(2)

AD-A220 945

DTIC
ELECTE
APR 25 1990
S D

RUTGERS UNIVERSITY
Center for Expert Systems Research

Final Report:
Empirical Analysis and Refinement of Expert System Knowledge Bases
Contract Number
N00014-87-K-0398
Office of Naval Research

March 31, 1990

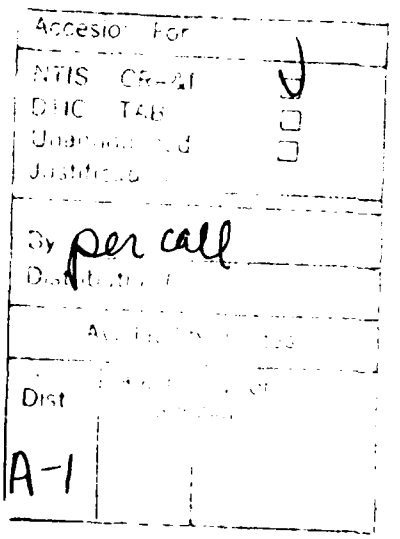
Principal Investigators:
Sholom M. Weiss
Casimir A. Kulikowski

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

Table of Contents

STATEMENT "A" per Alan Meyrowitz
ONR/Code 1133
TELECON
4/25/90

VG



1. Technical Project Summary

Knowledge base refinement is the modification of an existing expert system knowledge base with the goals of localizing specific weaknesses in a knowledge base and improving an expert system's performance. Systems that automate some aspects of knowledge base refinement can have a significant impact on the related problems of knowledge base acquisition, maintenance, verification, and learning from experience. The SEEK system was the first expert system framework to integrate large-scale performance information into all phases of knowledge base development and to provide automatic information about rule refinement. A recently developed successor system, SEEK2 [Ginsberg, Weiss, and Politakis 88] significantly expands the scope of the original system in terms of generality and automated capabilities. The investigators made significant progress in automating empirical expert system techniques for knowledge acquisition, knowledge base refinement, maintenance, and verification.

2. Principal Innovations

The investigators demonstrated a rule refinement system in an application of the diagnosis of complex equipment failure: computer network troubleshooting. The expert system demonstrates the following advanced capabilities:

- automatic localization of knowledge base weaknesses
- automatic repair (refinement) of poorly performing rules
- automatic verification of new knowledge base rules
- automatic learning capabilities

3. Summary of Progress

During the two years of the contract the following was accomplished:

- functioning equipment diagnosis and repair knowledge base, suitable for refinement. This is a subset of DEC's Network Troubleshooting Consultant (NTC).
- demonstration of functioning equipment diagnostic system with capabilities of localization of weak rules, automatic refinement, automatic verification.
- demonstration of initial rule learning capabilities.
- development of case generation simulator and randomized rule modifier.
- comparative studies demonstrating superiority of PVM rule induction procedure in low dimensional applications.

- demonstration of refinement system, using subset of DEC's Network Troubleshooting Consultant (NTC), a rule-based expert system that gives interpretive analysis of Ethernet/DECnet related problems. The system automatically recovers from many forms of damage to knowledge base.

- demonstration of system with capabilities for automatic refinement, and verification of knowledge base consistency.

- demonstration of significant automated rule learning capabilities.

- completed comparative studies of empirical techniques for machine learning, statistical pattern recognition, and neural nets.

This work is the basis for further progress in developing an automated refinement system. We pursued the refinement and learning tasks from both an expert system rule-based perspective and a machine learning rule induction perspective. In order to develop the strongest form of refinement system, we examined numerous techniques for empirical rule induction. We also developed a procedure, Predictive Value Maximization [Weiss, Galen, and Tadepalli 90], that shows strong results for induction of single relatively short rules. Our fundamental objective is to mix the best rule induction procedures with a rule-based expert system to achieve the strongest empirical results.

The fundamental approach of rule refinement is to constrain changes that can be made to the knowledge base to those that are fully consistent with the rules of the expert-supplied knowledge base. Unlike a refinement system, a pure learning system such as a rule induction system, attempts to learn directly from data, unconstrained by human expert knowledge. A more constrained learning approach maintains the expert supplied rules but allows for some additions to the rules. The new learning procedures added to the refinement system use generalization and specialization models to perform 2 functions:

- add a variable to a rule to specialize the rule

- add a new rule to the knowledge base to generalize the rule

The procedure for adding components and rules is detailed in Section 3.1. Some key parts of the procedure are analogous to current tree generation procedures such as ID3/C4 or CART, where the split is performed on the single best node. In our case during a given refinement cycle, we attempt to induce the single best variable and decision threshold. The following preliminary results were found for a knowledge base of 100 rules and 5 endpoints that previously was refined from a performance of 73% (88/121) to 100% (121/121).

- The same 100% refinement performance was achieved with the learning capability.

- When all 100 rules, with an average of 4 variables per rule, were deleted from the knowledge base, the system was able to generate 14 rules and 21 variables that achieved 88% (107/121) correct classification.

As a pure learning procedure, these techniques are somewhat weaker than induced decision trees. The heuristic refinement strategy of generalization and refinement does not appear to perform as well when train and test simulations are used to estimate the true error rate. However, this refinement strategy is not meant to be a learning strategy that applied only to sample data. It can readily work on an existing knowledge base and produces a new knowledge base that is consistent with the original expert derived knowledge base. These results demonstrate the potential for robust mixed knowledge base refinement and learning procedures.

Additional results for learning with the Network Troubleshooting Consultant are listed in table 3-1. In these simulations, the knowledge base was perturbed, and then the refinement system attempted to *fix* the knowledge base. Each *bash* is one random modification to a rule attribute in the knowledge base. Table 3-1 lists the number of random changes made to rules in the knowledge base, the subsequent performance of the rule-bases system using these bashed rules as measured in correct cases, the number of refinements the leaning system makes to the knowledge base, and the subsequent performance after refinement. There are 74 stored cases.

no. of bashes	correct cases	num. of refinements	refined correct
1	74	-	-
2	74	-	-
4	74	-	-
8	72	1	74
16	69	4	74
32	66	3	74
64	66	2	74
128	57	5	72
256	47	6	72

Figure 3-1: Refinement of Randomly Perturbed Knowledge Base

In addition to the learning techniques, a limited language was developed for constraining the refinement process based on domain specific characteristics. The following constraints were implemented and tested:

- Disallow modifications to a specified set of rules.
- Disallow any refinements that reach erroneous conclusions for any case in a set of specified cases.
- Restrict learning refinement such that only attributes from the specified set may be used to add to an existing rule or to form a new rule.

3.1. Refinement Learning Procedure

The following procedure briefly outlines the techniques used to add components to existing rules and to create new rules:

Add a Finding to a rule: Specializing the Rule

1. While calculating the statistics for use by the heuristics, store a list of GAIN and LOSS cases for each rule. GAIN is the number of cases that would be gained if the rule was eliminated. LOSS is the number of cases that would be lost if the rule was eliminated.
2. The requirement for trying an experiment is that GAIN(rule)>0. Probable gain is less than or equal to GAIN.
3. Mark the LOSS cases as H+, the GAINs as H-, others ignored.
4. Generate the best attribute to be added to this rule.
5. If there is a best attribute, add it to the rule under consideration. Test.

Add A New Rule To The Knowledge Base:Generalization

1. Calculate the number of false positive and false negative cases for a given conclusion. If there are more FPs than FNs, skip the heuristic. Else proceed.
2. Go through all the cases. Mark all unknown, test cases and true positive cases to be ignored. Mark the FN cases as H+, and the rest as H-.
3. Generate the best attribute to be used as a new rule.

Generating the Best Attribute

The following table is computed for each attribute over the indicated set of cases:

	Attribute true	Attribute false
H+ cases	A	B
H- cases	C	D

1. Loop through the true/false findings. For each attribute FIN, consider both true and false attributes. Loop through each case to set up a predictive analysis table for each attribute.

2. Calculate the estimators and probable gain for each attribute.

   - a. For adding to an existing rule, estimator = A+D-B-C probable gain = D-B

   - b. For a new rule, estimator = A+D-B-C probable gain = A

3. Save the attribute with the highest estimator.

4. Loop through each numerical finding FIN.

5. Loop through the H+ cases to get each numerical VAL. Consider each attribute at each cutoff with greater and less than operators. Loop through each case to set up a predictive analysis table for each attribute. Calculate the estimator for each attribute. Save the best overall.

6. If the probable gain>0, return the best attribute.

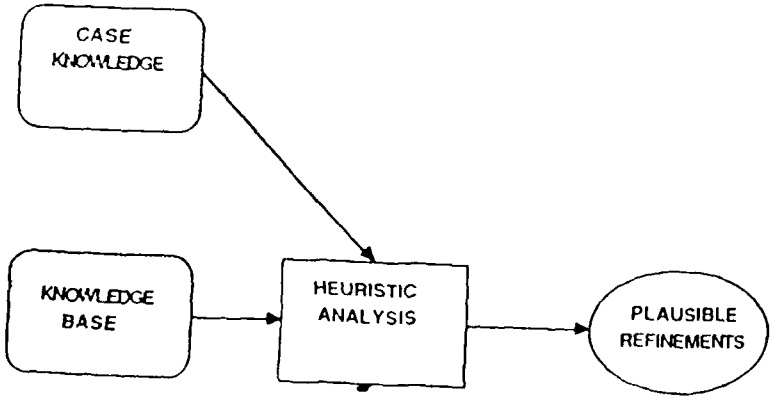
4. Overview of Research Approach

The following figure describes the fundamental approach of our knowledge base refinement research.

Knowledge Base Refinement

APPROACH.

Knowledge base refinement is the modification of an existing expert system knowledge base with the goals of localizing specific weaknesses in a knowledge base and improving an expert system's performance. Rutgers is proposing to demonstrate a rule refinement system in an application of the diagnosis of complex equipment failure: computer network troubleshooting.



OBJECTIVES.

The expert system should demonstrate the following advanced capabilities:

- automatic localization of knowledge base weaknesses
- automatic repair (refinement) of poorly performing rules
- automatic verification of new knowledge base rules
- some automatic learning capabilities
- development and testing of a new rule learning method, Predictive Value Maximization (PVM)

ACCOMPLISHMENTS.

- initial functioning equipment diagnosis and repair knowledge base, suitable for refinement. This is a subset of DEC's Network Troubleshooting Consultant (NTC).
- initial demonstration of functioning equipment diagnostic system with capabilities of localization of weak rules, automatic refinement, automatic verification.
- demonstration of initial rule learning capabilities.
- development of case generation simulator and randomized rule modifier.
- initial comparative studies demonstrating superiority of Predictive Value Maximization rule induction procedure.

Figure 4-1 described the simulation environment for the refinement system.

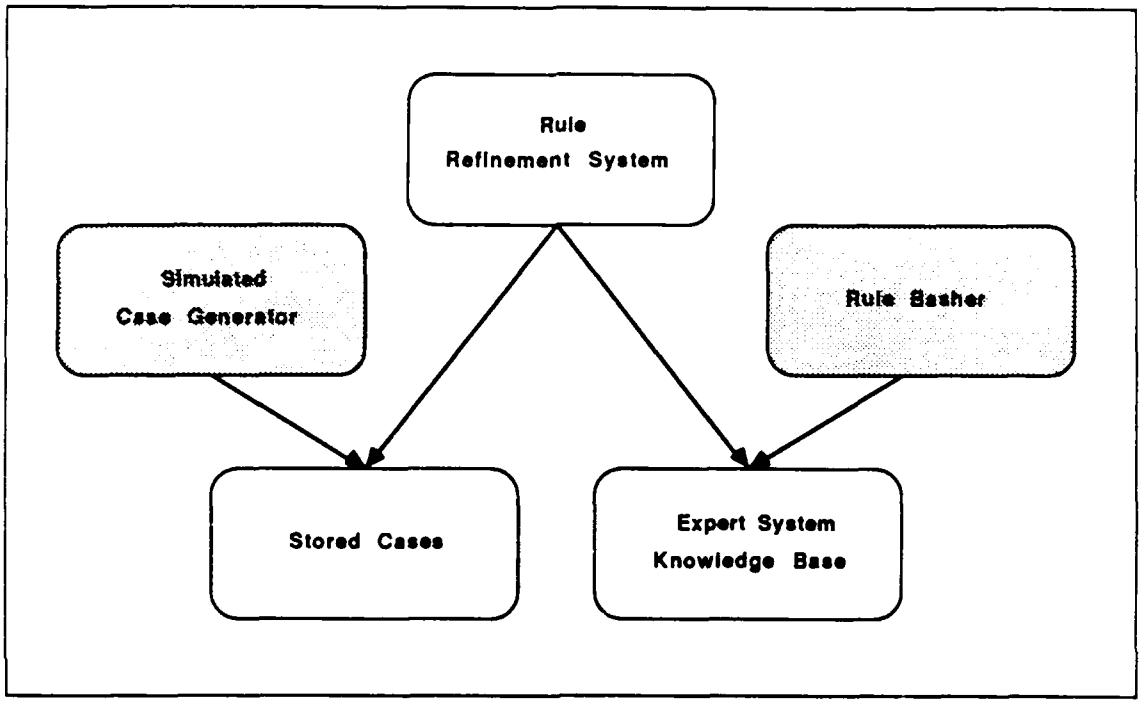


Figure 4-1: Simulation Environment for the Refinement System

5. Specific Research Reports

Summarizing the our key research accomplishments, the following papers and technical reports are included in this report: reports

- Ginsberg, A., Weiss, S., and Politakis, P., Automatic Knowledge Base Refinement for Classification Systems. Artificial Intelligence :197-226, 1988.

- Weiss, S. and Kapouleas, I., An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods, *Proceedings International Joint Conference on Artificial Intelligence.* (1989).

- Weiss, S., Galen, R., and Tadepalli, P., Maximizing the Predictive Value of Production Rules" *Journal of Artificial Intelligence,* in press (1990).

- Indurkhya, N. and Weiss, S., Models for Measuring Performance of Medical Expert Systems" *Artificial Intelligence in Medicine,* vol. 1, num 2, pp. 61-70 (1989).

Automatic Knowledge Base Refinement for Classification Systems

Allen Ginsberg,[1] Sholom M. Weiss, and Peter Politakis[2]
Department of Computer Science, Rutgers University, New Brunswick, New Jersey

[1]Current Address: AT&T Bell Laboratories, Holmdel, N.J.
[2]Current address: Digital Equipment Co., Hudson, Ma.

Abstract

An automated approach to *knowledge base refinement*, an important aspect of knowledge acquisition is described. Using empirical performance analysis, SEEK2 extends the capabilities of its predecessor rule refinement system, SEEK [17]. In this paper, the progress made since the original SEEK program is described: (a) SEEK2 works with a more general class of knowledge bases than SEEK, (b) SEEK2 has an *automatic* refinement capability, it can perform many of the basic tasks involved in knowledge base refinement without human interaction, (c) a metalanguage for knowledge base refinement has been specified which describes knowledge about the refinement process. Methods for estimating the expected gain in performance for a refined knowledge base and prospective test cases are described and some results are reported. An approach to justifying refinement heuristics is discussed.

1. Knowledge Acquisition and the Knowledge Base Refinement Problem

The problem of summarizing an expert's domain knowledge in an efficient formal representation, the *knowledge acquisition problem*, is a key problem in artificial intelligence research. As a practical matter, the most difficult aspect of expert system development is usually the construction of the knowledge base. The rate of progress in developing useful expert systems is directly related to the rate at which expert knowledge bases can be assembled.

The knowledge acquisition problem can be divided into two phases. In phase one the knowledge engineer extracts an initial rough knowledge base from the expert, rough in the sense that the overall level of performance of this knowledge base is usually not comparable to that of the expert. In the second phase, the *knowledge base refinement phase*, the initial knowledge base is progressively refined into a high performance knowledge base. In terms of a rule-based knowledge base, phase one involves the acquisition of entire rules, indeed entire sets of rules, for concluding various hypotheses. The refinement phase, on the other hand, is characterized not so much by the acquisition of entire rules but by the addition, deletion, and alteration of

rule-components in certain rules in the existing knowledge base, in an attempt to improve the system's *empirical adequacy*, i.e., its ability to reach the correct conclusions in the problems it is intended to solve. Obviously the foregoing description of knowledge base construction is an idealization. In practice the line between these two phases is not as sharply drawn.[1]

A knowledge base refinement problem can be thought of as an optimization problem in which we start with a proposed general solution to a given set of domain problems and the goal is to refine it so that a superior solution is obtained. The proposed solution is a working knowledge base that is in need of minor adjustments, but not a major overhaul, i.e., one assumes that the rules given by the expert are basically sensible propositions concerning the problem domain. The refinements applied to the rules of this knowledge base must not only meet the obvious requirements of being syntactically and semantically admissible, they must also be conservative, in the sense that they tend to preserve, as far as possible, the expert's given version of the rules. Employing rule refinements that meet these requirements makes it more likely that the construction of a refined knowledge base will not simply be a matter of curve fitting, but will result in a knowledge base with genuinely improved empirical adequacy, that at the same time remains close to the actual knowledge of the expert. Thus when we speak of optimizing the performance of a knowledge base, we mean improving performance on sample case data as much as possible, subject to constraints of conservatism.

2. Related Work

In this section we briefly consider the relationship of this work to other work on classification systems in statistical pattern recognition, traditional empirical machine learning, and the more recent explanation-based machine learning techniques.

Statistical pattern recognition and empirical machine learning classification systems both learn from case data. Statistical techniques assume a statistical distribution or a mathematical formalism; empirical machine learning classifiers are characterized by relatively simple sets of rules with logical AND/OR operators. While machine learning approaches usually attempt to completely cover the sample cases, empirically derived estimates of error rates on new cases can be relatively large [13].

Designers of rule based expert systems recognize the limits of learning directly from case data and rely on experts to provide summarizing rules of their experience. These rules can have intermediate hypotheses and sometimes heuristic procedures for combining uncertainty measures. Because these expert systems rely on knowledge and experience, they potentially can operate in

---

[1] In this paper we limit our concern to knowledge bases that are structured collections of production rules. By the term expert system, we mean a *production rule* based *classification* expert system [20, 1]. Furthermore we confine our attention to refinements of production rules that can be achieved as the result of the sequential application of certain generic refinement operations that are either generalization or specialization operations.

relatively large dimensions, far exceeding what could be extracted from even representative sample cases. Thus, case data have been traditionally limited to validation tests for rule-based expert systems.

Our approach makes significant use of some of the key features of statistical pattern recognition work, namely, an emphasis on optimizing the *overall performance* of the classification system, and the use of disjoint sets of cases for training and testing [5]. Train and test techniques have been applied to rules that have been learned directly from sample case data [13]. In our work, we show how a system with a knowledge base acquired from experts can use case data for validation, estimation of error rates, and knowledge base refinement.

Explanation-based approaches to machine learning assume that it is possible to prove that an instance falls under a concept using a *domain theory* [15]. This proof may then be generalized to yield a general concept description. While there have been many variations on this basic idea [4, 9, 18], explanation-based approaches share one feature: they depend upon knowledge - the domain theory - over and above the known instances (cases), the primitive concepts of the domain, and the experiential rules that can be acquired from domain experts.

Our approach to knowledge base refinement is designed to be used in applications where such domain theories are either unknown, inherently uncertain or statistical in nature, too time-consuming to specify, or too complex to be useful given current technology. For such domains it is either impractical or simply impossible to design a refinement mechanism that generates refinements that are guaranteed to be correct or lead to an improvement in overall performance. This is why we employ a heuristic refinement generation procedure, and why we test a candidate refinement for its effect on the overall performance of the classification system. However, our approach is compatible with, and can be improved through, the use of a domain theory when additional knowledge is available (See Section 9).

Previous research on knowledge base refinement, in the sense defined here, has been limited. Refining a large scale knowledge base is clearly a complex undertaking. Related empirical machine learning work in concept learning has concentrated on the learning of entire rules from case data and hierarchical descriptions [12], rather than the incremental refinement of rules acquired from a human expert. Because numerous expert systems are actively under development, in recent years there has been increased interest in immediate results to ease the knowledge acquisition task. While restricted to relatively simple knowledge bases and representations, a few papers have appeared on empirical refinement [11, 21]. Wilkins and Buchanan emphasize the need for a more complex form of analysis, similar to the optimization procedures used in SEEK2.

The form of empirical analysis that is described in this paper does not exclude other forms of analysis that are useful in knowledge acquisition. For example, systems that employ concepts and strategies for extracting domain knowledge via interaction with an expert have seen renewed

interest among researchers [3, 10, 6]. In addition, researchers working on the Programmer's Apprentice have reported some progress in improving knowledge acquisition in programming tasks by combining algorithmic fragments stored in software libraries with intelligent editing facilities [19]. While these efforts may be viewed as alternative techniques for knowledge acquisition, they may also be viewed as complementary to an empirical approach to refinement.

