h/e) - P(h)) / (1 - P(h)) \approx P(h/e)$$

Therefore, to overcome this problem, we define "degree of certainty" by separating out the strength factor which should be considered in AI.

References

- [Aiello 83] Aiello, N.
 A comparative study of control strategies for expert systems: AGE
 implementation of three variations of PUFF.
 In *Proceedings of the third National Conference on Artificial Intelligence*.
 Washington, D.C., 1983.
- [Anderberg 73] Anderberg, M. R.
Cluster Analysis for Applications.
 Academic Press, New York, 1973.
- [Balakrishnan 84] Balakrishnan, A. V.
Kalman Filtering Theory.
 Optimization Software, Inc., New York, 1984.
- [Barnett 81] Barnett, J. A.
 Computational methods for a mathematical theory of evidence.
 In *Proceedings of 7th International Joint Conference on Artificial
 Intelligence*. Vancouver, 1981.
- [Becker 78] Becker, P. W.
*Recognition of Patterns Using the Frequencies of Occurrence of Binary
 Words*.
 Springer-Verlag, New York, 1978.
- [Blasbalg 62] Blasbalg, H. and Blerkom, R. V.
 Message compression.
 In Davisson, L. D. and Gray, R. M. (editor), *Data Compression*.
 Dowden, Hutchinson & Ross, Inc., New York, 1962.
- [Blum 82] Blum, R. L.
*Discovery and representation of causal relationships from a large time-
 oriented clinical database: the RX project*.
 PhD thesis, Stanford University, 1982.

- [Brillouin 62] Brillouin, L.
Science and Information Theory.
 Academic Press Inc., New York, NY, 1962.
- [Buchanan 69] Buchanan, B. G. and Sutherland, G. L. and Feigenbaum, E. A.
 Heuristic DENDRAL: A program for generating explanatory
 hypotheses in organic chemistry.
 In B. Meltzer and D. Michie (editor), *Machine intelligence. vol. 4.*,
 Edinburgh University Press, Edinburgh, 1969.
- [Buchanan 78a] Buchanan, B. G. et al.
 Model-directed learning of production rules.
 In Waterman, D. and Hayes-Roth, F. (editor), *Pattern-Directed
 Inference systems.* Academic Press, New York, 1978.
- [Buchanan 78b] Buchanan, B. G., Mitchell, T. M., Smith, R. G., Johnson, C. R. Jr.
 Models of Learning Systems.
 In Belzer, J., Holzman, A. G., and Kent, A. (editor), *Encyclopedia of
 Computer Science and Technology.* M. Dekker, New York, 1978.
- [Buchanan and Shortliffe 84]
 Buchanan, B. G. and Shortliffe, E. H.
Rule-Based Expert Systems
 Addison-Wesley Publishing Company, Inc., Reading, Massachusetts,
 1984.
- [Carbonell 83] Carbonell, J. G., Michalski, R. S., Mitchell, T. M.
 An Overview of Machine Learning.
 In *Machine Learning*, chapter 1, . Tioga, Palo Alto, CA, 1983.
- [Clancey 79] Clancey, W. J.
Transfer of rule-based expertise through a tutorial dialogue.
 PhD thesis, Computer Science Department, Stanford University, 1979.
- [Clancey 83] Clancey, W. J.
 The epistemology of a rule-based expert system: A framework for
 explanation.
Artificial Intelligence 20, 1983.

[Clancey and Letsinger 81]

Clancey, W. J., and Letsinger, R.

NEOMYCIN: Reconfiguring a rule-based expert system for application to teaching.

In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*. Vancouver, 1981.

[Cohen and Feigenbaum 82]

Cohen, P. R. and Feigenbaum, E. A. (editor).

The Handbook of Artificial Intelligence, Volume III.

Kaufmann, Los Altos, CA, 1982.

[Croxtton, Cowden, and Klein 67]

Croxtton, F. E., Cowden, D. J., and Klein, S.

Applied General Statistics

Prentice-Hall Inc., Englewood Cliffs, NJ, 1967.

[Davis 76]

Davis, R.

Applications of meta-level knowledge to the construction, maintenance, and use of large knowledge bases.

PhD thesis, Computer Science Department, Stanford University, June, 1976.