3. The Basic Approach: Empirical Analysis of Rule Behavior Using Case Knowledge

3.1. Overview

In this section we briefly review the basic approach to knowledge base refinement taken by SEEK [17] - an approach that we have continued to employ in SEEK2. A fundamental assumption of this approach is that case knowledge can be used in an empirical analysis of rule behavior in order to generate *plausible* suggestions for rule refinement (see Figure 3-1). Case knowledge is given in the form of a data base of cases with known conclusions, i.e., each case contains not only a record of the case observations but also a record of the expert's conclusion for the case. Empirical analysis of rule behavior involves gathering certain statistics concerning rule behavior with respect to the data base of cases; suggestions for rule refinements are generated by the application of refinement heuristics that relate the statistical behavior and structural properties of rules to appropriate classes of rule refinements. We will shortly explicate the nature of the statistical evidence gathered and give an example of these heuristics.

We have said that the goal of our heuristic analysis is to suggest plausible refinements. This is a term that could have a number of legitimate meanings in this context[2], and therefore, we need to clarify exactly what we mean. For us plausibility is a three-place relation holding among a contemplated refinement γ, a set of misdiagnosed cases M, and a body of evidence E regarding the behavior of various rules in the current knowledge base. Since we are using *heuristic* analysis, we do not, in general, expect the evidence to be so complete that we are able to determine whether γ will actually correct any of the misdiagnosed cases in M. Rather, our goal is to generate refinements that, given our evidence E, can be determined to have a chance of correcting one or more of the cases in M. The more cases γ has a chance of correcting, the greater its plausibility [7].

Given that a knowledge base can be expected to have a large number of rules, a refinement system must have a mechanism for focusing its attention on small subsets of rules as potential candidates for correcting certain misdiagnosed cases. The focus-of-attention mechanism employed in SEEK and SEEK2 is a *divide and conquer* strategy and depends upon certain reasonable assumptions concerning the logical structure of the knowledge base. We assume that the expert

---

[2]For example, see [18].

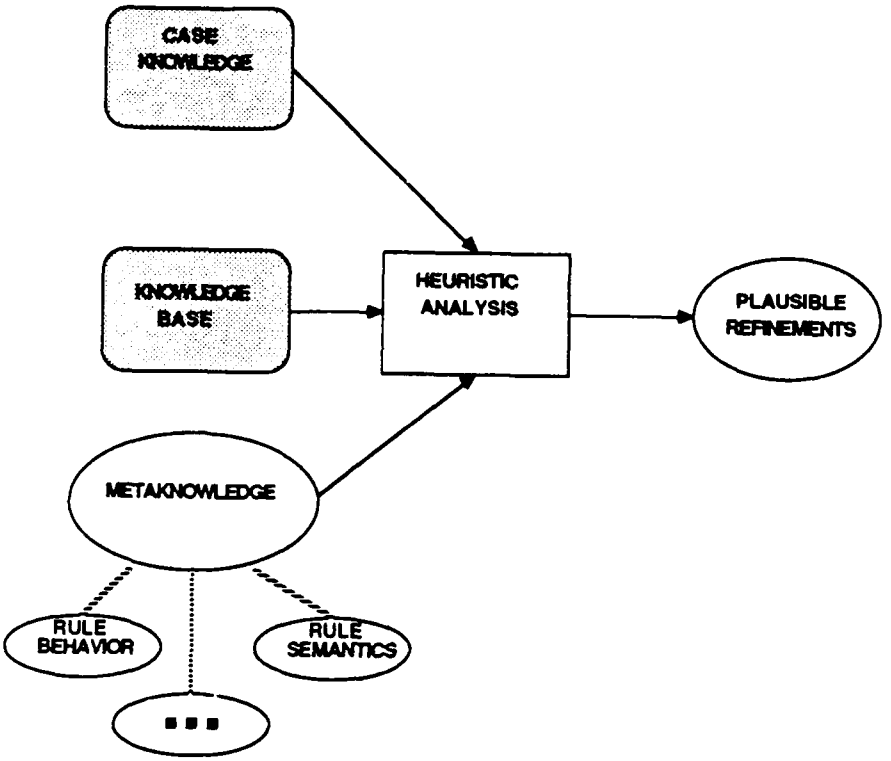


Figure 3-1: Basic Approach to Refinement

and knowledge engineer can identify a finite set of *final diagnostic conclusions* or *endpoints*; these are the conclusions that the expert uses to classify the given cases. We also assume that every rule has only a single conclusion. One can then confine one's attention to the refinement of rules that are involved in concluding a particular endpoint, e.g., if the domain is rheumatology one may decide to work on refining those rules involved in concluding the single final diagnosis Systemic Lupus. Thus at any given moment the system is applying the refinement heuristics only to a proper subset of the rules in the domain knowledge base. Even within this chosen subset, however, another focus-of-attention mechanism, based upon the *chaining* pattern of the rules in the knowledge base, serves to constrain the refinement generation process. If our chosen endpoint is Systemic Lupus, for example, we begin by applying the heuristics to all the rules in the knowledge base that directly conclude Systemic Lupus, i.e., rules whose right hand side is this conclusion. A rule that directly concludes some endpoint will, in general, have components on its left hand side that themselves are the conclusions of some other rules; such components are called *intermediate hypotheses*. The rules that conclude intermediate hypotheses may themselves include components that are intermediate hypotheses. Whenever the refinement heuristics suggest modifying an intermediate hypothesis IH, such as deleting it from some rule, the rules that conclude IH are thereby implicated as candidates for refinement.

3.2. Some Statistics and A Heuristic

At the highest level, many refinements of production rules may be thought of as falling in one of two possible classes: *generalizations and specializations* [14, 17]. By a rule generalization we mean any modification to a rule that makes it easier for the rule's conclusion to be accepted in any given case. A generalization refinement is usually accomplished by deleting or altering a component on the left hand side of the rule or by raising the confidence factor associated with the rule's conclusion. By a rule specialization we mean modifications to a rule that make it harder for the rule's conclusion to be accepted in any given case. A rule specialization is usually accomplished by adding or altering a component on the left hand side or by lowering the confidence factor associated with the rule's conclusion.[3]

On the side of evidence for rule generalization, one of the concepts we have employed in both SEEK and SEEK2 is a statistical property of a rule computed by a function that we call *Gen(rule)*. Gen(rule) is the number of cases in which

a. This rule's conclusion *should have been reached but wasn't*,

b. Had this rule been satisfied, the conclusion would have been reached, and

---

[3]Confidence factors are *combined* according to the following rule: when two or more rules for the same conclusion are both satisfied, the maximum (absolute) confidence is used.

c. Of all the rules for which the preceding clauses hold in the case, this one is the *closest to being satisfied.*[4]

For example during the processing of a case, the Gen(r_i) measure is incremented in the following hypothetical situation for rule r_i: Findings f_1 & f_2 & f_3 imply H_1 with confidence .9; H_1 is the correct answer; f_2 and f_3 are satisfied, but f_1 is not; and the incorrect computer conclusion has a confidence of .5.

On the side of evidence for rule specialization, one of the concepts we have defined is a statistical property of a rule that is computed by a function we call *SpecA(rule)*. SpecA(rule) is the number of cases in which

a. This rule's conclusion *should not have been reached but was*, and

b. If this rule had *failed to fire* the correct conclusion would have been reached.[5]

For example during the processing of a case, the SpecA(r_i) measure is incremented in the following hypothetical situation: Findings f_1 & f_2 & f_3 imply H_1 with confidence .9; the rule is satisfied, but H_1 is the wrong conclusion. SpecA(r_i) will be incremented when the second choice for the computer conclusion is the correct answer.

If there is *more than one satisfied rule* that concludes the incorrect first choice, then none of these rules has its SpecA measure incremented; instead we have defined an additional concept to cover this situation called *SpecB(rule)*: each of these rules has its SpecB measure incremented.

To get a feeling for the sort of heuristics employed by these systems, suppose that for a certain rule r it has been found that Gen(r) > [SpecA(r) + SpecB(r)], in other words the evidence suggests that it is more appropriate to generalize than specialize r. Another piece of information would help us decide which component of r should be deleted or altered, viz., the *most frequently missing component*, i.e., the component of r that has the lowest frequency of satisfaction relative to the cases that contribute to Gen(r). The function that computes this statistic is called *Mfmc(rule)*. Mfmc(rule) also tells us the syntactic category of this most frequently missing component. For example, one sort of component often used in medical diagnostic systems is called a *choice component*. These have the form *k*: C_1, ..., C_n, where *k, the choice number* is a positive integer and the C_i's are findings

---

[4]A measure of how close a rule is to being satisfied in a case, based on the minimal number of additional findings required for the rule to fire, is easily computed from the case data. Speaking figuratively, one may think in terms of paths through the knowledge base that would have led to the rule's being satisfied. Since the rule is unsatisfied, all these paths are blocked. We look for the path that would require the least number of changes in the case data to become unblocked. For details of the algorithm used by SEEK see [16]; SEEK2's closeness measure is essentially the same.

[5]The correct conclusion was the second choice in the case (due to its having the second highest confidence), and the only circumstance preventing its being the first choice is the fact that this rule is satisfied.

or hypotheses.[6] A choice component is satisfied *iff* at least k of its C_i's are satisfied. If we know that the rule r should be generalized and that Mfmc(r) is a particular choice component, then a natural thing to do is to decrease the choice number of that choice component. Being conservative we decrease the choice number by 1.

To summarize the discussion so far we now display in full the particular heuristic we have described.

If:	Gen(rule) > [SpecA(rule) + SpecB(rule)] & Mfmc(rule) is CHOICE-COMPONENT C
Then:	Decrease the choice-number of CHOICE-COMPONENT C in rule.
Reason:	This would generalize the rule so that it will be easier to satisfy.

A complete list of the refinement concepts and heuristics currently used in SEEK2 is given in Appendix A. Both generalization and specialization refinements are accomplished by modifying choice numbers, confidence measures, or numerical ranges. Generalization heuristics may delete rule components. For specialization, SEEK2 does not yet add components to rules. Our objective has been to explore the limits of a strictly conservative strategy of refinement.

4. The SEEK Experience

A salient feature of the original SEEK program [17] is that it was not designed to solve the entire knowledge base refinement problem on its own, rather it was intended to help, interactively, an expert or knowledge engineer solve the overall problem by offering potential solutions to various sub-problems that arise along the way. SEEK helps its user in the following ways: (a) it provides a performance evaluation of the knowledge base relative to the case data base, (b) using its statistical concepts and heuristics it identifies rules that are plausible candidates for refinement and suggests appropriate refinements, (c) a user can instruct SEEK to calculate what the actual performance results of a particular refinement to the knowledge base would be, and if the user desires, SEEK will incorporate the change in the knowledge base.

4.1. Basic Cycle of Operation

Although control in SEEK always resides with the user, and there are a number of paths and facilities available to the user at almost every point, SEEK can be thought of as having a basic cycle of operation. The system is given an initial knowledge base and the case knowledge data base.

---

[6]Findings are observations that are true, false or have a numerical value (such as age). Hypotheses are conclusions that may be assigned confidence values in the range of -1 to 1.

SEEK first obtains a performance evaluation of the initial knowledge base on the data base of cases. This is done by running the initial knowledge base on each of the cases in the data base, and then comparing the knowledge base's conclusion with the stored expert's conclusion. The performance evaluation consists primarily of an overall score, e.g. 75% of cases diagnosed correctly, as well as a breakdown by final diagnostic category of the number of cases in which the system agrees with the expert in reaching a particular diagnosis, i.e., *true positives*, and the number of cases in which the system reaches that diagnosis but the expert does not, i.e., *false positives*.

The user must decide on a diagnosis for which he would like to see refinements in the knowledge base in order to obtain better performance, e.g., if the domain is rheumatology the user may decide to try to upgrade the system's performance in diagnosing Systemic Lupus. For the sake of brevity, we call this user-specified diagnosis the GDX for the current cycle of operation, where the G stands for *given*, since this is a directive that the user must give the system. The next part of the cycle involves computing statistical properties concerning the rules of the knowledge base that conclude the GDX. Plausible refinements are then generated by evaluating a set of heuristics similar to the one presented above for each of these rules, as well as any rule that becomes implicated via an intermediate hypothesis (see Section 3.1 above).

Once SEEK has given its advice - we think of each piece of advice as a possible experiment to improve the knowledge base - the user will initiate an experimentation phase. This is a sub-cycle in which the user, interacting with SEEK, determines the exact effect of incorporating any one of the proposed experiments. The user will then decide which, if any, of these refinements should be accepted, and instructs SEEK accordingly. This ends the basic cycle, which can now be repeated starting with the modified knowledge base. This process continues until the user is satisfied with the overall performance evaluation.

4.2. Limitations

One of SEEK's limitations has already been mentioned: it does not have the capability to attempt to solve a refinement problem on its own. We discuss how SEEK2 removes this limitation in Section 5.1 below.

Another important limitation of SEEK is that it does not work with a general production rule system, rather it expects that the domain knowledge base will be written in a form known as the *criteria table representation*. This mode of representation requires the knowledge engineer to specify a list of *Major* observations and a list of *Minor* observations for each possible (diagnostic) conclusion in the knowledge base. Rules for reaching particular conclusions are then stated in terms of the number of Majors and Minors for the conclusion, *Requirements* and *Exclusions*. The latter are additional observations or conclusions, or conjunctions of such, that are relevant to the diagnosis: a Requirement is some condition that must be satisfied to reach the conclusion; an Exclusion is some condition that rules out the conclusion. Furthermore, any rule can reach its

conclusion at one of only three possible confidence levels, viz. possible, probable, definite. As an example, assuming that a list of majors and minors for the conclusion Systemic Lupus has been specified, a rule for concluding the latter might state that if (at least) two of the majors and two of the minors are present then the conclusion is warranted at the definite level.

While this mode of representation has proven to be useful in the rheumatology domain [16] and other medical applications, it is in fact not as powerful a representation language as that of EXPERT [20] or similar production rule systems, in the sense that one can write knowledge bases in general production rule languages that are not translatable into the criteria table format. However, any criteria table can be translated into production rule syntax. Thus the set of criteria table knowledge bases is a proper subset of the set of production rule knowledge bases.

SEEK's knowledge engineering knowledge, i.e., its statistics and heuristics, was formulated with reference to the criteria table representation scheme, and criteria table concepts also were embedded in the control structure of the program. As a consequence, certain forms of rule refinement were not available or were restricted in SEEK, e.g., changing a rule's confidence factor was limited to making jumps from one level to another, such as probable to possible. In general, SEEK could do very little with a knowledge base that was not written in a criteria table format.

SEEK2, on the other hand, is a refinement system that will work with any knowledge base written in EXPERT's rule representation language.[7] In designing SEEK2, we found it was possible to decouple SEEK's knowledge engineering concepts from the criteria table representation; we were able to apply many of these concepts in relation to features of more general types of production rules. For example, criteria table rule-components using the notion of Majors and Minors are special cases of rule-components using choice-functions. Decreasing or increasing the number of Majors or Minors required by a rule, is a special case of decreasing or increasing the choice-number of a choice-function. Thus the example heuristic given above in Section 3.2 is a generalization in SEEK2 of two separate similar heuristics originally stated in SEEK, one for Majors, one for Minors.

In moving to a more general representation language as the target language for knowledge base refinement, we broadened the scope of the set of generic refinement operations available to the system. For example, confidence factors for generalization experiments may be increased based on an average of the highest-weighted (erroneous) conclusion for a set of misclassified cases.[8]

From the programmer's point of view, SEEK's *own knowledge base*, the representation of its

---

[7]This language is tailored to a classification system. Unlike OPS5, where rules fire one at a time, all rules that match are evaluated, and a relatively simple confidence scoring system is employed. The current design of SEEK2 assumes a single correct conclusion.

[8]Changing the confidence measure of a conclusion is usually considered more *radical* than minor changes to the left hand side of rules.

knowledge engineering statistics and heuristics, was strictly separate from its control structure. However, this was not the case from the point of view of the user, since there was no facility by which the user, qua user and not qua programmer, could access and modify SEEK's own knowledge base, in the way that a user can modify the domain knowledge base. Our approach to this issue forms part of a broader project which we describe in Section 8.1 below.

5. The SEEK2 Refinement System

5.1. Automatic Refinement Capability

Unlike SEEK, SEEK2 is a system that can present plausible solutions to the overall refinement problem *without* the need for interaction with an expert. The output of SEEK2 running in automatic mode is not a list of suggested rule refinements for a particular GDX (Given Dx), rather it is a refined version of the entire knowledge base, i.e., a set of rule refinements to the initial knowledge base which yield an improvement in overall performance. In this section we describe SEEK2's current automatic refinement capability. Figure 5-1 is an overview of SEEK2's automatic refinement control strategy.

The attempt to find a sequence of refinements that optimizes performance is a search problem. Where there is a search problem of sufficient complexity, good heuristics must be found to guide the search. As we will see, SEEK2's current automatic refinement algorithm is a heuristic search algorithm, in the sense that it uses a classic weak method, hill-climbing.

When running in automatic mode SEEK2 makes three types of decisions that were previously made by the user of SEEK: (a) choice of GDX for the current cycle, (b) which rule refinement experiments to try, (c) which refinements to incorporate in the knowledge base given the results of the experiments (see Figure 5-2). Additionally SEEK2 has to know when to stop.

In the current implementation, SEEK2 orders the potential GDXs in descending order according to a simple measure on the number of false negatives and false positives, information that is given by the performance evaluation phase. Potential rule refinement experiments for a GDX are ordered by simple measures on the statistics used in generating the refinement, e.g., if the generalization heuristic given in Section 3.2 fires, the quantity Gen(rule) - [SpecA(rule) + SpecB(rule)] is used as an estimate of the *expected net gain* to be derived by performing the experiment.

Information of this sort could be used to limit the number of experiments performed in a cycle. However, in the current implementation, the information is used only to determine the order in which GDXs are chosen and experiments attempted; ultimately *every* potential GDX (for which perfect performance has not been obtained) is chosen, and every experiment suggested by the heuristics is performed. In other words, an automatic refinement cycle involves attempting,

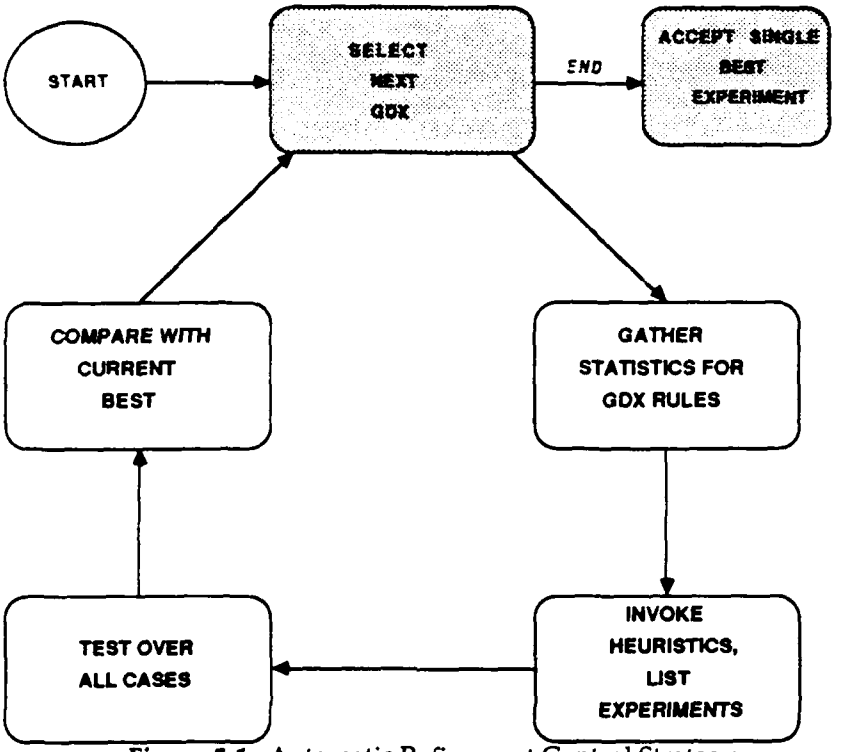


Figure 5-1: Automatic Refinement Control Strategy

according to the ordering just given, every proposed refinement experiment for every potential final diagnostic conclusion in the knowledge base. (Of course, the number of experiments generated by the heuristics represents a small fraction of the total number of logically admissible refinements.) Of all these attempted experiments, SEEK2 accepts only one, the one that gives the greatest *net gain* in knowledge base performance for *all* final diagnostic conclusions, not just for one GDX.[9] An internal record of the accepted refinement is kept; and then the next automatic refinement cycle begins. If the current automatic refinement cycle is such that no attempted experiment leads to an actual net gain, SEEK2 stops.

We present a simplified example in order to illustrate these concepts. Let us suppose that we have a rheumatology knowledge base dealing only with the two final diagnoses Systemic Lupus and Rheumatoid Arthritis, that a data base of 20 cases is available, and that our human expert has diagrosed 10 of these cases as Systemic Lupus and the other 10 as Rheumatoid Arthritis. Suppose that the initial performance evaluation computed by SEEK2 is as follows:

DX	True Positives	False Positives
Rheumatoid Arthritis	9 / 10	6
Systemic Lupus	3 / 10	1
None	0 / 0	1
Total	12 / 20	8

The measure SEEK2 uses to compute GDX order is the maximum of the false negatives and false positives. Thus in our example Systemic Lupus would be the first diagnosis in the GDX ordering since it has 7 false negatives, i.e., 7 out of the 10 cases that should have been diagnosed as Systemic Lupus were not. Therefore SEEK2 will first generate refinement experiments for Systemic Lupus.

Continuing the example, suppose that rule r concludes Systemic Lupus, and SEEK2 finds that Gen(r) = 6, SpecA(r) = 1, SpecB(r) = 0, and Mfmc(r) = Choice-Component C. These findings would satisfy the antecedent of the refinement heuristic presented in Section 3.2. Therefore SEEK2 will post the decreasing of C's choice-number as a refinement experiment. SEEK2's estimate of the expected net gain of performing this experiment is given by Gen(r) - [SpecA(r) + SpecB(r)] = 5. (This is an estimate; the only way to know what the precise effect of decreasing the choice-number of C will be, is to decrease it, and then recompute the system's performance on *all* cases in the data base.) Once all the refinement experiments for Systemic Lupus have been posted and ordered according to their expected net gain, SEEK2 performs *all* the experiments on this list as ordered. If

[9]While the overall performance must increase, it is possible that the performance of some conclusions may decrease.

SEEK2 finds that decreasing the choice-number of component C in rule r leads to an overall performance gain of 3 cases, i.e., the bottom line performance total for *both* Rheumatoid Arthritis and Systemic Lupus improves from 12 to 15/20, and this turns out to be the maximum net gain of all the experiments for Systemic Lupus, SEEK2 records this fact.

Next, it will select Rheumatoid Arthritis as the GDX, and repeat the process. Suppose that the aforementioned experiment for rule r yields a greater net gain than the best refinement experiment for Rheumatoid Arthritis. Then SEEK2 will accept the refinement to rule r, i.e., it will modify its internal copy of the domain knowledge base to reflect this refinement, and a new cycle will commence.

The automatic refinement algorithm is a hill-climbing procedure: at each step SEEK2 is guided totally by local information as to which proposed refinement of the current knowledge base results in the best improvement. SEEK2 stops when none of the experiments suggested by the heuristics leads to a net gain. Because SEEK2 does not examine all possible combinations of refinement experiments, the accepted experiments may fail to achieve a global maximum. Moreover, because SEEK2's heuristics are estimators that do not consider every possible refinement, the accepted experiments may also fall short of a local maximum. SEEK2's optimization goal is tempered by constraints of conservatism and computational efficiency.

5.2. Using Disjoint Training and Testing Sets

The best evidence for the validity of an approach can come only from actual examples of its successful use. On this score, we can say that with respect to the rheumatology knowledge base we have used as a test case, SEEK2 has generated refinements that are similar to those produced by SEEK, some of which were found to be acceptable to the experts [16].[10]

However, some evidence of the reliability of the approach in producing refinements that improve the general empirical adequacy of a knowledge base (not just its empirical adequacy with respect to the given data base of cases) can be obtained via experimentation with a single knowledge base by using various statistical techniques. In this section we describe the results of such an experiment.

The experiment will be called a *Train-and-Test* experiment, and is actually a series of similar experiments or runs. In a single typical train-and-test run the given data base of cases is divided into two randomly selected disjoint subsets (not necessarily of equal size) preserving the distribution of cases by endpoint. Let us call these sets σ_1 and σ_2. The first phase of a train-and-test experiment involves running SEEK2 using σ_1 as case knowledge, or as the training

[10]While the SEEK refinements required human intervention during many of the refinement cycles, SEEK2 used automatic refinement procedures.

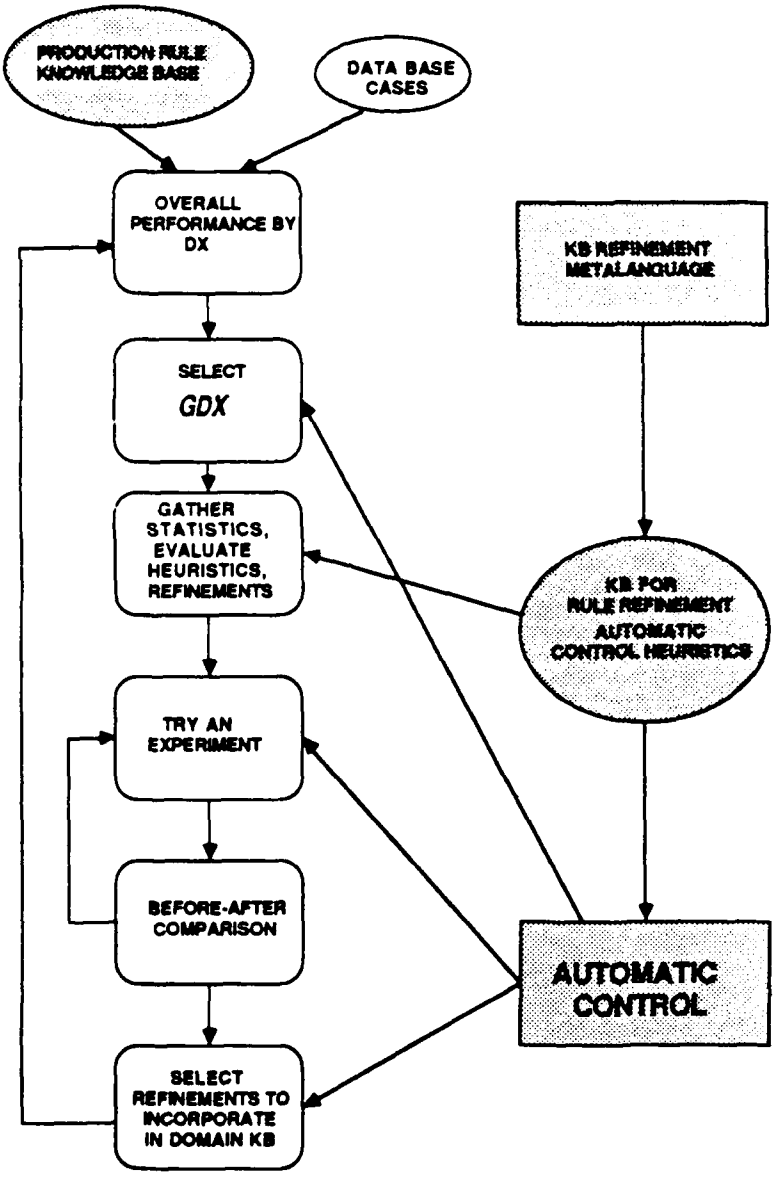


Figure 5-2: Overview of SEEK2; shaded portions indicate extensions to SEEK

set. SEEK2's refined version of the knowledge base is then tested over σ_2 (and the combined set $\sigma_1 \cup \sigma_2$). In the second phase of the experiment the roles of σ_1 and σ_2 are interchanged.

Figure 5-3 gives the results of such a run with training and test samples of equal size. Training over σ_1 led to a performance increase of 29% (69% to 98%). When tested over the new set of cases in σ_2, there was an increase in performance of 15% (78% to 93%). The results of the second run are similar. While there was often less improvement observed over the test sets than in the training sets in these runs, the fact is that the experiment shows that refinements that were learned by SEEK2 with respect to one set of cases also improved empirical adequacy with respect to a new set of cases.

This experiment has a more precise interpretation. A single train-and-test experiment can be viewed as giving an *estimate* of the *probability of error* - i.e., the probability that the refined knowledge base (classifier) misdiagnoses a case [5]. Under this interpretation, the total performance ratio obtained in the test run, e.g., the 93% figure in testing set 1, estimates the probability of error to be .07. While this figure is certainly more conservative than the .02 estimate that would be obtained by using the results of the training run as an estimate, it is only a point-estimate. To obtain a more reliable estimate, one needs to average over the results of many train-and-test experiments. Alternatively, a more accurate figure could also be obtained by employing a *leaving-one-out* or jackknifing technique for error estimation [5]. Unfortunately, the general application of these techniques in knowledge base refinement would appear to be computationally prohibitive for large-scale problems.

However, additional train-and-test runs for the rheumatology knowledge base have been performed and these results can be used to derive a more reliable estimate of the probability of error. In all, 15 train-and-test runs were conducted. In 6 runs, the size of the training sample was 50% of the cases; 3 runs each with training sample sizes of 33%, 67%, and 75%, respectively, were also conducted. The average performance increase observed in the test cases in these 15 runs was 21.2%. The average total performance over the test cases was 94.5%, which gives an estimate of probability of error of .055. The lowest overall performance over a test set obtained in any of these runs was 90% (this occurred in a run with training sample size 67%), which yields a .1 point estimate of probability of error. The highest overall performance over a test set obtained in any of these runs was 100% (this occurred in a run with training sample size 75%). The average number of rule refinements to the knowledge base was 8.

As is shown in the sample session below (see Section 6), a recent implementation of SEEK2 allows the user to perform such Train-and-Test experiments.

Train and Test Experiment 1		
	Training Set 1	Testing Set 1
Start	42/62 (69%)	46/59 (78%)
Finish	61/62 (98%)	55/59 (93%)
	Overall:	116/121 (96%)

Train and Test Experiment 2		
	Training Set 2	Testing Set 2
Start	46/59 (78%)	42/62 (69%)
Finish	59/59 (100%)	59/62(95%)
	Overall:	118/121 (98%)

Figure 5-3: Train and Test Experiment

6. A Sample Session

A sample session is abstracted below. Annotations are italicized; user inputs are boxed.

Enter knowledge base name: RHEUM

The following option allows the user to establish a confidence threshold, Θ, to determine whether a case is to be considered correctly diagnosed when a tie occurs. If the knowledge base reaches the correct conclusion as well as an additional erroneous conclusion at confidence level $\alpha \geq \Theta$, the case will be considered misdiagnosed. If $\alpha < \Theta$ the case will be considered correctly diagnosed.

Enter threshold below which ties are wins (1.1): 0.5

The user has the option of dividing the data base of cases into disjoint, randomly selected, training and testing sets, and can specify the percentage to be used for training versus testing. In this example the training sample size is 50%.

Would you like to set up a training sample? * Y

Enter percentage of cases to be in training sample (1:99): 50

SEEK2 now prints out the PDX (expert's conclusion) and the CDXs (knowledge base's current conclusions) for each case. Mnemonics are used to reference specific conclusions. This output can be suppressed.

```
1.   Pdx: MCTD     Cdx:   PSS (.4)    SLE (.4)     RA (.4)
2.   Pdx: RA       Cdx:   RA (.9)     SLE (.4)
                            •
                            •
                            •
121. Pdx: RA       Cdx:   RA (.7)
```

The system now prints the current performance breakdown by endpoint for the training set, the test sets, and the union of these sets.

TRAINING SAMPLE

	True Positives	False Positives
RA	21/21 (100%)	7
MCTD	4/16 (25%)	0
SLE	6/9 (67%)	1
PSS	11/11 (100%)	3
PM	1/2 (50%)	1
NONE	0/0 (0%)	4
Total	43/59 (73%)	16

TEST CASES

	True Positives	False Positives
RA	21/21 (100%)	4
MCTD	5/17 (29%)	0
SLE	6/9 (67%)	3
PSS	11/12 (92%)	1
PM	2/3 (67%)	0
NONE	0/0 (0%)	9
Total	45/62 (73%)	17

ALL CASES

	True Positives	False Positives
RA	42/42 (100%)	11
MCTD	9/33 (27%)	0
SLE	12/18 (67%)	4
PSS	22/23 (96%)	4
PM	3/5 (60%)	1
NONE	0/0 (0%)	13
Total	88/121 (73%)	33

Because performance on the MCTD cases is the weakest, SEEK2 chooses to refine rules for MCTD first. It lists the relevant cases; a case number with an X after it indicates that this case is not part of the training sample.

GDX: MCTD

Relevant cases for this DX:

1.X	Pdx: MCTD	CDX:	PSS (.4)	SLE (.4)	RA (.4)		
4.X	Pdx: MCTD	CDX:	SLE (.4)				
11.	Pdx: MCTD	CDX:	PSS (.7)	MCTD (.4)	SLE (.4)	RA (.4)	
12.	Pdx: MCTD	CDX:	RA (.4)				

•
•
•

The system now evaluates its refinement concepts and heuristics with respect to the relevant cases and rules, and posts the refinement experiments generated. It then attempts each experiment in order of decreasing probable gain. In the following example, the system is on the first cycle of refinements; no experiments have been accepted. MCTD is ranked as the top candidate for refinement.

Suggested Experiments:

Cycle: 1; GDX rank: 1; Number of GDX experiments: 1; Total experiments: 1

1. In rule 3.7, decrease the choice number of component-1 from 3 to 2
Probable gain: 8

Rule 3.7

Choose 3 or more of the following:

Hypothesis RAYES has confidence between .9 and 1
Finding SWOLH is true
Finding SCLDY is true
Finding DCO is less than 70
Hypothesis MYOSS has confidence between .9 and 1

CONCLUDE Hypothesis MCTD with confidence .4

The system prints the result, on a case-by-case basis, of performing this experiment. The net gain of the experiment over the training cases is also given.

1.X	Pdx:	MCTD	CDX:	MCTD (.4)	PSS (.4)	SLE (.4)	RA (.4)
2.	Pdx:	RA	CDX:	RA (.9)	SLE (.4)		
				•			
				•			
				•			
121.	Pdx:	RA	CDX:	RA (.7)			

Net gain for experiment: 5 cases.

After considering 13 experiments, SEEK2 determines that the first experiment is the best. The knowledge base is modified, and the first cycle of experimentation is complete. With the newly modified knowledge base, a new cycle of experimentation begins. SEEK2 prints the the following performance summary of the accepted experiment.

EXPERIMENT 1 ACCEPTED. CYCLE 1 completed.

TRAINING SAMPLE

BEFORE

	True Positives	False Positives
RA	21/21 (100%)	7
MCTD	4/16 (25%)	0
SLE	6/9 (67%)	1
PSS	11/11 (100%)	3
PM	1/2 (50%)	1
NONE	0/0 (0%)	4
Total	43/59 (73%)	16
		•
		•
		•

AFTER

True Positives	False Positives
21/21 (100%)	6
9/16 (56%)	0
6/9 (67%)	1
11/11 (100%)	1
1/2 (50%)	1
0/0 (0%)	2
48/59 (81%)	11

The following sequence of experiments illustrates the way in which SEEK2 attempts to find less radical alternatives to certain classes of refinements, and it also illustrates the use of the logical structure of the knowledge base to control the refinement process.

Cycle: 2; GDX rank: 3; Number of GDX experiments: 4; Total experiments: 26

4. In rule 4.11, delete component-3 referencing hypothesis RD203
Probable gain: 1

Rule 4.11

Choose 2 or more of the following:

- **Hypothesis NEPH has confidence between .9 and 1**
- **Finding MALAR is true**
- **Hypothesis SEROS has confidence between .9 and 1**
- **Hypothesis CNS has confidence between .9 and 1**
- **Finding HEMAN is true**

AND

Choose 2 or more of the following:

- **Finding FEV is true**
- **Finding ARTH is true**
- **Finding GGLOB is greater than 1.8**
- **Hypothesis HCMP has confidence between .9 and 1**
- **Finding PLAT is less than 100**

AND

Hypothesis RD203 has confidence between .9 and 1

AND

Hypothesis EX1SL has confidence between -1 and .05

CONCLUDE Hypothesis SLE with confidence .9

Net gain for experiment: 1 cases.

Since the component considered for deletion has an associated confidence range, SEEK2 goes on to suggest generalizing the range as an alternative to deletion. Because the component being considered for deletion is an intermediate hypothesis, SEEK2 also will suggest ways of modifying the rules that conclude this intermediate hypothesis as an alternative to modifying rule 4.11.

•
•
•

7. Justification of the Heuristics

While certain parts of the heuristics used in SEEK and SEEK2 are obviously reasonable, e.g., to generalize a choice component one may decrease its choice number, the statistical comparisons used in these heuristics may require some explanation.

We can view the basic goal of heuristic refinement generation as being the generation of refinements that maximize expected gains in performance with respect to a given set of misdiagnosed cases M. Or we can view it as being tempered by the desire not to degrade

performance over the set, C, of cases currently diagnosed correctly; the goal is to generate refinements that maximize expected gains over the misdiagnosed cases in M, but at the same time minimize the expected losses over C. Let us call the first policy *Max_Gain*, and the second *Max_Gain+Min_Loss*. SEEK and SEEK2 adopt the latter policy, and this is why all their heuristics contain a clause that is intended to compare expected gain with expected loss.

To elaborate on this point, we need to define some terms. We say that a knowledge base has made a *True Positive* (TP) judgment whenever (the expert system using) it reaches an endpoint that was also reached by the expert; we say that it has made a *False Positive* (FP) judgment whenever it reaches an endpoint that was not reached by the expert; we say that it has made a *True Negative* (TN) judgment whenever it refrains from reaching an endpoint that the expert also did not reach; and, finally, we say that the knowledge base has made a False Negative (FN) judgment whenever it refrains from reaching an endpoint that the expert did reach.

The goal of the overall knowledge base refinement process is to minimize the number of FP and FN judgements of the knowledge base, consistent with the constraints of conservatism. (Minimizing FPs is equivalent to maximizing TNs and minimizing FNs is equivalent to maximizing TPs.) Given this overall goal, a generalization refinement may be seen as an attempt to contribute to it by *increasing the number of TPs* for an endpoint (equivalently, decreasing the number of FNs for that endpoint). A specialization refinement is an attempt to contribute to the overall goal by *decreasing the number of FPs* for an endpoint (equivalently, increasing the number of TNs for that endpoint).

However, anytime a generalization is made there is a possibility that the refinement will lead to an increase in the number of FPs for the endpoint in question as well, which is clearly at odds with our goal. Anytime a specialization is made there is a possibility that the refinement will lead to an increase in the number of FNs (equivalently, a decrease in the number of TPs) for the endpoint in question as well, which is also at odds with our goal.

The role of the refinement generator is to produce refinements that not only have the chance of reducing the number of FPs and FNs over the misdiagnosed cases in M, but that also have the least chance of generating new FPs and FNs over other currently correctly diagnosed cases in C. We now formulate two general criteria, for generalization and specialization refinements respectively, that characterize this point of view.

Let ΔTP represent the total net change (over all cases and all endpoints) in TPs that will occur due to a generalization refinement R_g; let ΔFP represent the total net change in FPs that will occur due to R_g. Then the refinement R_g contributes to our overall goal if

$$\Delta TP > 0 \text{ and } \Delta TP > \Delta FP \quad (1).$$

Similarly, let ΔTN represent the total net change (over all cases and all endpoints) in TNs that will occur due to a specialization refinement R_s; let ΔFN represent the total net change in FNs that will occur due to R_s. Then the refinement R_s contributes to our overall goal if:

$$\Delta TN > 0 \text{ and } \Delta TN > \Delta FN \quad (2).$$

An optimal heuristic for generating generalization refinements would be one that never suggested a refinement that violates condition (1), and, an optimal heuristic for generating specializations would never suggest a refinement that violates condition (2). It is doubtful that there are any truly heuristic principles that are optimal in this sense. One has to settle for something that is less than optimal, and computationally feasible as well.

This is where refinement concepts such as Gen, SpecA, and SpecB, come into the picture. These functions have a twofold character: they can be used as indicators of problematic rule behavior, but they can also be used as estimators of expected gains due to refinements. Thus Gen can be used as an estimator of ΔTP for appropriate generalization refinement operations, and, intuitively, this seems plausible. Therefore, if we can find a plausible estimator of ΔFP for generalization refinements then we will be able construct heuristics for generating generalizations that use these estimators as an app ɔximation to condition (1). A seemingly good concept for this role would be something like the following:

AntiGen(r) = the number of *correctly diagnosed cases* in which:
- (a) r is not satisfied,
- (b) if r *had been satisfied* the case would have been misdiagnosed, and
- (c) r is the closest to being satisfied of all the rules for which (a) and (b) are true.

The problem with AntiGen(r) is that from a computational point of view it is not consistent with the general refinement strategy adopted in SEEK2. When attempting to refine rules involved in reaching endpoint DX, SEEK2 gathers information concerning cases in which either DX is reached by the expert or DX is concluded by the knowledge base. It does not gather information from cases in which DX does not figure as either the expert's or knowledge base's conclusion. Aside from the fact that the former cases are clearly most relevant to the discovery of useful refinements for these rules, there is also a computational rationale for this policy. On the average we expect that for any endpoint there will be far fewer cases in which it is reached by the expert or knowledge base, than cases in which it is not reached by the expert or knowledge base, especially if there are several *a priori* equally probable endpoints. To compute AntiGen(r) requires one to analyze every correctly diagnosed case in the data base of cases having a stored expert conclusion that does *not* match the conclusion of r (to see whether the satisfaction of r would lead to a new false positive). Thus AntiGen(r) does not conform to the policy adopted in SEEK2.

SEEK2 uses the quantity [SpecA(r) + SpecB(r)] as an estimator of ΔFP due to a generalization refinement, and therefore the condition

$$\text{Gen}(r) > [\text{SpecA}(r) + \text{SpecB}(r)]$$

that appears in some of SEEK2's generalization heuristics are approximations to condition (1).

Although in this context we are using [SpecA(r) + SpecB(r)] as an estimator of ΔFP, the main role of this quantity is as an estimator of ΔTN due to a specialization refinement.[11]

A similar account of the role of the conditions used in SEEK2's specialization heuristics can be given. For example the comparison:

$$SpecA(r) > Signif(r)$$

is used as an approximation to condition (2). In this case, SpecA(r) is an estimator of ΔTN due to a specialization refinement, and Signif(r) is an estimator of ΔFN for the same refinement. These correspondences are intuitively plausible, as the reader can verify by scanning the appropriate definitions in Appendix A.