[Davis 79]

Davis, R.

Interactive transfer of expertise: Acquisition of new inference rules.

Artificial Intelligence 12, 1979.

[Davis 80]

Davis, R.

Meta-rules: Reasoning about control.

Artificial Intelligence 15, 1980.

[Davis and Buchanan 77]

Davis, R., and Buchanan, B. G.

Meta-level knowledge: Overview and applications.

In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*. Cambridge, Mass., August, 1977.

- [Dempster 83] Dempster, A. P. and Rubin, D. B.
Introduction.
In Madow, W. G., Olkin, I., and Rubin, D. B. (editor), *Incomplete Data in Sample Surveys, Volume 2*, chapter 1. . Academic Press Inc., New York, 1983.
- [Dietterich 81] Dietterich, T. G. and Michalski, R. S.
Inductive learning of structural descriptions.
Artificial Intelligence 16, 1981.
- [Dietterich 83] Dietterich, T. G., Michalski, R. S.
A Comparative Review of Selected Methods for Learning from Examples.
In *Machine learning*, chapter 3, . Tioga Publishing Company, Palo Alto, CA, 1983.
- [Dietterich and Buchanan 81]
Dietterich, T. G. and Buchanan, B. G.
The Role of Critic in Learning Systems.
Technical Report STAN-CS-81-891, Computer science department, Stanford University, 1981.
- [Duda 78] Duda, R. O. et al.
Development of the PROSPECTOR consultant system for mineral exploration.
Technical Report, Artificial Intelligence, SRI International, 1978.
- [Fu 84] Fu, Li-Min and Buchanan, B. G.
Enhancing Performance of Expert Systems by Automated Discovery of Meta-rules.
In *Proceedings of the 1st conference on Artificial Intelligence Applications*. IEEE, Denver, Colorado, December, 1984.
- [Gordon 84] Gordon, J. and Shortliffe, E. H.
The Dempster-Shafer Theory of Evidence.
In Buchanan, B. G., Shortliffe, E. H. (editor), *Rule-Based Expert Systems*, chapter 13. . Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1984.

- [Gray 74] Gray, R. M. and Davisson, L. D.
A mathematical theory of data compression?
In Davisson, L. D. and Gray, R. M. (editor), *Data Compression*, .
Dowden, Hutchinson & Ross, Inc., New York, 1974.
- [Haggerty 84] Haggerty, J.
*Referee and Rulecritic: Two Prototypes for Assessing the Quality of
Medical Paper*.
Technical Report, Computer Science Department, Stanford University,
Master Thesis, 1984.
- [Hahn 29] Hahn, H.
Empiricism, Mathematics, and Logic.
In McGuinness, B. (editor), *Hans Hahn: Empiricism, Logic, and
Mathematics*, . D. Reidel Publishing Company, Boston,
Massachusetts, 1929.
- [Hahn 30] Hahn, H.
The Significance of the Scientific World View, Especially for
Mathematics and Physics.
In McGuinness, B. (editor), *Hans Hahn: Empiricism, Logic, and
Mathematics*, . D. Reidel Publishing Company, Boston,
Massachusetts, 1930.
- [Hayes-roth 76] Hayes-Roth, F.
Patterns of Induction and Associated Knowledge Acquisition
Algorithms.
In Chen, C (editor), *Pattern Recognition and Artificial Intelligence*, .
Academic Press, New York, 1976.
- [Hayes-roth 78] Hayes-roth, F. et al.
Machine Methods for Acquiring, Learning, and Applying Knowledge.
Technical Report R-6241, The RAND Corporation, 1978.
- [Hayes-Roth 83] Hayes-Roth, F., Waterman, D., and Lenat, D.
An Overview of Expert Systems.
In *Building Expert Systems*, chapter 1, . Addison-Wesley Publishing
Company, Inc., Reading, Massachusetts, 1983.