8. A Metalanguage for Knowledge Base Refinement

In this section we briefly describe some of the important primitives of a metalanguage designed specifically for the refinement task [7, 8]. Using this metalanguage one can define general knowledge engineering concepts and heuristics, such as *Gen(rule)*, as well as domain-specific metaknowledge in terms of a set of primitive concepts and operations.

8.1. General and Domain-Specific Metaknowledge in Knowledge Base Refinement

Refinement concepts such as Gen, Mfmc, SpecA, and SpecB and the heuristics that employ them, are examples of *general metaknowledge*, i.e., general knowledge about the conditions under which rules should be considered for refinement. Other examples of general metaknowledge would include concepts and strategies for extracting domain knowledge via interaction with an expert [3, 10], as well as concepts needed to encode knowledge concerning the strategic role of rules within the overall classification or diagnostic process [2]. An example of *domain-specific metaknowledge* [2, 18] is that certain rules in a knowledge base are definitional and should never be modified. Such knowledge involves properties of a knowledge base that are not ascertainable by means of a general a priori procedure.

8.2. Motivation for a Metalanguage

While SEEK2 is currently based on general metaknowledge, there are ways in which domain-specific metaknowledge could be used in this system. For example, the knowledge that certain rules in a knowledge base are definitely in their correct form could be used to prevent the system from gathering data and attempting refinement experiments for such rules. By specifying refinement systems in a flexible metalinguistic fashion it may be possible to capture and incorporate such knowledge in the refinement process as the need and opportunity arise.

---

[11] It is a measure of current false positives that might be corrected.

Another motivation for a metalanguage was alluded to in Section 4.2, where we mentioned that SEEK's knowledge base of heuristics and statistics was inaccessible to the user of the system. The ability to access and modify this knowledge base is quite desirable for designing and experimenting with refinement concepts. For example, some of the current statistics for SEEK2 may not be as useful with respect to an expert system that employs a scoring scheme for combining confidence factors. Useful variants of these statistics could be defined within the same metalanguage that we have developed for SEEK2. A high-level framework for the specification of refinement systems thus provides an environment for conducting research in knowledge base refinement.

Finally, another motivation for a refinement metalanguage is the issue of customization. In general, even within one expert system framework, different styles of knowledge bases are possible; it is likely therefore that different styles of refinement will be needed. For example, some knowledge bases employ a taxonomic ordering of hypotheses. Such an ordering provides knowledge that could be used, together with appropriate control heuristics, to formulate a more efficient version of SEEK2's automatic refinement algorithm. A knowledge base refinement metalanguage allows for the representation of such control heuristics (see Figure 5-2). Indeed, the experimental metalanguage that is being developed allows a user to specify different control strategies from the hill-climbing strategy employed in SEEK2. Several alternative control strategies are mentioned briefly in Section 9.

8.3. Some Metalinguistic Primitives

SEEK2's statistical concepts can be specified in a set-theoretic metalanguage that employs only a small number of refinement primitives together with some appropriate notions from simple set theory, arithmetic, and logic. Using these primitives it is possible to experiment with variations on SEEK2's statistics and define domain specific statistics as well.

A set-theoretic definition of concepts such as *Gen(rule)* (see Section 3.2) requires refinement primitives of the following sorts. Some primitive variables are needed to provide the system or a user with the ability to access various objects in the domain knowledge base and the data base of cases. For example, *rule* is a variable whose range is the set of rules in the domain knowledge base, *case* is a variable whose range is the set of cases in the data base of cases, and *dx* is a variable whose range is the set of possible final diagnostic conclusions in the knowledge base. In addition some primitive functions are needed to allow one to refer to selected parts or aspects of a rule or a case, e.g., *RuleCF(rule)* is a function whose value is the confidence factor associated with *rule*, *PDX(case)* is a function whose value is the expert's conclusion in *case*,[12] and *CDX(case)* is a function whose value is the conclusion reached for the current knowledge base in *case*. As an example of the way in which these primitives can be used, note that using the notions of PDX(*case*) and CDX(*case*) one may define a *misdiagnosed case* as any *case* for which PDX(*case*) ≠ CDX(*case*).

[12]PDX stands for Physician's Diagnosis and CDX stands for Computer's Diagnosis.

Certain special sets of objects are of importance in the knowledge base refinement process, and it is therefore useful to have primitives that refer to them, e.g., *Rules-For(dx)* is a function whose value is the set of rules that have *dx* as their conclusion. Finally various primitives that in some way involve semantic properties of rules, or the performance characteristics of the knowledge base as a whole are useful, e.g., *Satisfied(kb-item, case)* is a predicate that is true *iff kb-item* is satisfied by the findings in *case*, and false otherwise, where kb-item can be a identifier for a rule, a rule component or subcomponent; *ModelCF(dx, case)* is a function whose value is the system's confidence factor accorded to *dx* in *case*.

The current heuristic rules in Appendix A can be described in this metalanguage. For further details on the metalanguage see [7, 8].

9. Discussion

SEEK2 currently has ten statistical concepts and nine heuristics for generating refinements. Working in automatic mode on a rheumatology knowledge base of approximately 140 rules with 5 final diagnostic categories, and using a data base of 121 cases, SEEK2 was able to increase the overall performance of the system from a value of 73% (88/121) to a value of 100% (121/121). It used approximately 32 minutes of Vax-785 cpu time. The total number of experiments tried was 199, out of which 10 were accepted.

In evaluating the usefulness of SEEK2's automatic refinement capability it is important to keep in mind that the expert is still the final judge. Despite the assured gain in performance with respect to the *given* data base of cases, and the reasonable expectation of performance enhancement with respect to *new* cases, the expert may agree with only a subset of the total number of refinements suggested by SEEK2[13]. The measure of SEEK2's usefulness is not, however, simply how many of its experiments the expert accepts; even rejected experiments have value: they point out areas of the knowledge base that need to be examined if enhanced performance is to be achieved.

While we believe that we have demonstrated useful empirical refinement capabilities for expert classification systems, we recognize that this approach is not complete and that there are numerous avenues for further research. We briefly mention several areas where further work is warranted:

- generalization of problem-types covered by refinement. The current SEEK2 world is limited to production rules and classification systems.

- development of an extended knowledge base metalanguage. A prototype system has

[13]The incorporation of domain-specific metaknowledge in SEEK2 might enable the system itself to sometimes reject a refinement that in some way violates the expert's understanding of the domain, even though it may improve performance.

been developed [7] in which the current heuristics can be specified. With an appropriate description, the efficacy of the heuristics would be subject to experimentation. This would provide a means to evaluate the performance of heuristics and statistical assessment measures, and allow for comparative study of alternative policies regarding the conservatism vs. optimization trade-off. The language would also allow for the representation of domain-specific and alternative sources of knowledge.

- incorporation of domain-specific heuristics. This could include relatively simple forms of knowledge about rule sets that should never be modified, classes of cases that should never be misdiagnosed, or findings and rule components that should be modified first. More complicated knowledge about domain principles, such as cause and effect knowledge, could be used to verify knowledge base consistency.

- experimentation with the larger knowledge bases. While SEEK2 performs well on the cited knowledge base, many typical expert systems have much larger dimensions.[14]

- exploitation of parallelism in certain refinement operations. Larger knowledge base dimension significantly increase cpu-times.[15] This time may be cut dramatically by a parallel algorithm, for example each GDX could be pursued in parallel.

- elucidation of alternative strategies for validation and refinement. There are many alternative strategies that may be employed. These vary from strategies that consider ties, multiple conclusions, disagreement among experts. More complex search strategies may be specified that pursue multiple refinement paths. For example, when two refinements yield approximately the same improvement, we might want to postpone selecting just one until further derived refinements are pursued.

- inclusion of automatic learning heuristics. Some forms of learning can be viewed as an extension of refinement. Although SEEK2 has several specialization heuristics, SEEK2 never adds a component to a rule. Domain knowledge or empirical analysis could prove helpful in allowing for these types of rule modifications.

Validity and consistency are important goals in developing expert systems. Yet the design of these systems is often lacking in a coherent formal approach for achieving these goals. The approach to knowledge base refinement described here can lead to a more solid foundation for designing and validating expert classification system knowledge bases.

---

[14] A more recent version of the rheumatology knowledge base has 26 diagnoses, 400 intermediate hypotheses, 880 observations, and over 1000 rules.

[15] The more recent rheumatology knowledge base should take 2-3 days of cpu time for a 300 case data base.

Appendix A
SEEK2's Refinement Concepts and Heuristics

A.1. Refinement Concepts

Signif(rule) the number of cases in which PDX=CDX and this rule is the only rule that concludes CDX with confidence ≥ the model's confidence in the 2nd highest ranked conclusion. The set of cases counted by Signif(rule) is denoted by **Signif-Cases(rule)**.

Signif-level(rule,x)
the number of the Signif-Cases(rule) cases with the property that the 2nd highest confidence in the case is greater than x.

Gen(rule) the number of cases in which PDX ≠ CDX and this rule would correct the case if it had fired, and this rule is the closest to being satisfied of all rules that would correct the case. The set of cases counted by Gen(rule) is denoted by **Gen-Cases(rule)**.

SpecA(rule) the number of cases in which PDX ≠ CDX and if this rule had not fired the case would have been correct. The set of cases counted by SpecA(rule) is denoted by **SpecA-Cases(rule)**.

SpecB(rule) the number of cases in which PDX ≠ CDX and this rule is a member of a set of two or more rules concluding CDX such that if all the members of this set had failed to fire the case would have been correct. The set of cases counted by SpecB(rule) is denoted by **SpecB-Cases(rule)**.

Mfmc(rule) the most frequently missing (i.e., unsatisfied) component of this rule, relative to a set of cases in which the rule is unsatisfied, usually those cases in which this rule has been chosen as the candidate for generalization.

GenCF(rule) the number of cases in which PDX ≠ CDX and this rule concludes PDX and is satisfied and has confidence closest to the model's confidence in CDX of all the rules satisfied for PDX. The set of cases counted by GenCF(rule) is denoted by **GenCF-Cases(rule)**.

Mean-Cdx-CF[GenCF-Cases(rule)]
the mean value of the knowledge base's confidence in CDX in the cases denoted by GenCF-Cases(rule).

Mean-Pdx-CF[SpecA-Cases(rule) ∪ SpecB-Cases(rule)]
the mean value of the model's confidence in PDX in the cases denoted by (SpecA-Cases(rule) ∪ SpecB-Cases(rule)).

A.2. Generalization Heuristics

If: Gen(rule) > [SpecA(rule) + SpecB(rule)] &
Mfmc(rule) is equal to choice component C

Then: Decrease the choice-number of C in rule.

If: Gen(rule) > [SpecA(rule) + SpecB(rule)] &
Mfmc(rule) is some NON-CHOICE component C

Then: Delete this NON-CHOICE component in rule.

If: GenCF(rule) > [SpecA(rule) + SpecB(rule)]

Then: Raise the confidence level of the rule to
Mean-CDX-CF(GenCF-Cases(rule)) + .5 * (Standard-Deviation).

If: the NON-CHOICE component that has been suggested
to be deleted from the rule r1 is an INTERMEDIATE-HYPOTHESIS &
r2 is a rule that concludes the INTERMEDIATE-HYPOTHESIS
(at the indicated confidence range) &
r2 is closest to being satisfied in a plurality of the cases
in which r1 was chosen to be generalized

Then: Identify the most frequently missing component of r2 relative
to the cases in which r1 was chosen to be generalized;
if it is a CHOICE, lower the choice-number;
if it is a NON-CHOICE, delete it, or change its range if possible,
if it is an INTERMEDIATE HYPOTHESIS, apply this heuristic again.

If: the NON-CHOICE component that has been suggested to be
deleted from rule r1 is an INTERMEDIATE-HYPOTHESIS H
with associated confidence range (L:U) &
the majority of Gen-Cases(r1) in which H's confidence is not
in the range (L:U) are ones in which H's confidence factor is below L

Then: lower the value of L in the range (L:U) in r1 to
Mean-Value(confidence of H, {case | case is in Gen-Cases(r1) &
the confidence for H in case is less than L}).

If: the NON-CHOICE component that has been suggested to be deleted from rule r1 is an INTERMEDIATE-HYPOTHESIS H with associated confidence range (L:U) &
the majority of Gen-Cases(r1) in which H's confidence is not in the range (L:U) are ones in which H's confidence factor is above U

Then: raise the value of U in the range (L:U) in r1 to Mean-Value(confidence of H, {case | case is in Gen-Cases(r1) & the confidence for H in greater than L}).

If: the NON-CHOICE component that has been suggested to be deleted from rule r1 is an NUMERICAL-FINDING F with associated value range (L:U) &
the majority of Gen-Cases(r1) in which F's value is not in the range (L:U) are ones in which F's value is below L

Then: lower the value of L in the range (L:U) in r1 to Mean-Value(value of F, {case | case is in Gen-Cases(r1) & the value of F in case is less than L}).

If: the NON-CHOICE component that has been suggested to be deleted from rule r1 is a NUMERICAL-FINDING F with associated value range (L:U) &
the majority of Gen-Cases(r1) in which F's value is not in the range (L:U) are ones in which F's value is above U

Then: raise the value of U in the range (L:U) in r1 to Mean-Value(value of F, {case | case is in Gen-Cases(r1) & the value of F in case is greater than L}).

A.3. Specialization Heuristics

If: SpecA(rule) > Signif(rule) &
there is a CHOICE in rule

Then: Increase the choice-number of CHOICE in rule.

If: [SpecA(rule) + SpecB(rule)] >
Signif-level(rule,Mean-Pdx-CF(SpecA-Cases(rule) ∪ SpecB-Cases(rule)))

Then: lower the confidence level of the rule to
Mean-Pdx-CF(SpecA-Cases(rule) ∪ SpecB-Cases(rule)) - .5 * (Standard-Deviation).

If: SpecA(rule) > Signif(rule) > 0

Then: For every component C in rule with associated range (L:H) do the following:

calculate:
L′ = Mean-value(C,Signif-Cases(rule)) - 2 * (Standard-deviation)
H′ = Mean-value(C,Signif-Cases(rule)) + 2 * (Standard-deviation)

if L′>L, raise L to L′; if H′<H, lower H to H′.

References

[1] Clancey, W.
Heuristic Classification.
Artificial Intelligence 27:289-350, 1985.

[2] Clancy, W.
The Epistemology of a Rule-Based Expert System: A Framework for Explanation.
Artificial Intelligence 20(3):215-251, 1983.

[3] Davis, R.
Interactive Transfer of Expertise: Acquisition of New Inference Rules.
Artificial Intelligence 12:121-157, 1979.

[4] Doyle, R.
Constructing and Refining Causal Explanations from An Inconsistent Domain Theory.
In *Proceedings of the Fifth Annual National Conference on Artificial Intelligence*, pages 538-544.
Philadelphia, Pa., 1986.

[5] Duda, R., and Hart, P.
Pattern Classification and Scene Analysis.
Wiley, New York, 1973.

[6] Eshelman, L. and McDermott, J.
MOLE: A Knowledge Acquisition Tool That Uses Its Head.
In *Proceedings of the Fifth Annual National Conference on Artificial Intelligence*, pages 950-955.
Philadelphia, Pa., 1986.

[7] Ginsberg, A.
Refinement of Expert System Knowledge Bases: A Metalinguistic Framework for Heuristic Analysis.
PhD thesis, Department of Computer Science, Rutgers University, 1986.

[8] Ginsberg, A.
A Metalinguistic Approach to the Construction of Knowledge Base Refinement Systems.
In *Proceedings of the Fifth Annual National Conference on Artificial Intelligence.* Philadelphia,Pa., 1986.

[9] Hall, R.
Learning By Failing to Explain.
In *Proceedings of the Fifth Annual National Conference on Artificial Intelligence*, pages 568-572.
Philadelphia, Pa., 1986.

[10] Kahn, G., Nowlan, S., Mcdermott, J.
MORE: An Intelligent Knowledge Acquisition Tool.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 581-584.
Los Angeles, CA, 1985.

[11] Lee, W. and Ray, S.
Rule Refinement Using the Probabilistic Rule Generator.
In *Proceedings of the Fifth Annual National Conference on Artificial Intelligence*, pages 442-447.
Philadelphia, Pa., 1986.

[12] Michalski, Carbonell, Mitchell (editors).
Machine Learning.
Tioga Publishing Company, Palo Alto, 1983.

[13] Michalski, R., Mozetic, I., Hong, J., and Lavrac, N.
The Multi-purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains.
In *Proceedings of the Fifth Annual National Conference on Artificial Intelligence*, pages 1041-1045. Philadelphia, Pa., 1986.

[14] Mitchell, T.
Generalization as Search.
Artificial Intelligence 18:203-226, 1982.

[15] Mitchell, T., Keller, R., Kedar-Cabelli, S.
Explanation-Based Generalization: A Unifying View.
Machine Learning 1:47-80, 1986.

[16] Politakis, P.
Using Empirical Analysis to Refine Expert System Knowledge Bases.
PhD thesis, Department of Computer Science, Rutgers University, 1982.

[17] Politakis, P. and Weiss, S.
Using Empirical Analysis to Refine Expert System Knowledge Bases.
Artificial Intelligence 22:23-48, 1984.

[18] Smith, R., Winston, H., Mitchell, T., and Buchanan, B.
Representation and Use of Explicit Justification for Knowledge Base Refinement.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 673-680. Los Angeles, California, 1985.

[19] Waters, R.
KBEmacs: A Step Toward the Programmer's Apprentice.
IEEE Transactions on Software Engineering 11:1296-1320, 1985.

[20] Weiss, S. and Kulikowski, C.
A Practical Guide to Designing Expert Systems.
Rowman and Allanheld, Totowa, New Jersey, 1984.

[21] Wilkins, D. and Buchanan, B.
On Debugging Rule Sets When Reasoning Under Uncertainty.
In *Proceedings of the Fifth Annual National Conference on Artificial Intelligence*, pages 448-454. Philadelphia, Pa., 1986.

An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods

Sholom M. Weiss and Ioannis Kapouleas
Department of Computer Science, Rutgers University, New Brunswick, NJ 08903

Abstract

Classification methods from statistical pattern recognition, neural nets, and machine learning were applied to four real-world data sets. Each of these data sets has been previously analyzed and reported in the statistical, medical, or machine learning literature. The data sets are characterized by statistical uncertainty; there is no completely accurate solution to these problems. Training and testing or resampling techniques are used to estimate the true error rates of the classification methods. Detailed attention is given to the analysis of performance of the neural nets using back propagation. For these problems, which have relatively few hypotheses and features, the machine learning procedures for rule induction or tree induction clearly performed best.

1 Introduction

Many decision-making problems fall into the general category of classification [Clancey, 1985, Weiss and Kulikowski, 1984, James, 1985]. Diagnostic decision making is a typical example. Empirical learning techniques for classification span roughly two categories: statistical pattern recognition [Duda and Hart, 1973, Fukunaga, 1972] (including neural nets [McClelland and Rumelhart, 1988]) and machine learning techniques for induction of decision trees or production rules. While a method from either category is usually applicable to the same problem, the two categories of procedures can differ radically in their underlying models and the final format of their solution. Both approaches to (supervised) learning can be used to classify a sample pattern (example) into a specific class. However, a rule-based or decision tree approach offers a modularized, clearly explained format for a decision, and is compatible with a human's reasoning procedures and expert system knowledge bases.

Statistical pattern recognition is a relatively mature field. Pattern recognition methods have been studied for many years, and the theory is highly developed [Duda and Hart, 1973, Fukunaga, 1972]. In recent years, there has been a surge in interest in newer models of classification, specifically methods from machine learning and neural nets.

Methods of induction of decision trees from empirical data have been studied by researchers in both artificial intelligence and statistics. Quinlan's ID3 [Quinlan, 1986] and C4 [Quinlan, 1987a] procedures for induction of decision trees are well known in the machine learning

community. The Classification and Regression Trees (CART) [Breiman, Friedman, Olshen, and Stone, 1984] procedure is a major nonparametric classification technique that was developed by statisticians during the same period as ID3. Production rules are related to decision trees; each path in a decision tree can be considered a distinct production rule. Unlike decision trees, a disjunctive set of production rules need not be mutually exclusive. Among the principal techniques of induction of production rules from empirical data are Michalski's AQ15 system [Michalski, Mozetic, Hong, and Lavrac, 1986] and recent work by Quinlan in deriving production rules from a collection of decision trees [Quinlan, 1987b].

Neural net research activity has increased dramatically following many reports of successful classification using hidden units and the back propagation learning technique. This is an area where researchers are still exploring learning methods, and the theory is evolving.

Researchers from all these fields have all explored similar problems using different classification models. Occasionally, some classical discriminant methods are cited in comparison with results for a newer technique such as a comparison of neural nets with nearest neighbor techniques. In this paper, we report on results of an extensive comparison of classification methods on the same data sets. Because of the recent heightened interest in neural nets, and in particular the back propagation method, we present a more detailed analysis of the performance of this method. We selected problems that are typical of many applications that deal with uncertainty, for example medical applications. In such problems, such as determining who will survive cancer, there is no completely accurate answer. In addition, we may have a relatively small data set. An analysis of each of the data sets that we examined has been previously published in the literature.

2 Methods

We are given a data set consisting of patterns of features and correct classifications. This data set is assumed to be a random sample from some larger population, and the task is to classify new patterns correctly. The performance of each method is measured by its error rate. If unlimited cases for training and testing are available, the error rate can readily be obtained as the error rate on the test cases. Because we have far fewer cases, we must use resampling techniques for estimating error rates. These are described in the next section.[1]

2.1. Estimating Error Rates

It is well known that the *apparent* error rate of a classifier on all the training cases[2] can lead to highly misleading and usually over-optimistic estimates of performance [Duda and Hart, 1973].

---

[1]While there has been much recent interest in the "probably approximately correct" (PAC) theoretical analysis for both rule induction [Valiant, 1985, Haussler, 1988] and neural nets [Baum and Haussler, 1989], the PAC analysis is a worst case analysis to guarantee for *all* possible distributions that results on a training set are correct to within a small margin of error. For a real problem, one is given a sample from a single distribution, and the task is to estimate the true error rate. This type of analysis requires far fewer cases, because only a single albeit unknown distribution is considered and independent cases are used for testing.

[2]This is sometimes referred to as the resubstitution or reclassification error rate.

This is due to overspecialization of the classifier to the data.[3]

Techniques for estimating error rates have been widely studied in the statistics [Efron, 1982] and pattern recognition [Duda and Hart, 1973, Fukunaga, 1972] literature. The simplest technique for "honestly" estimating error rates, the holdout or H method, is a single train and test experiment. The sample cases are broken into two groups of cases: a training group and a test group. The classifier is independently derived from the training cases, and the error estimate is the performance of the classifier on the test cases. A single random partition of train and test cases can be somewhat misleading. The estimated size of the test sample needed for a 95% confidence interval is described in [Highleyman, 1962]. With 1000 independent test cases, one can be virtually certain that the error rate on the test cases is very close to the true error rate.

Instead of relying on a single train and test experiment, multiple random test and train experiments can be performed. For each random train and test partition, a new classifier is derived. The estimated error rate is the average of the error rates for classifiers derived for the *independently* and randomly generated partitions. Random resampling can produce better error estimates than a single train and test partition.

A special case of resampling is known as leaving-one-out [Fukunaga, 1972, Efron, 1982]. Leaving-One-Out is an elegant and straightforward technique for estimating classifier error rates. Because it is computationally expensive, it is often reserved for relatively small samples. For a given method and sample size n, a classifier is generated using n-1 cases and tested on the remaining case. This is repeated n times, each time designing a classifier by *leaving-one-out*. Each case is used as a test case and, each time nearly all the cases are used to design a classifier. The error rate is the number of errors on the single test cases divided by n.

Evidence for the superiority of the leaving-one-out approach is well-documented [Lachenbruch and Mickey, 1968, Efron, 1982]. While leaving-one-out is a preferred technique, with large samples it may be computationally expensive. However as the sample size grows, traditional train and test methods improve their accuracy in estimating error [Kanal and Chandrasekaran, 1971].

The leaving-one-out error technique is a special case of the general class of *cross validation* error estimation methods [Stone, 1974]. In k-fold cross validation, the cases are randomly divided into k mutually exclusive test partitions of approximately equal size. The cases not found in each test partition are independently used for training, and the resulting classifier is tested on the corresponding test partition. The average error rates over all k partitions is the cross-validated error rate. The CART procedure was extensively tested with varying numbers of partitions and 10-fold cross validation seemed to be adequate and accurate, particularly for large samples where

---

[3]In the extreme, a classifier can be constructed that simply consists of all patterns in the given sample. Assuming identical patterns do not belong to different classes, this yields perfect classification on the sample cases.

leaving-one-out is computationally expensive [Breiman, Friedman, Olshen, and Stone, 1984][4] For small samples, bootstrapping, a method for resampling with replacement, has shown much promise as a low variance estimator for classifiers [Efron, 1983, Jain, Dubes, and Chen, 1987, Crawford, 1989]. This is an area of active research in applied statistics.

Figure 1 compares the techniques of error estimation for a sample of n cases. The estimated error rate is the average of the error rates over the number of iterations. While these error estimation techniques were known and published in the 1960s and early 1970s, the increase in computational speeds of computers, makes them much more viable today for larger samples and more complex classification techniques [Steen, 1988].

	Holdout	Random Subsampling
Training cases	j	j
Testing cases	n-j	n-j
Iterations	1	B<<n

	Leaving-One-Out	10-fold CV
Training cases	n-1	90%
Testing cases	1	10%
Iterations	n	10

Figure 1: Comparison of Techniques for Estimating Error Rates

Besides improved error estimates, there are a number of significant advantages to resampling. The goal of separating a sample of cases into a training set and testing set is to help design a classifier with a minimum error rate. With a single train and test partition, too few cases in the training group can lead to the design of a poor classifier, while too few test cases can lead to erroneous error estimates. Leaving-One-Out, and to a lesser extent random resampling, allow for accurate estimates of error rates while training on most cases. For purposes of comparison of classifiers and methods, resampling provides an added advantage. Using the same data, researchers can readily duplicate analysis conditions and compare published error estimates with new results. Using only a single random train and test partition introduces the possibility of variability of partitions to explain the divergence from a published result.

[4]Empirical results also support the stratification of cases in the train and test sets to approximate the percentage (prevalence) of each class in the overall sample.

2.2. Classification Methods

In this section, the specific classification methods used in the comparison will be described. We do not review the methods or their mathematics, but rather state the conditions under which they were applied. References to all methods are readily available. Our goal is to apply each of these methods to the same data sets and report the results.

2.2.1. Statistical Pattern Recognition

Several classical pattern recognition methods were used. Figure 2 lists these methods. These methods are well-known and will not be discussed in detail. The reader is referred to [Duda and Hart, 1973] for further details. Instead, we give the specific variation of the method that we used.

Linear discriminant
Quadratic discriminant
Nearest Neighbor
Bayes independence
Bayes second order

Figure 2: Statistical Pattern Recognition Methods

The linear and quadratic discriminants are the standard multivariate normal discriminants. The linear discriminant simplifies the normality assumption to equal covariance matrices. This is probably the most commonly used form of discriminant analysis; we used the canned SAS and IMSL programs. A recent report has demonstrated improved results in game playing evaluation functions using the quadratic classifier [Lee, 1988].

We used the nearest neighbor method (k=1) with the Euclidean distance metric. This is one of the simplest methods conceptually, and is commonly cited as a basis of comparison with other methods. It is often used in case-based reasoning [Waltz, 1986].

Bayes rule is the optimal presentation of minimum error classification. All classification methods can be viewed as approximations to Bayes optimal classifiers. Because the Bayes optimal classifier requires complete probability data for all dependencies in its invocation, for real problems this would be impossible. As with other methods, simplifying assumptions are made. The usual simplification is to assume conditional independence of observations. While one can point to dozens of classifiers that have been built (particularly in medical applications [Szolovits and Pauker, 1978]) using Bayes rule with independence, such approaches have also been recently reported in the AI literature (although in the context of unsupervised learning) [Cheeseman, 1988]. Although independence is commonly assumed, there are mathematical expansions to incorporate higher order correlations among the observations. In our experiments, we tried both Bayes with independence and Bayes with the second order Bahadur expansion.[5]

[5]Continuous variables were broken into 10 (binary) intervals with width of half a standard deviation from the mean.

2.2.2. Neural Nets

A fully connected neural net with a single hidden layer was considered. The back propagation procedure [McClelland and Rumelhart, 1988] was employed and the general outline of the data analysis described in [Gorman and Sejnowski, 1988] was followed. The specific implementation used was [McClelland and Rumelhart, 1988].[6] In most experiments a learning rate of 1 and a momentum of 0 was used.[7] Patterns were presented randomly to the learning system.[8].

The analysis model of [Gorman and Sejnowski, 1988] corresponds to a 10-fold cross validation. Unlike the other methods examined in this study, back propagation usually commences with the network weights in a random state. Thus, even with sequential presentation of cases, the weights for one learned network are unlikely to match the same network that starts in a different random state. There is also the possibility of the procedure reaching a local maximum. In this analysis model, for each train and test experiment, the weights are learned 10 times, and test results averaged over all 10 experiments. Therefore, 10 times the usual number of training trials must be considered. For a 10-fold cross-validation, 100 learning experiments are made.

For each data set, these experiments were repeated for networks having 0,2,3,6,9,12, or 24 hidden units (in a single layer). This is equivalent to using resampling to estimate the appropriate number of hidden units. Because the data sets may not be separable with these numbers of hidden units, we took the following measures to determine a sufficient amount of computation time. Before doing the train and test experiments, the nets were trained several times on all samples for all size hidden units. We determined a number of *epochs*, i.e. complete presentations of the data set, that was sufficient to result in each increment of additional hidden units fitting the cases better than the lesser number of hidden units. In addition, for one problem where the data set was extremely large, we sampled the results every 500 epochs, and computed whether the average total squared error continued to be reduced. This indicated whether progress was being made.

One output unit was used for each class. The hypothesis with the highest weight was selected as the conclusion of the classifier, and the error rate was computed.

This is the general outline of the procedures followed. In Section 3, we describe the variations on this theme that were necessary for the specific data set analyses.

For computational reasons, in some instances it was necessary to reduce the number of repeated trials to be averaged. For back propagation, we described a computational procedure that performed 10 train and test experiments for each one that would be necessary for other methods.

---

[6]The program was readily ported to a Sun 4.

[7]These two parameters were changed from the program defaults because it was observed that the program converged towards a solution much faster, and no problems were encountered with local maximums.

[8]For the studied data sets, sequential presentation tended to lead rather quickly to a local maximum.

However, the data sets described in Section 3 are not readily separable. Thus, the computation demands are quite large. We estimate that 6 months of Sun 4/280 cpu time were expended to compute the neural nets results in Section 3.

2.2.3. Machine Learning Methods

In this category, we place methods that produce logistic solutions. As indicated earlier these methods have been explored by both the machine learning and statistics community. These are methods that produce solutions posed as production rules or decision trees. Conjunction or disjunction may be used as well as logical comparison operators on continuous variables such as greater than or less than.

Predictive Value Maximization [Weiss, Galen, and Tadepalli, 1987] was tried on all data sets. This is a heuristic search procedure that attempts to find the best *single* rule in disjunctive normal form. It can be viewed as a heuristic approximation to exhaustive search. It is applicable to problems where a relatively short rule provides a good solution. For such problems, it should have an advantage in that many combinations are considered, in contrast to current decision tree procedures that split nodes without considering combinations. For more complex problems, a decision tree procedure is preferable. The appropriate rule length or tree size is determined by resampling.

In addition, for two of the smaller data sets, an exhaustive search was performed for the optimal rule of length 2 in disjunctive normal form. For the other 2 data sets, the published decision tree results are available for methods using variations of ID3 and its successor C4.

3 Results

In this section, we review the results of the various classification methods on four data sets. All of the data sets have been published, and in most instances we attempted to perform the analyses in a manner consistent with previously known results.

3.1. Iris Data

The iris data was used by Fisher in his derivation of the linear discriminant function [Fisher, 1936], and it still is the standard discriminant analysis example used in most current statistical routines such as SAS or IMSL. Linear or quadratic discriminants under assumptions of normality perform extremely well on this data set. Three classes of iris are discriminated using 4 continuous features. The data set consists of 150 cases, 50 for each class. Figure 3 summarizes the results. The first error rate is the apparent error rate on all cases; the second error rate is the leaving-out-one error rate. Leaving-one-out results have been previously widely disseminated for several of the statistical pattern recognition methods.

The rule-based solution has 2 rules with a total of 3 variables.[9] For the neural nets, the apparent error rate is the average of five trials. The leaving-one-out result is the average of 5 complete leaving-one-out trials. The nets were trained for 1000 epochs. The best neural net in terms of

[9]The optimal rule is also induced by PVM during cross-validation.

Method	Err_{App}	Err_{Cv}
Linear	.020	.020
Quadratic	.020	.027
Nearest neighbor	.000	.040
Bayes independence	.047	.067
Bayes 2nd order	.040	.160
Neural net (BP)	.017	.033
PVM rule	.027	.040
Optimal rule size 2	.020	.020
CART tree	.040	.047

Figure 3: Comparative Performance on Fisher's Iris Data

cross-validated error occurs at 3 hidden units, and is the one listed in Figure 3. The relationship between the number of hidden units and the error rates is listed in Figure 4.

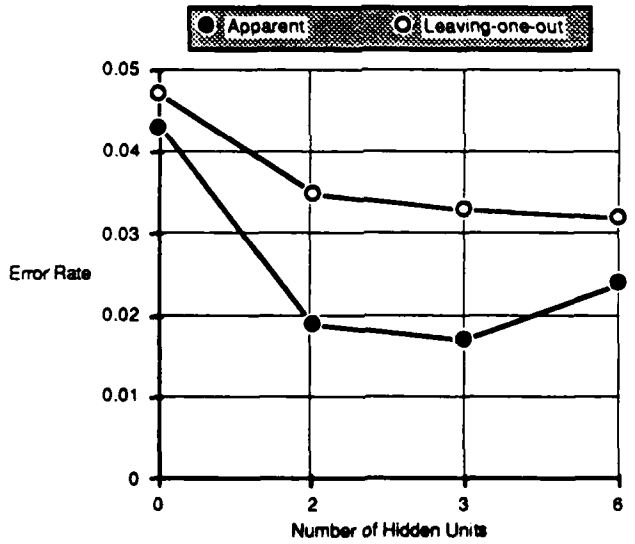


Figure 4: Neural Net Error Rates for Iris Data

3.2. Appendicitis Data

This data set is from a published study on the assessment of 8 laboratory tests to confirm the diagnosis of appendicitis [Marchand, Van Lente, and Galen, 1983].[10] Following surgery, only 85 of 106 patients were confirmed by biopsy to have had appendicitis. Thus, the ability to discriminate the true appendicitis patients by lab tests prior to surgery would prove extremely valuable.

[10]These are patients admitted to an emergency room with a tentative diagnosis of acute appendicitis.

The samples consist of 106 patients and 8 diagnostic tests. Because one test had some missing values, for purposes of comparison, we excluded results from that test. Figure 5 summarizes the results. The first error rate is the apparent error rate on all cases; the second error rate is the leaving-out-one error rate

Method	Err_{App}	Err_{Cv}
Linear	.113	.132
Quadratic	.217	.264
Nearest neighbor	.000	.179
Bayes independence	.113	.170
Bayes 2nd order	.047	.189
Neural net (BP)	.098	.142
PVM rule	.085	.104
Optimal rule size 2	.085	.104
CART tree	.094	.151

Figure 5: Comparative Performance on Appendicitis Data

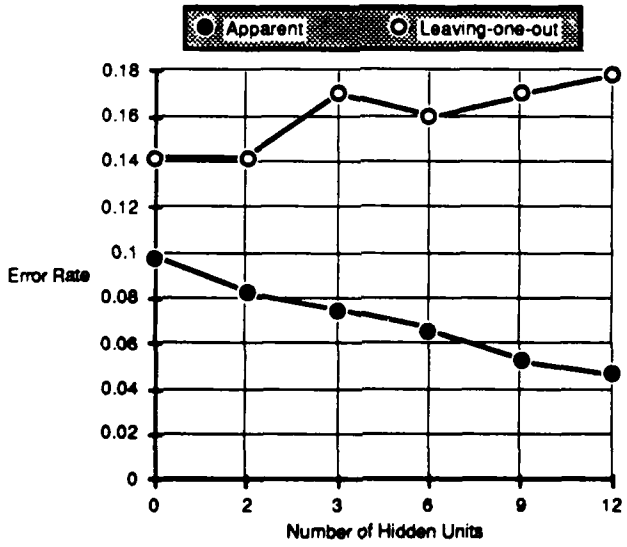


Figure 6: Neural Net Error Rates for Appendicitis Data

The rule-based solution has 1 rule with a total of 2 variables. For the neural nets, the apparent error rate is the average of five trials. The leaving-one-out result is for a single leaving-one-out trial.[11] The nets were trained for 15000 epochs. The best neural net in terms of cross-validated

[11]The results for the average of 5 complete leaving-one-out trials is available for 1000 epochs. These show poorer performance, but 100 epochs were not sufficient for training the larger number of hidden units.

error occurs at 0 hidden units, and is the one listed in Figure 5. The relationship between the number of hidden units and the error rates is listed in Figure 6.

3.3. Cancer Data

A data set for evaluating the prognosis of breast cancer recurrence was analyzed by Michalski's AQ15 rule induction program and reported in [Michalski, Mozetic, Hong, and Lavrac, 1986]. They reported a 64% accuracy rate for expert physicians, and a 68% rate for AQ15, and a 72% rate for the pruned tree procedure of ASSISTANT [Kononenko, Bratko, and Roskar, 1986], a descendant of ID3.[12] The authors derived the error rates by randomly resampling 4 times using a 70% train and a 30% test partition.

The samples consist of 286 samples, 9 tests, and 2 classes. We created 4 randomly sampled data sets with 70% train and a 30% test partitions; each method was tried on each of the four data sets and the results averaged. Thus, the experimental results are consistent with the original study. Figure 7 summarizes the results. The first error rate is the apparent error rate on the training cases; the second error rate is the error rate on the test cases.

Method	Err_{Train}	Err_{Test}
Linear	.254	.294
Quadratic	.245	.344
Nearest neighbor	.000	.347
Bayes independence	.241	.282
Bayes 2nd order	.091	.344
Neural net (BP)	.243	.285
PVM rule	.226	.229
ASSISTANT tree	-	.280
CART tree	.226	.229

Figure 7: Comparative Performance on Cancer Data

The rule-based solution has 1 rule with a total of 2 variables.[13] For the neural nets, the apparent error rate is the average of ten training trials. Each testing result is the corresponding average testing result of the same 10 complete trials.[14] The nets were trained for 2000 epochs. The best neural net in terms of cross-validated error occurs at 0 hidden units, and is the one listed in Figure 7. The relationship between the number of hidden units and the error rates is listed in Figure 8.

---

[12]The prevalence of the larger class is 70%.

[13]The same rule was induced on all four 70% training sets.

[14]Also considered was the best of the 10 training results and its corresponding test result. These results are within 1% of the average results.

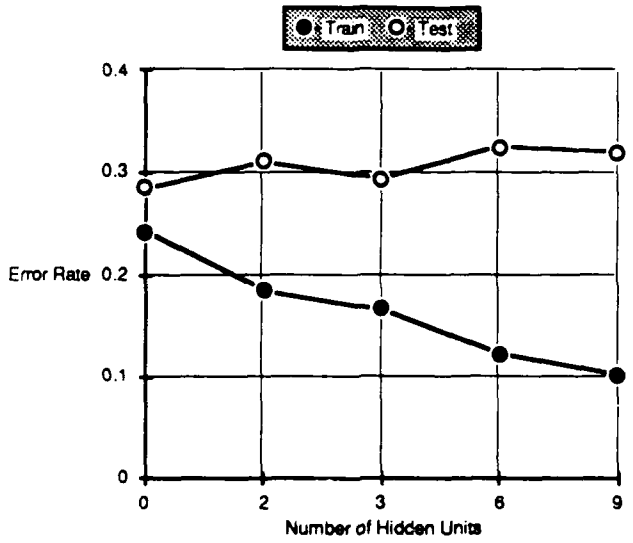


Figure 8: Neural Net Error Rates for Cancer Data

3.4. Thyroid Data

Quinlan reported on results of his analysis of hypothyroid data in [Quinlan, 1987b], and in greater detail in [Quinlan, 1987a]. The problem is to determine whether a patient referred to the clinic is hypothyroid, the most common thyroid problem. In contrast to the previous applications, relatively large numbers of samples are available.

The samples consist of 3772 cases from the year 1985. These are the same cases used in the original report and were used for training. The 3428 cases from 1986 were used as test cases. There are 22 (principal) tests, and 3 classes. Over 10% of the values are missing because some lab tests were deemed unnecessary. For purposes of comparison of the methods, these values were filled in with the mean value for the corresponding class.

Figure 9 summarizes the results.[15] The first error rate is the error rate on the 3772 training cases; the second error rate is the error rate on the 3428 test cases. From a medical perspective, it is known that (based on lab tests) excellent classification can be achieved for diagnosing thyroid dysfunction. For these data, the correct answer stored with each sample is derived from a large rule-based system in use in Australia. While most error rates in Figure 9 are low, it is important to note that 1% of the total sample represents over 70 people. Over 92% of the samples are not hypothyroid. Therefore, any acceptable classifier must do significantly better than 92%.

---

[15]The C4 tree cited in the original study has a training error rate of .0021 and a testing error rate of .0085. However, the training data contained missing values.

Method	Err_{Train}	Err_{Test}
Linear	.0615	.0615
Quadratic	.1031	.1161
Nearest neighbor	0	.0473
Bayes independence	.0297	.0394
Bayes 2nd order	.0228	.0756
Neural net (BP)	.0050	.0146
PVM rule	.0021	.0067
CART tree	.0021	.0064

Figure 9: Comparative Performance on Thyroid Data

The rule-based solution has 2 rules with a total of 8 variables. For the neural nets, the apparent error rate is the best of 2 trials. The nets were trained for 2000 epochs. The best neural net in terms of testing error occurs at 3 hidden units. The relationship between the number of hidden units and the error rates is listed in Figure 10.