- [Holland 80] Holland, J. H.
Adaptive Algorithms for Discovering and Using General Patterns in
Growing Knowledge bases.
Policy Analysis and Information Systems 4. 1980.
- [Hunt 66] Hunt, E. B.
Experiments in Induction.
Academic Press, New York. 1966.
- [Hunt 75] Hunt, E. B.
Artificial Intelligence.
Academic Press, New York. 1975.
- [Krupp 82] Krupp, M. A. et al.
Current Medical Diagnosis and Treatment.
Lange Medical Publications, Los Altos, CA. 1982.
- [Kulikowski 82] Kulikowski, C. and Weiss, S.
Representation of expert knowledge for consultation: The CASNET and
EXPERT projects.
In P. Szolovits (editor), *Artificial Intelligence in Medicine*, . Westview
Press, Boulder, Colorado. 1982.
- [Langley 83] Langley, P., Bradshaw, G. L., and Simon, H. A.
Rediscovering chemistry with the BACON system.
In *Machine learning*, chapter 10. . Tioga Publishing Company, Palo
Alto, CA. 1983.
- [Larson 77] Larson, J.
*Inductive Inference in the Variable-Valued Predicate Logic System VL21:
Methodology and Computer Implementation*.
PhD thesis. University of Illinois. May, 1977.
- [Lenat 83] Lenat, D. B.
Theory formation by heuristic search. The nature of heuristics II:
Background and examples.
Artificial Intelligence 21. 1983.

- [Madow, Nisselson, and Olkin 83]
 Madow, W. G., Nisselson, H., and Olkin, I.
 Review of Theory.
 In Madow, W. G., Nisselson, H., and Olkin, I. (editor), *Incomplete Data in Sample Surveys, Volume I*, . Academic Press Inc., New York, 1983.
- [Malchow-moller 81]
 Malchow-Moller, A. et al.
 Causes and Characteristics of 500 Consecutive cases of Jaundice.
Scandinavia Journal of Gastroenterology 16, 1981.
- [Martin 71] Martin, W. A. and Fateman, R. J.
 The MACSYMA system.
 In *Proceedings of the 2nd Symposium on Symbolic and Algebraic Manipulation*. 1971.
- [Michalski 75] Michalski, R. S.
 Variable-Valued Logic and its Applications to Pattern Recognition and Machine Learning.
 In Rine, D. C. (editor), *Computer Science and Multiple-Valued Logic Theory and Applications*, . North-Holland, 1975.
- [Michalski 77] Michalski, R. S.
 A system of programs for computer-aided induction.
 In *Proceedings of 5th International Joint Conference on Artificial Intelligence*. Cambridge, Mass., 1977.
- [Michalski 78] Michalski, R. S. and Larson, J. B.
Selection of most representative training examples and incremental generation of VLI hypotheses: The underlying methodology and description of programs ESEL and AQ11.
 Technical Report 867, University of Illinois, 1978.
- [Michalski 83a] Michalski, R. S.
 Theory and Methodology of Inductive Learning.
 In *Machine Learning*, chapter 4, . Tioga, Palo Alto, CA, 1983.

- [Michalski 83b] Michalski, R. S. and Stepp, R. E.
Learning from observations: Conceptual clustering.
In *Machine learning*, chapter 11, . Tioga Publishing Company, Palo Alto, CA, 1983.

- [Michie 84] Michie, D., Muggleton, S., Riese, C., Zubrick, S.
RULEMASTER: A Second-Generation Knowledge-Engineering Facility.
In *Proceedings of the 1st conference on Artificial Intelligence Applications*. IEEE, Denver, Colorado, December, 1984.

- [Miller, Pople, and Meyers 82] Miller, R. A., Pople, H.E., Meyers, J. D.
INTERNIST-1, an experimental computer-based diagnostic consultant for general internal medicine.
The New England Journal of Medicine 307, 1982.

- [Minsky 63] Minsky, M.
Steps toward Artificial Intelligence.
In Feigenbaum, E. A. and Feldman, J. (editor). *Computers and Thought*. McGraw-Hill, New York, 1963.

- [Mitchell 78] Mitchell, T. M.
Version Spaces: An approach to concept learning.
PhD thesis, Stanford University, December, 1978.

- [Papoulis 65] Papoulis, A.
Probability, Random Variable, and Stochastic Processes. McGraw-Hill, New York, 1965, chapter 11.

- [Patil 82] Patil, R. S., Szolovits, P., and Schwartz, W. B.
Information acquisition in diagnosis.
In *Proceedings of the National Conference on Artificial Intelligence*. Pittsburgh, PA. 1982.