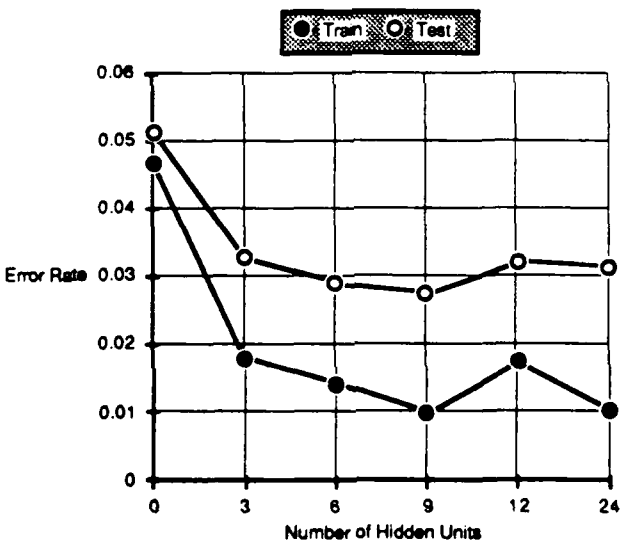


Figure 10: Neural Net Error Rates for Thyroid Data

The cpu times for training a neural net with back propagation on this size data set were great: for 3 hidden units 500 epochs required 1.5 hours of Sun 4/280 cpu time, while 24 units required 11.5 hours. In Figure 10, the apparent error rates for the larger numbers of hidden units support the hypothesis that additional training was necessary. We initiated a new set of experiments with

fewer numbers of hidden units.[16] We let these trials run for an unlimited period of time as long as slight progress was being made, as indicated by sampling every 500 epochs. Therefore, for this experiment not every size neural net was run an equal number of epochs. Figure 11 summarizes the results of this effort. The best result encountered during the sampling of results occurred for 3 hidden units, and this result is listed in Figure 9.

Units	Epochs	Err_{Train}	Err_{Test}
0	6000	.0260	.0359
3	70000	.0050	.0146
6	45000	.0037	.0163
9	24000	.0040	.0193

Figure 11: Extended Neural Network Training on Thyroid Data

4 Discussion

The applications presented here represent a reasonable cross section of prototypical problems widely encountered in the many research communities. Each problem has few classes and is characterized by uncertainty of classification. In some applications such as the cancer data, the features were relatively weak and good predictive capabilities are unlikely. In others, such as the thyroid data, the features are quite strong, and almost error-free prediction is possible.

For the smaller data sets, resampling was used. With over 100 cases, resampling techniques such as cross-validation should give excellent estimates for the true error rate. In fact, the data from the iris study has been reviewed over many years, and comparisons have been made on the basis of the leaving-one-out error. It is interesting to note (for those who wish to avoid concepts such multivariate distributions and covariance matrices), that a trivial set of 2 rules with a total of 3 variables can produce equal results.

For many application fields, this in fact is a major advantage of the logistic approaches, i.e. the rule based or decision tree based approaches. The solution is compatible with elementary human reasoning and explanations. It is also compatible with rule-based systems. Thus, if everything were equal, many would choose the logistic solution.

In our experiments, everything was not equal. In every case a logistic solution was found that exceeded the performance of solutions posed using different underlying models. PVM has an advantage when a short rule works, but for more complex problems the decision tree would be indicated. We note that the largest problem studied, the thyroid application, is somewhat biased towards logistic solutions. The endpoints were derived from a rule-based system that apparently uses the same lab test thresholds to specify high or low readings for all hypotheses.

---

[16] The momentum was changed to .9, and the learning rate to .5, to help prevent local maximums.

These results cannot necessarily be extrapolated to more complex problems. However, our experience is not unique. Numerous experiments by the developers of CART [Breiman, Friedman, Olshen, and Stone, 1984] demonstrated that in most instances, they found a tree superior to alternative statistical classification techniques.

In our experiments, the statistical classifiers performed consistently with expectations. The linear classifiers (with the assumption of a normal distribution) gave good performance in all cases except the thyroid experiment. These classifiers are widely used, because they are simple and the training error rate usually holds up well on test cases. The natural extension, the quadratic classifier, fits better to normally distributed data, but degrades rapidly with nonnormal data. It did poorly in most of our experiments. Similarly Bayes with independence does moderately well, but the 2nd order fits were not good on the test data. Nearest neighbor does well with good features, but tends to degrade with many poor features. There are many alternative statistical classifiers that might be tried, such nonparametric piecewise linear classifiers [Foroutan and Sklansky, 1985]. In addition, one could try to reduce the number of features for training (i.e. feature selection), since many of these methods can actually improve performance on test cases by feature reduction.[17]

The neural nets did perform well, and they were the only statistical classifiers to do well on the thyroid problem. However, overall they were not the best classifiers; they consumed enormous amounts of cpu time; and they were sometimes equaled by simple classifiers. Research on improving performance for neural nets training and representation is quite active, so it may be possible that performance can be improved.

The relationship between the number of hidden units and the two error rates followed the classical pattern for classifiers. As the number of hidden units increased, the apparent error decreased.[18] However, at some point, as the classifier overfits the data, the true error rate curve flattens and even begins to increase. Much the same behavior can be observed for decision trees as the number of nodes increases, or production rules, as the rule length increases.

The question remains open as to how well any classifier can do on more complex problems with many more features and many more classes, possibly non-mutually exclusive classes. There are also questions of how many cases are actually needed to learn significant concepts. Our study does not answer many of these questions, but helps show in a limited fashion where we are currently with many commonly used classification techniques.

---

[17]Because the linear classifier performed poorly on the thyroid cases, we tried to train a classifier on just the lab tests, which are the most significant tests. The results did not improve.

[18]Occasionally there is some slight variability in the decrease of the apparent error rate because back propagation minimizes distance as opposed to errors.

Appendix: Induced Rules

- iris. Petal length < 3 → Iris Setosa; Petal length > 4.9 OR Petal Width > 1.6 → Iris Virginica
- appendicitis. MNEA>6600 OR MBAP>11
- cancer. Involved Nodes>0 & Degree=3
- thyroid. TSH>6.1 & FTI <65 → primary hypothyroid; TSH>6 & TT4<149 & On Thyroxin=false & FTI>64 & Surgery=false → compensated hypothyroid

References

[Baum and Haussler, 1989]
Baum, E. and Haussler, D. "What Size Net Gives Valid Generalization?" *Neural Computation.* (1989) 151-160.

[Breiman, Friedman, Olshen, and Stone, 1984]
Breiman, L., Friedman, J., Olshen, R., and Stone, C. *Classification and Regression Tress.* Monterrey, Ca.: Wadsworth, 1984.

[Cheeseman, 1988]
Cheeseman P., Self, M., Kelly J., Stutz J., Taylor W., and Freeman D. "Bayesian Classification." In *Proceedings of AAAI-88.* Minneapolis, 1988, 607-611.

[Clancey, 1985]
Clancey, W. "Heuristic Classification." *Artificial Intelligence.* 27 (1985) 289-350.

[Crawford, 1989]
Crawford, S. "Extensions to the CART Algorithm." *International Journal of Man-Machine Studies.* (1989) in press.

[Duda and Hart, 1973]
Duda, R., and Hart, P. *Pattern Classification and Scene Analysis.* New York: Wiley, 1973.

[Efron, 1982]
Efron, B. "The Jackknife, the Bootstrap and Other Resampling Plans." In *SIAM.* Philadelphia, Pa., 1982.

[Efron, 1983]
Efron, B. "Estimating the Error Rate of a Prediction Rule." *Journal of the American Statistical Association.* 78 (1983) 316-333.

[Fisher, 1936]
Fisher, R. "The Use of Multiple Measurements in Taxonomic Problems." *Ann. Eugenics.* 7 (1936) 179-188.

[Foroutan and Sklansky, 1985]
Foroutan, I. and Sklansky, J. "Feature Selection for Piecewise Linear Classifiers." In *IEEE Proc. on Computer Vision and Pattern Recognition.* San Franscisco, 1985, 149-154.

[Fukunaga, 1972]
Fukunaga, K. *Introduction to Statistical Pattern Recognition.* New York: Academic Press, 1972.

[Gorman and Sejnowski, 1988]
Gorman R. and Sejnowski T. "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets." *Neural Networks.* 1 (1988) 75-89.

[Haussler, 1988]
Haussler, D. "Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework." *Artificial Intelligence.* 36 (1988) 177-221.

[Highleyman, 1962]
Highleyman, W. "The Design and Analysis of Pattern Recognition Experiments." *Bell System Technical Journal.* 41 (1962) 723-744.

[Jain, Dubes, and Chen, 1987]
Jain, A., Dubes, R., and Chen, C. "Bootstrap Techniques for Error Estimation." *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 9 (1987) 628-633.

[James, 1985]
James, M. *Classification Algorithms.* New York: John Wiley & Sons, 1985.

[Kanal and Chandrasekaran, 1971]
Kanal, L. and Chandrasekaran, B. "On Dimensionality and Sample Size In Statistical Pattern Classification." *Pattern Recognition.* (1971) 225-234.

[Kononenko, Bratko, and Roskar, 1986]
Kononenko, I., Bratko, I., Roskar, E. "ASSISTANT: A System for Inductive Learning." *Informatica.* 10 (1986) .

[Lachenbruch and Mickey, 1968]
Lachenbruch, P. and Mickey, M. "Estimation of Error Rates in Discriminant Analysis." *Technometrics.* (1968) 1-111.

[Lee, 1988]
Lee K. and Mahajan S. "A Pattern Classification Approach to Evaluation Function Learning." *Artificial Intelligence.* 36 (1988) 1-25.

[Marchand, Van Lente, and Galen, 1983]
Marchand, A., Van Lente, F., and Galen, R. "The Assessment of Laboratory Tests in the Diagnosis of Acute Appendicitis." *American Journal of Clinical Pathology.* 80:3 (1983) 369-374.

[McClelland and Rumelhart, 1988]
McClelland, J. and Rumelhart, D. *Explorations in Parallel Distributed Processing.* Cambridge, Ma.: MIT Press, 1988.

[Michalski, Mozetic, Hong, and Lavrac, 1986]
Michalski, R., Mozetic, I., Hong, J., and Lavrac, N. "The Multi-purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains." In *Proceedings of the Fifth Annual National Conference on Artificial Intelligence.* Philadelphia, Pa., 1986, 1041-1045.

[Quinlan, 1986]
Quinlan, J. "Induction of Decision Trees." *Machine Learning.* 1 (1986) 1.

[Quinlan, 1987a]
Quinlan, J. "Simplifying Decision Trees." *International Journal of Man-Machine Studies.* 27 (1987) 221-234.

[Quinlan, 1987b]
Quinlan, J. "Generating Production Rules from Decision Trees." In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence.* Milan, Italy, 1987, 304-307.

[Steen, 1988]
Steen, L. "The Science of Patterns." *Science*. 240 (1988) 611-616.

[Stone, 1974]
Stone, M. "Cross-Validatory Choice and Assessment of Statistical Predictions." *Journal of the Royal Statistical Society*. 36 (1974) 111-147.

[Szolovits and Pauker, 1978]
Szolovits, P., and Pauker, S. "Categorical and Probabilistic Reasoning in Medical Diagnosis." *Artificial Intelligence*. 11 (1978) 115-144.

[Valiant, 1985]
Valiant, L.G. "Learning disjunctions of conjunctions." In *Proceedings of IJCAI-85*. Los Angeles, 1985, 560-566.

[Waltz, 1986]
Stanfill G. and Waltz D. "Toward Memory-Based Reasoning." *Communications of the ACM*. 29 (1986) 1213-1228.

[Weiss and Kulikowski, 1984]
Weiss, S. and Kulikowski, C. *A Practical Guide to Designing Expert Systems*. Totowa, New Jersey: Rowman and Allanheld, 1984.

[Weiss, Galen, and Tadepalli, 1987]
Weiss, S., Galen, R., and Tadepalli, P. "Optimizing the Predictive Value of Diagnostic Decision Rules." In *Proceedings of the Sixth Annual National Conference on Artificial Intelligence*. Seattle, Washington, 1987, 521-526.

Maximizing the Predictive Value of Production Rules

Sholom M. Weiss*, Robert S. Galen**, and Prasad V. Tadepalli*

*Department of Computer Science, Rutgers University, New Brunswick, NJ
**Department of Epidemiology and Biostatistics, Case Western Reserve University, Cleveland, OH

Abstract

A new method for empirical rule induction under conditions of uncertainty is described. The problem is to find the single best production rule of a fixed length for classification. Predictive Value Maximization (PVM), a heuristic search procedure through the hypothesis space of conjunctions and disjunctions of variables and their cutoff values, is outlined. Examples are taken from laboratory medicine, where the goal is to find the best combination of tests for making a diagnosis. Resampling techniques for estimating error rates are integrated into the PVM procedure for rule induction. Excellent results for PVM are reported on data sets previously analyzed in the AI literature using alternative classification techniques.

1. Introduction

Many decision-making problems fall into the general category of classification [Clancey, 1985, Weiss and Kulikowski, 1984, James, 1985]. Diagnostic decision making is a typical example. Empirical learning techniques for classification span roughly two categories: statistical pattern recognition [Duda and Hart, 1973, Fukunaga, 1972] (including neural nets [McClelland and Rumelhart, 1988]) and machine learning techniques for induction of decision trees or production rules. While a method from either category is usually applicable to the same problem, the two categories of procedures can differ radically in their underlying models and the final format of their solution. Both approaches can be used to classify a sample pattern (example) into a specific class. However, a rule-based or decision tree approach offers a modularized, clearly explained format for a decision, and is compatible with a human's reasoning procedures and expert system knowledge bases.

Methods of induction of decision trees from empirical data have been studied by researchers in both artificial intelligence and statistics. Quinlan's ID3 [Quinlan, 1986] and C4 [Quinlan, 1987a] procedures for induction of decision trees are well-known in the machine learning community. The Classification and Regression Trees (CART) [Breiman, Friedman, Olshen, and Stone, 1984] procedure is a major nonparametric classification technique that was developed by statisticians during the same period as ID3. These procedures developed for decision tree induction are quite similar. The major distinction between CART and ID3/C4 is that the CART

procedure uses resampling techniques for both accurate error rate estimation and tree pruning [Stone, 1974]. Empirical comparisons of CART-derived decision trees with traditional statistical discriminant analysis has shown that the decision trees are very competitive in finding a minimum error solution. In almost all instances studied, the induced decision trees were as good or better than traditional statistical methods [Breiman, Friedman, Olshen, and Stone, 1984].

Production rules are related to decision trees; each path in a decision tree can be considered a distinct production rule. Unlike decision trees, a disjunctive set of production rules need not be mutually exclusive. Among the principal techniques of induction of production rules from empirical data are Michalski's AQ15 system [Michalski, Mozetic, Hong, and Lavrac, 1986] and recent work by Quinlan in deriving production rules from a collection of decision trees [Quinlan, 1987b]. Additional examples of rule induction systems can be found in [Fu and Buchanan, 1985, Spackman, 1988].

Machine learning techniques for induction of decision rules have evolved from procedures that cover all cases in a data set to more sophisticated and less biased procedures for estimating error rates by train and test sampling. Procedures that prune a set of decision rules and the components of these rules have been successful in increasing the performance of an induced rule set on new unseen test cases [Michalski, Mozetic, Hong, and Lavrac, 1986, Quinlan, 1987a]. Empirical results reported in the literature indicate that often a relatively short rule may provide a better solution than a more complex set of induced rules [Michalski, Mozetic, Hong, and Lavrac, 1986].

In this paper, we describe Predictive Value Maximization (PVM), a heuristic procedure for learning the *single* best decision rule of a fixed length. In contrast to the decision tree induction techniques, a commitment is not made to split a single test node at a time. Instead, this method is a heuristic approximation to exhaustive generation of all possible rules of a fixed length. While an exhaustive search is not feasible in most applications, a small number of heuristics reduce the search space to manageable proportions.

In Section 2, a detailed description of the underlying model is given. The complexity of exhaustive search is presented in Section 3. The PVM procedure is described in Section 4. In section 5, the concept of resampling and unbiased error rate estimation is introduced. In a fashion similar to CART procedure for deciding appropriately sized trees, the PVM procedure is modified to use resampling for finding the appropriate length rule. In Section 6.1 two data sets are analyzed, and the results of PVM are compared with the optimal production rule solution and with several statistical pattern recognition solutions. A comparison of results for two other data sets reported in the machine learning literature is given in Section 6.2.

2. The Model of Induction

In our discussion, examples from laboratory medicine will be used. However, the solution is general and should be applicable to many areas outside medicine. Let us assume that we are developing a new diagnostic test whose measurement yields a numerical result in a continuous range. For a single test, the problem is to select a cutoff point, known formally as a *referent value*, that will lead to satisfactory decisions. For example, a physician may conclude that all patients having a result greater than a specific cutoff have the disease, while others do not. For a specific cutoff, there are four possible outcomes for each test case in the sample.[1] These are illustrated in Figure 2-1. There are well-known measures to describe the performance of a test at a specific cutoff for a sample population. These measures, defined in Figure 2-1, are *sensitivity, specificity, positive predictive value, negative predictive value, and accuracy* [Galen and Gambino, 1975].

	Rule Positive (R+)	**Rule Negative** (R-)
Hypothesis Positive (H+)	True Positives (TP)	False Negatives (FN)
Hypothesis Negative (H-)	False Positives (FP)	True Negatives (TN)

Sensitivity	TP / H+
Specificity	TN / H-
Predictive value (+)	TP / R+
Predictive value (-)	TN / R-
Accuracy	(TP+TN) / ((H+) + (H-))

Figure 2-1: Formal Measures of Classification Performance

While all of these measures have their purpose, the one that is implicitly used in large-scale rule-based systems is positive predictive value. Positive predictive value measures how often a decision is correct when a test result is positive. Thus one may use a positive test that has high predictive value in rules that confirm a diagnosis, and apply different tests when the result is negative. Many rule-based systems may be thought of as collections of rules with very highly positive predictive values. The two types of errors, false positives and false negatives, need not be weighted equally. For example, in medical applications it is often required that the sensitivity be high, i.e. few false negatives, with perhaps more false positives.

We illustrate these points by describing data taken from a published study on the assessment of 8 laboratory tests to confirm the diagnosis of acute appendicitis for patients admitted to an emergency room with a tentative diagnosis of acute appendicitis [Marchand, Van Lente, and Galen, 1983]. Following surgery, only 85 of 106 patients were confirmed by biopsy to have had

[1] For purposes of this discussion, we are eliminating the possibility of unknowns.

appendicitis. Thus, the ability to discriminate the true appendicitis patients by labs tests prior to surgery would prove extremely valuable. Because acute appendicitis is life-threatening, it is imperative to avoid false negatives, i.e. patients with appendicitis who do not receive surgery. In the example of Figure 2-2, the white blood cell count (WBC) is used as a test to determine the true appendicitis patients.

	T+	T-
H+	71	14
H-	6	15

Sensitivity	83.5%
Specificity	71.4%
Predictive value (+)	92.2%
Predictive value (-)	51.7%
Accuracy	81.1%

Figure 2-2: Example of the 5 Measures of Performance for WBC>10000

In summary, for a single test with a given cutoff and the application of an arithmetic operator (less than or greater than) these five measures can be determined for a population. The problem of determining an optimal cutoff can be described as maximizing one of these measures subject to specific constraints on the other measures.[2] Constraints are the minimum required values for sensitivity, specificity, predictive values, and accuracy. Finding the optimum cutoff for WBC can be posed in the form illustrated in Figure 2-3. Not every combination of these performance measures yields different results. In the appendicitis example, 100% sensitivity is required, i.e. no false negatives. This is equivalent to requiring 100% negative predictive value.

MAXIMIZING Predictive value (+) of WBC

The constraints are given below:

Sensitivity	≥ 100.00%
Specificity	≥ 0.00%
Predictive value (-)	≥ 0.00%
Accuracy	≥ 0.00%

Figure 2-3: Example of Problem Constraints for a Single Test

Referent value analysis, or cutoff selection, is commonly done for single tests. We have

[2]Sensitivity and specificity move continuously in opposite directions. For example, a 100% sensitivity cutoff with 0% specificity can always be found by classifying every sample as having the hypothesis. Predictive values have no such relationship and vary greatly.

developed procedures that allow for the possibility of choosing the set of constraints and maximizing the remaining measure not only for one or two, but for a larger number of tests. When more than one test is specified, combinations are formed by using logical AND or OR operators. We formulate the problem as finding the *best* combination of tests that will satisfy the given constraints for the data set.[3] An additional constraint is added to the problem, in that the length of the expression is limited by a chosen threshold. This sets a limit on the number of terms that may be used in the decision rule. Some tests may be also deliberately excluded from consideration and some tests may be designated as mandatory. This allows for further pruning of the search space. In Figure 2-4 using the appendicitis data set, the problem is to find the best solution in the form of a logical expression whose length is no greater than 3 tests.

MAXIMIZING Predictive value (+)

The constraints are given below:

Sensitivity	≥ 100.00%
Specificity	≥ 0.00%
Predictive value (-)	≥ 0.00%
Accuracy	≥ 0.00%
Number of terms	≤ 3

Figure 2-4: Example of Problem Constraints for 3 or Fewer Tests

At this point we note that the rules are just like many found in typical classification expert systems, since, like productions, they are described as logical combinations of findings that are not mutually exclusive.[4] Thus, they have the intuitive appeal of explaining decisions in a format consistent with human reasoning, while being supported empirically by their performance over the data set. These rules classify under conditions of uncertainty, where two types of classification errors, false positives and false negatives, need not be considered of equal importance.

3. Complexity of Exhaustive Generation of Expressions

In Section 2, we described the problem as finding the best logical expression of a fixed length or less that covers a sample population. In this section, we consider the complexity of exhaustively generating and testing all possibilities. Except for relatively small populations or numbers of tests, with few potential cutoffs, the exhaustive approach is not computationally feasible.

---

[3]Two tests or combinations of tests may have the same value for the maximized measure. For example, two different tests may both have 100% positive predictive value. In this situation, the conjugate measure is used to decide which test is better. Sensitivity and specificity are treated as conjugates to one another and so are positive and negative predictive values. When maximizing accuracy, either sensitivity or specificity can be chosen as the next decisive function.

[4]An OR condition may encompass several conditions that are not mutually exclusive. The classification may have less than 100% diagnostic accuracy.

Equation 1 is the number of expressions having only ANDs; Equation 2 is for expressions having either ANDs or ORs.[5] In these equations, n is the number of tests, k is is the maximum number of tests in the expression, c is the number of constants (cutoff values) to be examined for each test, and c^i is c raised to the ith power. While the *number* of distinct values that must be examined for each test may vary, we have used a fixed number, c, to simplify the notation and analysis. In Equation 2, expressions are generated in disjunctive normal form, which corresponds to that used by the heuristic procedure described in Section 4.

$$\sum_{i=1}^{k} \binom{n}{i} c^i \tag{1}$$

$$\sum_{i=1}^{k} \binom{n}{i} c^i B_i \tag{2}$$

where B_i is the ith Bell number.The Bell number is the number of ways a set of i elements can be split into a set of disjoint subsets. For i=0,1,2,3, B_i=1,1,2,5 respectively [Andrews, 1976]. The Bell number is defined recursively as

$$B_{i+1}=\sum_{k=0}^{i} \binom{i}{k} B_k$$

For applications having several continuous variables, the most computationally expensive (exponential) component of Equation 2 component is c^i. It is possible to devise exhaustive procedures that do not require the examination of every value of a test found in the data set. For each test, one may examine only those points that overlap in the H+ and H- populations.[6] Even taking this into account, relatively small values of c will make the computation prohibitive. Because one may allow for the repetition of a test in an expression, for example a>50 OR (a >30 AND b <20), the number of generated expressions may be substantially greater than Equation 2.

The problem of finding the best subset of features or tests is a well-known and difficult statistical problem. For some statistical discrimination methods, such as normal linear discriminants, branch and bound methods can find an optimal feature subset for k tests [Narenda and Fukunaga, 1977, Roberts, 1984]. These procedures start with all n tests, eliminate one test at a time, and evaluate classification performance for subset of k tests. They are only optimal in terms of specific

---

[5]It is assumed that the less than or greater than operators are selected simply on the basis of the means or medians for each class.

[6]Moreover, only the smaller set of the two sets of points in the overlapping zone need be candidates for cutoffs. Each test would have a distinct number of cutoffs that must be examined, c_t. In the equations, instead of c^i, the products of c_t for each generated expression must be summed.

statistical distance measures, not in terms of error rates. For production rules, such techniques are not applicable. To compute the best k tests, it would be necessary to determine many subsets of tests with cutoffs of size much greater than k, which is not computationally feasible for larger k values.

4. A Heuristic Procedure for Maximizing Predictive Values

Because of the computational complexity of an exhaustive search, we have developed a heuristic search procedure for finding the best combination. In this section, we describe the procedure. While this procedure is not guaranteed to find an optimal solution, the expression found should almost always be quite good. In Section 6, empirical evidence is provided to demonstrate that in several situations the optimal production rule is found. In almost every real experimental situation, the logical expression found by the computer will be better than what a human experimenter can compose. These are situations where the experimenter is analyzing new data and does not know a priori the best rule.

Before specifying the heuristic search procedure, a few general comments can be made. In an exhaustive search approach, it is possible to specify a procedure that needs no additional memory. Logical expressions are generated and they are compared with the current best. The heuristic procedure is based on an alternative beam search strategy. A relatively small table of the most promising expressions is kept. Combinations of expressions are used to generate longer expressions. The most promising longer expressions in turn are stored in the table and are used to generate even longer expressions. Thus memory is needed to store the most promising or useful expressions. In Equation 2, the exponential component is the c^i. Thus, if one can reduce the value of c, i.e. the number of cutoffs for a test, the possible combinations are greatly reduced.

The Predictive Value Maximization (PVM) procedure was originally developed for finding the best logical combination of laboratory tests for making a diagnosis. In this section, we give a brief overview of the procedure.

The goal is to find the single best rule of length less than or equal to n. A rule for predicting a hypothesis or class consists of variables, constants, arithmetic operators and logical operators. The arithmetic operators are *less than*, *greater than*, or *equals*. The logical operators are AND or OR. For example, *X>30 OR Y<100* is a valid rule format. In terms of overall accuracy of classification, the best rule is the one that has the fewest number of errors in classification where the number of variables in the expression is no more than the stated length. The method is an approximation to exhaustive generation of all possible rules of a fixed length or less.

For each variable, *interesting* constants are determined. These cutoff points are local maxima of the predictive values. Logical expressions with variables are generated (in disjunctive normal form) and instantiated with the interesting constants. A relatively small table of the most promising expressions is kept. Combinations of the stored expressions are used to generate longer

expressions. The most promising longer expressions in turn are stored in the table and are used to generate even longer expressions.

Figure 4-1 illustrates the key steps of the heuristic procedure. In Section 4.1, the approach taken to greatly reduce the number of (interesting) cutoffs is discussed.

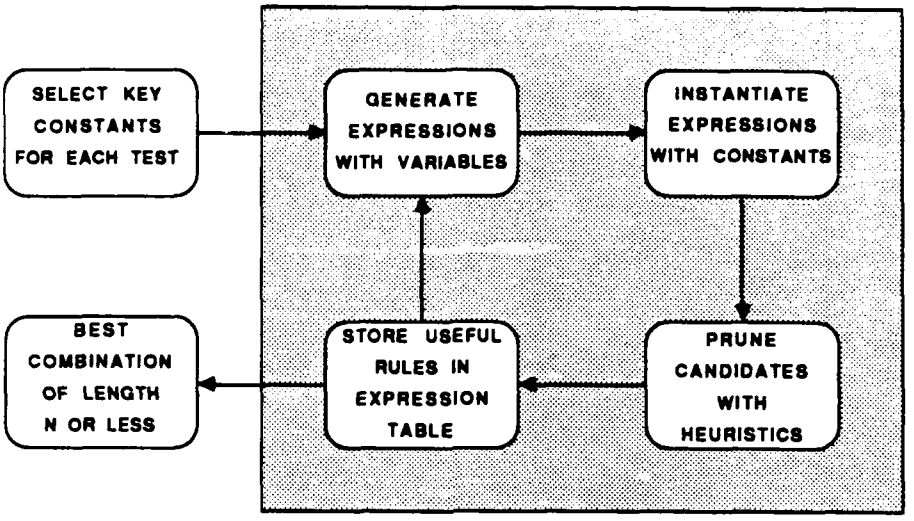


Figure 4-1: Overview of Heuristic Procedure for Best Test Combination

4.1. Selection of Cutoffs

For each test in the data set, the median is found for the cases satisfying the hypothesis (H+) and the cases not satisfying the hypothesis (H-). If the H+ has the greater median, the ">" operator is used. If H+ has the smaller median, the "<" operator is used. The equality operator "=" may also be used for discrete (categorical) tests corresponding to simple encodings that are unordered or have few values such as multiple choice questions.

The next task is to select the test cutoffs. For a test, cutoffs that fall at *interesting boundaries* are selected. Interesting boundaries are those where the predictive values (positive or negative) are locally maximum. For example, if WBC>10000 has a positive predictive value of 97% and WBC>9900 and WBC>10100 each has a positive predictive value less than 97%, then 10000 is an interesting boundary for WBC. The procedure first determines the interesting boundaries on a coarse scale. Then it zooms in on these boundaries and collects all the interesting boundaries on a finer scale. Finally, the boundaries are smoothened without changing the predictive statistics of the rule. Test cutoffs that have very low sensitivity or specificity are immediately pruned.[7]

[7]In the current version of the program, 10 equally spaced intervals are used for the region where the two populations overlap. For zooming in on an interval, 20 finer intervals are used between its 2 neighbors on the coarse scale. The minimum acceptable sensitivity or specificity for a test is currently set to be 10%.

4.2. Expression Generation

Logical expressions of all test variables in all combinations are generated in disjunctive normal form. This method avoids duplication of equivalent expressions since AND and also OR are symmetric. For example, a AND (b OR c) must be written as (a AND b) OR (a AND c). These expressions are stored in an expression table and longer expressions are generated combining shorter expressions. As each new expression is generated, the test variables are instantiated in all combinations of cutoff values. The test cutoffs were selected prior to expression generation. Figure 4-2 is a simple illustration of this process for 3 tests, {a, b, c} and expressions of length 2 or less.

a
b
c
a AND b
a AND c
b AND c
a OR b
a OR c
b OR c

Figure 4-2: Example of Expressions with Variables (tests)

If b has interesting cutoffs at b>10, b>20 and c has interesting cutoffs at c<30, c<40, c<50, then the expression b AND c would lead to the possibilities of Figure 4-3.

b >10 AND c <30
b >10 AND c <40
b >10 AND c <50
b >20 AND c <30
b >20 AND c <40
b >20 AND c <50

Figure 4-3: Example of Instantiated Expression

Because new longer expressions are generated from shorter expressions that have been stored in a table, those expressions that have been pruned will not appear in any longer expression. During the course of instantiation of the variables, some heuristics can be applied to prune the possibilities. These are discussed in Section 4.3.

4.3. Heuristics for Pruning Expressions

Although the heuristic cutoff analysis limits the search space to the most *interesting* cutoffs, the search space may still remain relatively large. Several heuristics and some provably correct pruning rules are employed by the procedure. The first 3 pruning rules are always correct, the others are heuristics that attempt to consider the most promising candidates for combination into new longer rules.

1. If the sensitivity and specificity values of an expression are both less than the constraints, then that expression does not contribute to any useful rules.

2. If an expression has less specificity than required, then any expression formed by ORing that expression with another will also have less specificity than required.

3. If an expression cannot be extended to one that contains all the mandatory tests, while satisfying the length constraint, it is immediately pruned.

4. If an expression has better positive and negative predictive values than another expression that differs from the first only by the constants in the expression, then the expression with lower predictive values is ignored.

5. If there are rules shorter and better than a new candidate rule, compute the sum of their lengths. If this sum, including the length of the current rule, exceeds the maximum length possible for any rule, then ignore the new rule. In the current implementation, the maximum rule length is fixed as 6. As the expression length increases, the number of potential combinations greatly increases as does the number of entries in the expression table. The objective of this heuristic is to emphasize the most promising shorter rules that will be combined into lengthier rules.

After all interesting expressions have been generated, the best expression in the expression table is offered as the answer.[8] Because all promising expressions are stored, a program that implements this procedure can readily determine its next best expression. If the constraints are made stricter, the expression table remains valid, and the procedure's new best expression is immediately available.

---

[8]During expression generation, whenever a superior expression is found, it is displayed. If no expression is found meeting the constraints, this is indicated when the search terminates. Depending on the allocated table space for storing intermediate expressions, the program may terminate from an overflow of the table. This is unlikely to occur with relatively small expressions.

4.4. Variations on the Standard PVM Application

The standard application of PVM was described in the previous section. The basic model is for two-class discrimination. Modifications can be made to the procedure to handle multiclass problems and step-wise refinement.

For multi-class problems, PVM is applied to a each class vs. the negation of that class, and the single best rule for each class is found. Thus the n-class problem is solved as n 2-class problems. For n classes, n rules are found, and the best n-1 are used. The remaining class is selected when no rule is satisfied. PVM is fundamentally a two class model, and it may be difficult to separate some classes from their negation. In these instances, it may be better to generate the n best rules, select the best one, remove the cases satisfying the rule, and then recursively re-apply the procedure. For the examples given in Section 6, it was not necessary to recursively apply PVM.

In some situations, another form of step-wise refinement is valuable. Because PVM may initially screen out some variables, the standard application of PVM may not work well for large numbers of variables or a low prevalence situation (i.e. many more cases of one hypothesis than another). PVM currently works with 18 variables at a time, and it uses only tests that have a minimum of 10% sensitivity or specificity. It does not apply both arithmetic operators (greater than and less than) simultaneously to the same variable. With step-wise refinement some of these restrictions can be overcome. Assuming a 2-class model, two strategies are worthwhile mentioning:

1. Find a highly predictive rule for a class; remove the cases satisfying the rule, and re-apply the procedure to the remaining cases that did not satisfy the rule.

2. Find a highly sensitive rule for a class, i.e. a rule that covers most or all of the cases in the class; remove the cases not satisfying the rule, and re-apply the procedure to the remaining cases that did satisfy the rule.

Situation (1) is equivalent to an OR condition, i.e. finding multiple rules to covering a class. An example of this variation is given in Section 6.1.2. Situation (2) is equivalent to an AND condition, i.e. extending a rule in step-wise fashion to create a longer rule. An example of this variation is given in Section 6.2.2.

5. Estimating Error Rates

5.1. Basic Principles of Error Estimation

A procedure has been described that finds a single rule that best covers the cases. It is well known that the *apparent* error rate of a classifier on all the training cases can lead to highly misleading and usually over-optimistic estimates of performance [Duda and Hart, 1973]. This is due to overspecialization of the classifier to the data. In the extreme, a classifier can be constructed

that simply consists of all patterns in the given sample. Assuming identical patterns do not belong to different classes, this yields perfect classification on the sample cases.

Techniques for estimating error rates have been widely studied in the statistics [Efron, 1982] and pattern recognition [Duda and Hart, 1973, Fukunaga, 1972] literature. The simplest technique for "honestly" estimating error rates, the holdout or H method, is a single train and test experiment. The sample cases are broken into two groups of cases: a training group and a test group. The classifier is independently derived from the training cases, and the error estimate is the performance of the classifier on the test cases. A single random partition of train and test cases can be somewhat misleading. The estimated size of the test sample needed for a 95% confidence interval is described in [Highleyman, 1962]. The following interpretation of these results is offered in [Duda and Hart, 1973]: "If no errors are made on 50 test samples, with a probability 0.95 the true error rate is between zero and eight percent. The classifier would have to make no errors on more than 250 test samples to be reasonably sure that the true error rate is below two percent."

Instead of relying on a single train and test experiment, multiple random test and train experiments can be performed. For each random train and test partition, a new classifier is derived. The estimated error rate is the average of the error rates for classifiers derived for the *independently* and randomly generated partitions. Random resampling can produce better error estimates than a single train and test partition.

A special case of resampling is known as leaving-one-out [Fukunaga, 1972, Efron, 1982]. Leaving-one-out is an elegant and straightforward technique for estimating classifier error rates. Because it is computationally expensive, it is often reserved for relatively small samples. For a given method and sample size n, a classifier is generated using n-1 cases and tested on the remaining case. This is repeated n times, each time designing a classifier by *leaving-one-out*. Each case is used as a test case and, each time nearly all the cases are used to design a classifier. The error rate is the number of errors on the single test cases divided by n.

Evidence for the superiority of the leaving-one-out approach is well-documented [Lachenbruch and Mickey, 1968, Efron, 1982]. While leaving-one-out is a preferred technique, with large samples it may be computationally expensive. However as the sample size grows, traditional train and test methods improve their accuracy in estimating error [Kanal and Chandrasekaran, 1971].

The leaving-one-out error technique is a special case of the general class of *cross-validation* error estimation methods [Stone, 1974]. In k-fold cross-validation, the cases are randomly divided into k mutually exclusive test partitions of approximately equal size. The cases not found in each test partition are independently used for training, and the resulting classifier is tested on the corresponding test partition. The average error rates over all k partitions is the cross-validated error rate. The CART procedure was extensively tested with varying numbers of partitions and 10-fold cross-validation seemed to be adequate and accurate, particularly for large samples where

leaving-one-out is computationally expensive [Breiman, Friedman, Olshen, and Stone, 1984][9] For small samples, bootstrapping, a method for resampling with replacement, has shown some promise as a low variance estimator for classifiers [Efron, 1983, Jain, Dubes, and Chen, 1987, Crawford, 1989].

Figure 5-1 compares the techniques of error estimation for a sample of n cases. The estimated error rate is the average of the error rates over the number of iterations. While these error estimation techniques were known and published in the 1960s and early 1970s, the increase in computational speeds of computers, makes them much more viable today for larger samples and more complex classification techniques [Steen, 1988].

	Holdout	Random Subsampling	Leaving-one-out	10-fold CV
Training cases	j	j	n-1	90%
Testing cases	n-j	n-j	1	10%
Iterations	1	<<n	n	10

Figure 5-1: Comparison of Techniques for Estimating Error Rates

Besides improved error estimates, there are a number of significant advantages to resampling. The goal of separating a sample of cases into a training set and testing set is to help design a classifier with a minimum error rate. With a single train and test partition, too few cases in the training group can lead to the design of a poor classifier, while too few test cases can lead to erroneous error estimates. Leaving-one-out, and to a lesser extent random subsampling, allow for accurate estimates of error rates while training on most cases. For purposes of comparison of classifiers and methods, resampling provides an added advantage. Using the same data, researchers can readily duplicate analysis conditions and compare published error estimates with new results. Using only a single random train and test partition introduces the possibility of variability of partitions to explain the divergence from a published result.

While error rates on test cases should be used to estimate the overall error rate for competing classifiers and methods, the best classifier design uses *all* cases in the sample set [Kanal and Chandrasekaran, 1971]. Resampling techniques provide better estimates of the error rates than a single train and test partition of the sample set [Efron, 1982].

[9]Empirical results also support the stratification of cases in the train and test sets to approximate the percentage (prevalence) of each class in the overall sample.

5.2. Resampling with PVM

Because PVM searches for a single rule of a fixed length. the procedure is particularly amenable to resampling techniques. Resampling is not limited to error estimation and can be used to estimate any population parameter [Efron, 1982]. PVM can be used in conjunction with resampling to estimate the expression length having the minimum expected error rate. The PVM induction procedure described in Section 4 does not directly indicate the specific rule length that yields the best performance. While increasing the length will never decrease performance on the training cases, performance on test cases may decrease. Thus after a certain length, estimated error rates may increase, due to overspecialization of the rule. Leaving-one-out and random resampling techniques can be used to provide estimates of the error rates for a specific expression length. In addition, these techniques can help perform a sensitivity analysis on competing expressions. Two estimating techniques are described: leaving-one-out and random resampling.

5.2.1. Leaving-one-out

PVM uses leaving-one-out in the following manner:

- For each expression length i, let J_i be the estimated error rate by leaving-one-out. Choose length k, such that J_k is minimum, i.e. choose the length that has the minimum expected error rate. Choose the best expression of length k using all n cases in the sample set.

- Alternatively, let k be the length of the rule with the minimum error rate. The leave-one-out procedure will generate n classifiers, where n is the sample size (total number of cases). Choose the rule that repeats the most times, i.e. the modal rule. This corresponds to a form of sensitivity analysis. Since only a single case is left out in each cycle, the pattern that is most stable and consistent with the estimated error rate is selected.

5.2.2. Random Resampling

When the data set is large, or the length of the expression is relatively long, leaving-one-out may be computationally too expensive. PVM uses random subsampling or 10-fold cross-validation in the following manner:

- For each expression length i, let RS_i be the estimated error rate by random resampling. Choose length k, such that RS_k is minimum, i.e. choose the length that has the minimum expected error rate. Select the best expression of length k using all n cases in the sample set.

- Alternatively, let k be the length of the rule with the minimum error rate. For each of the B test samples, generate the best rule of length k or less. If a rule frequently repeats, i.e. the mode is relatively large, choose the modal rule. If a pattern of variables and operators frequently repeats, but the constants vary (e.g. X> ? & Y< ?), apply the

induction method to *all* n cases. However, limit the process to the same variables and logical operators, adjusting only the constants.

6. Empirical Results

Because of the underlying empirical nature of the problem, by examining hundreds of possibilities, the program should be able to find better logical expressions than human experts when the samples are representative. This is particularly true when the human experimenter is examining new tests or performing an original experiment.

In the previous sections, the PVM procedure for rule induction was described. In the following sections, we will explore a number of remaining issues related to the performance of this procedure. Several data sets for which published studies are available were analyzed. The analysis of these data sets should help address the following questions:

- How close is the PVM solution to the optimal solution for the underlying model of a production rule formed by conjunction or disjunction of variables with constant cutoffs?
- How competitive is the rule-based model to other models, such as traditional statistical models?
- How competitive is PVM with other machine learning procedures?