- [Pawlak 81] Pawlak, Z.
Classification of objects by means of attributes.
Technical Report 423, Institute of Computer Science, Polish Academy of Science, 1981.

- [Petersdorf 83] Petersdorf, R. G. et al.
Harrison's Principles of Internal Medicine.
McGraw-Hill, New York, 1983.
- [Politakis 82] Politakis, Peter G.
Using empirical analysis to refine expert system knowledge base.
PhD thesis, LCSR, Rutgers University, October, 1982.
- [Quinlan 79] Quinlan, J. R.
Induction over large data bases.
Technical Report HPP-79-14, HPP, Stanford University, 1979.
- [Quinlan 83] Quinlan, J. R.
Learning efficient classification procedures and their applications to
chess end-games.
In Michalski, R. S. et al (editor), *Machine Learning*, chapter 15, . Tioga,
Palo Alto, CA, 1983.
- [Samuel 67] Samuel, A. L.
Some studies in machine learning using the game of checkers II - Recent
progress.
IBM Journal of Research and Development 11, 1967.
- [Schiff 46] Schiff, L.
The Differential Diagnosis of JAUNDICE.
The Year Book publishers, Inc., Chicago, 1946.
- [Shafer 76] Shafer, G.
A Mathematical Theory of Evidence.
Princeton University Press, Princeton, NJ, 1976.
- [Shortliffe 76] Shortliffe E.H.
Computer-based medical consultations: MYCIN.
American Elsevier, New York, 1976.
- [Simon 83] Simon, H. A.
Why Should Machine Learn?
In *Machine learning*, chapter 2, . Tioga Publishing Company, Palo Alto,
CA, 1983.

- [Sirken 83] Sirken, M. G.
Handling Missing Data by Network Sampling.
In Madow, W. G., Olkin, I., and Rubin, D. B. (editor). *Incomplete Data in Sample Surveys, Volume 2*, chapter 8. . Academic Press Inc., New York, 1983.
- [Smith 77] Smith, R. G., Mitchell, T. M., Chestek, R. A., and Buchanan, B. G.
A model for learning systems.
In *Proceedings of 5th International Joint Conference on Artificial Intelligence*. Cambridge, Mass., 1977.
- [Stern 75] Stern, S. et al.
Use of Computer Program for Diagnosing Jaundice in District Hospitals and Specialized Liver Unit.
British Medical Journal 2, 1975.
- [Suwa 84] Suwa, M., Scott, A. C., and Shortliffe, E. H.
Completeness and Consistency in a Rule-Based Expert System.
In Buchanan, B. G. and Shortliffe, E. H. (editor). *Rule-Based Expert Systems*, chapter 8. . Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1984.
- [Utgoff 82] Utgoff, P. E.
Acquisition of appropriate bias for inductive concept learning.
Technical Report, Thesis proposal, Department of Computer Science, Rutgers University, 1982.
- [Van Melle 80] van Melle, W.
System aids in constructing consultation programs.
UMI Research Press, Ann Arbor, MI, 1980.
- [Vere 75] Vere, S. A.
Induction of concepts in the predicate calculus.
In *Proceedings of 4th International Joint Conference on Artificial Intelligence*. Tbilisi, USSR, 1975.

- [Wallis 82] Wallis, J. W. and Shortliffe, E. H.
Explanation power for medical expert system: studies in representation
of causal relationships for clinical consultations.
Methods Info. Med. 21, 1982.
- [Warner 64] Warner, H. R. et al.
Experience with Bayes' Theorem for computer diagnosis of congenital
heart disease.
Ann. N. Y. Acad. Sci. 115, 1964.
- [Waterman 68] Waterman, D.
Machine Learning of Heuristics.
PhD thesis, Stanford University, 1968.
also report CS118, AI 74.
- [Winkelman 81] Winkelman, E. I.
The Differential Diagnosis of Jaundice.
Primary Care 8, 1981.
- [Winston 70] Winston, P. H.
Learning structural descriptions from examples.
Technical Report TR-76, Project MAC, MIT, 1970.
- [Zadeh 65] Zadeh, L. A.
Fuzzy sets.
Information and Control 8, 1965.

END

11-86

OTIC