6.1. Optimality and Model Adequacy

6.1.1. Production Rule Optimality

Several years after the appendicitis data used in our examples were reported in the medical literature, we re-analyzed the data. The samples consisted of 106 patients and 8 diagnostic tests. Because only 21 patients were normal, it is possible to construct an exhaustive procedure. In the original study, the experimenters were interested in maximizing accuracy, subject to the constraint of 100% sensitivity. Failure to treat was much less desirable than treating too many patients. In their paper, they cited a logical expression consisting of the disjunction of 3 diagnostic tests with positive predictive value of 89%. Using the heuristic procedure, the following results can be reported:

- A logical expression composed of only 2 tests with positive predictive value of 91% can be cited. The analysis takes 3 minutes of cpu time on a VAX 785.
- Using exhaustive search, the optimal expression of length 3 or less is identical to the one found by the heuristic procedure. The exhaustive search took 10 hours of cpu time on a VAX 785. The result reported in the literature was WBC>10500 OR MBAP>11% OR

CRP>1.2. The optimal solution is WBC>8700 OR CRP>1.8. The numbers in Figure 6-1 indicate that the shorter rule has the lower apparent error rates, Err_{App}, on all sample cases. Both rules do not classify 2 cases because of missing data. While the apparent error rate is a weak estimator of the true error rate, a shorter rule of equal or better performance is usually the better predictor on new cases, particularly when strictly empirical learning techniques are applied. Also listed is the the leaving-one-out error rate estimate, Err_{CV}, for the new rule.

	Original Err_{App}	New Err_{App}	New Err_{Cv}
Number of tests	3	2	2
Sensitivity	0.000	0.000	0.000
Specificity	0.526	0.421	0.474
Predictive value (+)	0.105	0.086	0.096
Predictive value (-)	0.000	.000	0.000
Accuracy	0.096	0.077	0.087

Figure 6-1: Comparison of Error Rates for Appendicitis Rules

6.1.2. Comparative Analysis for Iris Data

The iris data was used by Fisher in his derivation of the linear discriminant function [Fisher, 1936], and it still is the standard discriminant analysis example used in most current statistical routines such as SAS or IMSL. Linear or quadratic discriminants under assumptions of normality perform extremely well on this data set. Three classes of iris are discriminated using 4 continuous features. The data sets consists of 150 cases, 50 for each class. Figure 6-2 summarizes the results for the rule-based solution and several prominent statistical methods.[10] The optimal rules of size two were found by exhaustive search. These rules are quite simple and fully competitive with the other classifiers. *Petal length* < 3 perfectly separates Iris Setosa from the other classes and *Petal length* > 5 *OR Petal Width* > 1.7 separates Iris Virginica from the other classes with 3 errors. The PVM procedure directly finds two rules for Iris Virginica that have one more error than the optimal solution. By re-applying PVM to cases that did not satisfy one of the initially derived rules, the resultant ORed rule is equivalent to the optimal rule.[11]

As indicated in Figure 6-2, the iris discrimination problem is not difficult. Because of the wide citation of these data in examples for statistical programs, classification methods are unlikely to be

---

[10]The linear and quadratic discriminants assume a multivariate normal density function, and the results are from the discriminant analysis packages of SAS and IMSL. This quadratic discriminator has recently been used for evaluation functions in AI game playing with strong results [Lee and Mahajan, 1988]. The cited Bayes method assumes independent binary features; each continuous feature was divided into 10 discrete intervals.

[11]The rule is in the form of F3>5.1 OR F4>1.8 OR F3>4.9 OR F4>1.6. The optimal rule also is induced during one of the cross-validations.

Method	Err_{App}	Err_{Cv}
Linear	.020	.020
Quadratic	.020	.027
Nearest Neighbor	.000	.040
Bayes independence	.047	.067
Optimal Rule	.020	.020
PVM direct	.027	040
PVM indirect	.020	.020

Figure 6-2: Comparative Performance of Classifiers on Iris Data

accepted with poor performance on the iris data. We see that even in the classic normal case, the rule-based approach does well, and PVM finds an excellent expression. The CART work showed that decision trees perform extremely well relatively to competitive statistical classifiers [Breiman, Friedman, Olshen, and Stone, 1984]. Because production rules are related to decision trees, we can expect that rule-based solutions should also do well. In the next sections, we turn our attention to comparisons with alternative machine learning methods.

6.2. Comparison with Alternative Machine Learning Methods

6.2.1. Alternative Rule Induction Methods

A data set for evaluating the prognosis of breast cancer recurrence was analyzed by Michalski's AQ15 rule induction program and reported in [Michalski, Mozetic, Hong, and Lavrac, 1986]. There are 286 samples, 2 decision classes (recurrence of cancer or nonrecurrence) and 9 tests. They reported a 64% accuracy rate for expert physicians, and a 68% rate for AQ15, and a 72% rate for the pruned tree procedure of ASSISTANT [Kononenko, Bratko, and Roskar, 1986], a descendant of ID3. The prevalence of the larger class is 70%. The authors derived the accuracy rates by randomly resampling 4 times using a 70% train and a 30% test partition.

Because the authors randomly resampled, the experimental conditions can be replicated. Figure 6-3 is a summary of performance results (on the test cases). For length 2, the same expression,

Involved Nodes>0 & Degree=3

was selected by PVM on each of four 70% training samples, with an average accuracy of 77% on the test samples.[12] For these data, it is feasible to attempt to derive more accurate error estimates than can be found by randomly resampling four times on a 70% train, 30% test partition of the data set. By leaving-one-out the complete data set for rule length 2 and 3, one can see that the accuracy

[12]Using the same size partition, 20 additional trials were performed. The resultant error estimate was 76% on the test cases, and this rule appeared 16 times.

peaks at length 2 (.773 vs. .769 for length 3), and the same expression repeats itself each of the 286 times. Thus the modal rule is the only expression that is generated.

Method	Variables	Rules	Error Rate
AQ15	7	2	32%
PVM	2	1	23%

Figure 6-3: Comparative Summary for AQ15 and PVM on Breast Cancer Data

6.2.2. Alternative Decision Tree Induction Methods

Quinlan briefly reported on results of his analysis of hypothyroid data in [Quinlan, 1987b], and in greater detail in [Quinlan, 1987a]. The data consists of 3772 thyroid cases, representing almost all thyroid tests done at the Garvan Institute during 1985. Four hypotheses are considered, 3 types of hypothyroid disease (7.6% of the samples) and nonhypothyroidism. Because one of the classes is represented by only one case, data are available for two classes of hypothyroid disease: primary hypothyroid and compensated hypothyroid. Over 10% of the lab tests were unavailable, but all cases were classified. In the original study, a single random train and test partition was used: 3143 cases for training and 629 cases for testing. In some instances, only 2514 cases were used for training.[13] Quinlan's C4 program produced decision trees, and he used pruning routines to produce a small set of production rules that performed better (than the original tree) on the test cases [Quinlan, 1987a]. In the published study we are given a set of two induced rules.[14]

The question we address is whether there are better rules that can be induced from the 3772 cases. A number of factors, which taken together, make a comparative analysis between the published results and PVM's results seem difficult. These include the use of a single random partition of test cases, the low prevalence of 7.6% for hypothyroidism, and the excellent very low error rates achieved by Quinlan's C4 program. However, a new analysis is quite feasible because 3428 new cases for the year 1986 are also available. Without training on them, the 3428 new cases can provide objective verification as to whether improved results have been achieved.

Figure 6-4 summarizes C4's published results and PVM's on all 3772 cases from the year 1985 and on the 3428 new cases from 1986. Only the cases from 1985 were used for rule induction. The cases from 1986 are used solely for verification of the results. Because of the large number of cases and high accuracy levels, the number of errors is cited instead of error rates.

---

[13]Quinlan performed experiments to examine whether it is advantageous to have a separate set of cases that are used during training to guide the induction procedure. A second set of 629 cases were drawn from the 3143 training cases for this purpose, leaving 2514 training cases.

[14]A third rule cited for nonhypothyroid is equivalent to the absence of either of the two rules for the specified diseases.

Method	Variables	Rules	Errors (1985)	Errors (1986)
C4 pruned rules single holdout	8	2	31	43
PVM random resampling	8	2	17	30

Figure 6-4: Comparative Summary for C4 and PVM on Hypothyroid Data

PVM's performance was achieved using the random resampling procedure described in Section 5.2.2. The leaving-one-out procedure is computationally too expensive for this size data set. While standard procedure would involve using 3143 training cases, we used only 2514 training cases and 629 test cases for consistency with all of Quinlan's pruning experiments. Ten randomly drawn samples of train and test cases were used for each of the two diagnoses and the average number of errors (on the 629 test cases) for each length is given in Figure 6-5. Lengths beyond 6 were not considered. A length of zero represents the number of errors for no rule, i.e. the prevalence.

For the primary hypothyroid diagnosis the minimum error length is 2 and the modal rule is *TSH>6.1 & FTI <65.* This is also the rule that PVM induces for all 3772 cases. The characteristics of the random resampling analysis for the primary diagnosis are listed in Figure 6-6.

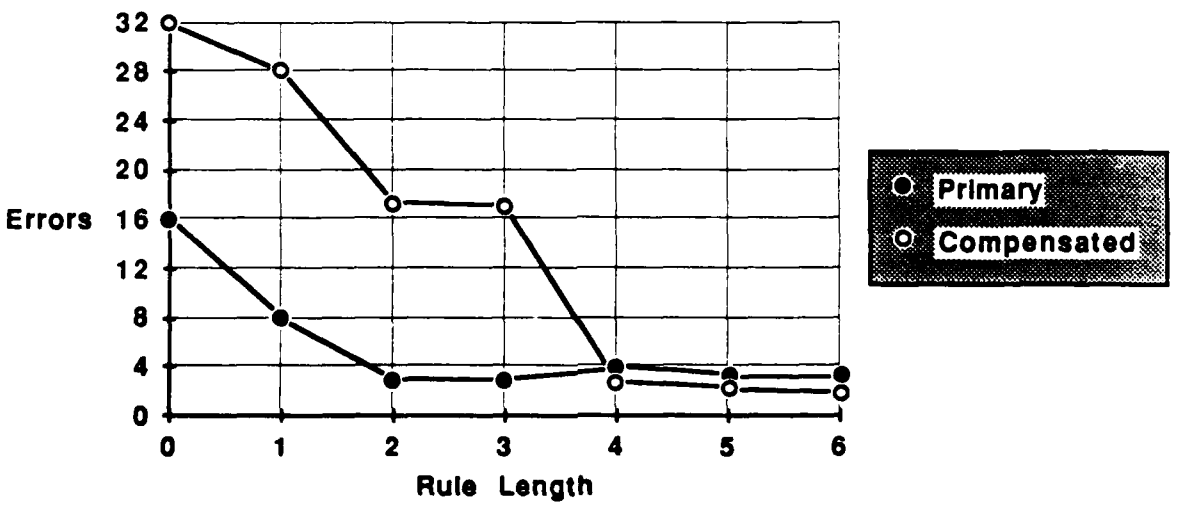


Figure 6-5: Estimated (average) Errors for Hypothyroid Diagnoses

For the compensated hypothyroid diagnosis, the minimum error length is 6, and the modal rule is

TSH>6 & TT4<149 & On Thyroxin=false & (FTI>64 or unknown) & TT4>50 & Surgery=false.

This is also the rule that PVM induces for all 3772 cases. A shorter rule also yields good results. If

the rules are restricted to length 4 or less, the results are 24 errors for the year 1985 cases and 36 errors on the year 1986 cases.[15]

Attribute	Value
Number of runs	10
Training sample size	2514
Test sample size	629
Minimum error length	2
Modal rule	TSH>6.1 & FTI<65
Rule mode j=2514	5
Modal variables	TSH>? & FTI<?
Variables only mode j=2514	10

Figure 6-6: Summary of Analysis of Primary Hypothyroid Diagnosis

PVM was originally designed to find the best combinations of medical lab tests. A typical application of this type would have a few hundred cases and relatively few unknown test results. The PVM procedure eliminates from consideration tests not having at least 10% sensitivity or specificity, because these are not considered good tests for a diagnostic class. We also prefer not to classify cases when the induced rules cannot make a decision because of missing data.

As presented, the original hypothyroid data analysis is somewhat atypical of an expected PVM application. The sample sizes are quite large, and most classes have a low prevalence. While the PVM procedure was not modified for this application, PVM was applied in two stages. This was also necessary for computational reasons. Lengths beyond 3 were calculated in two parts: (a) the best rule of length 3 with 90% sensitivity, and the continuation, (b) the best rule up to an additional length 3 for cases satisfying rule (a). In a low prevalence environment, the two part application is helpful in the selection and filtering of useful tests and in the classification of unknowns. Tests that have less than 10% sensitivity for all cases are not used in finding rule (a). These same unused tests may have greater than 10% sensitivity for cases satisfying rule (a) and may be used in finding rule (b). While some class prevalences may be low over all cases, the prevalence for classes satisfying rule (a) may be high. This may change the classification of cases that satisfy rule (a) but are unknown for the continuation, i.e. rule (b). PVM does not induce rules that explicitly state that a test must be unknown to reach a conclusion. However, for the rule induced for compensated

[15] A more direct comparison with the original C4 experiments can be made when each trial is considered a single holdout trial, and the minimum error rule on 629 test cases is selected. None of the 10 PVM runs had more than 26 errors on the cases from 1985 or 39 errors on the cases from 1986, and the average was 21.5 errors for year 1985 and 33.8 for 1986.

hypothyroidism, FTI is the only test that has unknown values in the data set. The FTI component of the rule is induced in the second stage, when the odds have already shifted to compensated hypothyroidism.

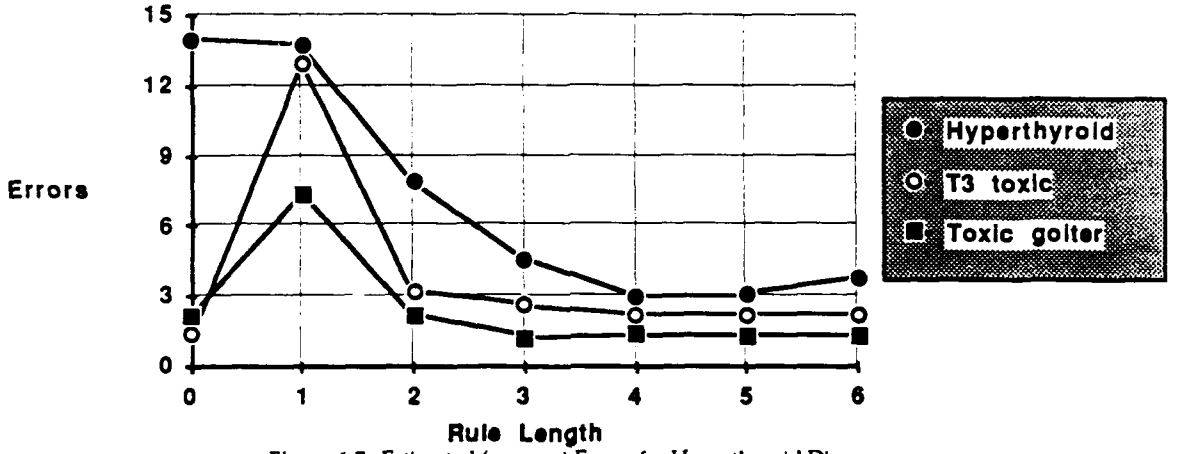


Figure 6-7: Estimated (average) Errors for Hyperthyroid Diagnoses

The same 3772 cases from 1985 were used in a separate study of rule induction for hyperthyroidism [Quinlan, 1987c]: 2800 cases for training and 972 cases for testing. There are sufficient cases to attempt to diagnose 3 hyperthyroid conditions. Again we ask the question whether better rules can be induced from the 3772 cases than those cited in [Quinlan, 1987c]. Using a 2100 case training set and 700 case test set, the errors (on the 700 test cases) for each length are summarized in Figure 6-7.

Method	Variables	Rules	Errors (1985)	Errors (1986)
C4 pruned tree single holdout	13	3	41	54
C4 pruned rules single holdout	38	7	28	48
PVM random resampling	7	2	31	46

Figure 6-8: Comparative Summary for C4 and PVM on Hyperthyroid Data

Two sets of rules are cited in [Quinlan, 1987c]: the rules formed by a single decision tree and a rule induction procedure that prunes a collection of decision trees [Quinlan, 1987b]. Their performance and that of PVM using 10 randomly drawn samples for each diagnosis is given in

Figure 6-8.[16] Interestingly, there are insufficient data for inducing a rule for T3 toxic, because the expected error rate is greater than the prevalence. The rule found by PVM for hyperthyroidism is

FTI>155 &
TT4>149 & On Thyroxin=false & (TSH<0.3 or unknown)

and the rule found for toxic goiter[17] is *Goiter & T3>2.8 & FTI<153.*

7. Discussion

The PVM procedure was originally developed for laboratory medicine applications [Weiss, Galen, and Tadepalli, 1987]. It was intended to help researchers find combinations of numerical tests that have greater predictive value than single tests. PVM assumes that a *single* short rule exists to classify a hypothesis. It does not expect perfect classification, and it can tradeoff false positive vs. false negative error rates.

Because relatively few tests are expected to be analyzed, an approximation to exhaustive enumeration was considered. For several hundred (varying) cases, exhaustive enumeration is not feasible, but experimental results support the contention that the PVM procedure will yield excellent, sometimes optimal results. In two studies where the optimal results for rules of a fixed length can be determined, PVM was able to find an optimal or near-optimal solution. Rule-based solutions appear to be quite competitive with alternative statistical procedures, with the advantage of simplicity and clarity of presentation. In its current implementation, PVM handles up to 18 tests at a time; filtering procedures and multi-stage analysis can be employed to reduce the number of tests to 18 at each stage.

In this paper, we re-analyzed data that had been analyzed using prominent machine learning techniques. We showed that superior rules could be induced from these data sets. In the case of the cancer data, a simple two variable rule produces better results than the more complex rules cited in the literature. While the published thyroid data analysis demonstrated excellent results, we showed that somewhat better rules can be induced than those cited in the original studies.

As is used in the CART procedure, resampling techniques are employed by PVM to estimate error rates for induced production rules. These techniques can be time-consuming, but can lead to better induction results. Because PVM induces rules for a fixed, relatively short length, resampling procedures are a natural extension of the basic method. The major advantage is that error estimates can be derived, while essentially the complete data sample may be used for classifier design.

---

[16] If each trial is considered a single holdout trial, and the minimum error rule on 700 test cases is selected, then none of the 10 runs had more than 38 errors on the cases from 1985 or 53 errors on the cases from 1986, and the average was 33.4 errors for year 1985 and 46.1 for 1986. Only one of the ten trials had more than 46 errors on the cases from 1986.

[17] This is the result for length 3 with 90% sensitivity.

PVM is not always superior to other empirical rule or tree induction procedures. Unlike the alternative methods, PVM in practice is limited to the induction of single short rules in relatively low dimensions with few classes. However, if a good solution exists in the form of a single short rule, PVM should have an advantage. Unlike incremental empirical induction procedures that select one test at a time, PVM examines combinations of tests with varying constants. There are many applications, such as when testing is expensive, where a short rule is highly desirable.

In comparison with faster learning methods, such as decision trees, there is a space and time tradeoff. PVM examines many more combinations and this takes extra time. In the cited applications, the number of combinations stored in the expression table for a single length 3 rule ranged from a low of 117 for the iris data to 1480 for the appendicitis data. Additional factors affect computing time, such as the number of cases and the number of expressions examined but rejected. The longest cpu time for induction of a single rule was for one of the thyroid diagnoses, which took less than hour on a Sun 4/280 processor.

Researchers in machine learning have noted that relatively small pruned rules often yield better results than more complex sets of induced rules [Quinlan, 1987b, Michalski, Mozetic, Hong, and Lavrac, 1986]. The number and size of rules that can be effectively inferred from even large data sets is often surprisingly small. The number of rules in many rule-based expert systems far exceeds those found in these machine learning applications. However, the rules in an expert system knowledge base are based on current known expertise. Induction procedures offer the potential to learn rules that are currently unknown. Clearly, humans are not competitive in this form of analysis. Using strictly empirical data, it is unlikely that a human can find a better rule than the computer. While the same argument could be made for a purely statistical analysis, decision rules are more consistent with human decision-making. With improved techniques and faster computers, we can expect to see greater use of induction techniques to help discover new decision rules and to verify and refine the quality of current rules acquired from experts.

In terms of knowledge base acquisition, this approach can prove valuable in both acquiring new knowledge, refining existing knowledge [Wilkins and Buchanan, 1986, Ginsberg, Weiss, and Politakis, 1988], and verifying correctness of old knowledge. Because a knowledge base of rules summarizes much more experiential knowledge than is usually covered by a data set of cases, in many instances this approach can be thought of as supplementary to the knowledge engineering approach to knowledge acquisition in rule-based systems.

Acknowledgments

We acknowledge the programming support of Kevin Kern, who programmed and tested many of the procedures described in this paper.

References

[Andrews, 1976]
Andrews, G. *Encyclopedia of Mathematics and its Applications II - The Theory of Partitions.* Reading, Mass.: Addison-Wesley, 1976.

[Breiman, Friedman, Olshen, and Stone, 1984]
Breiman, L., Friedman, J., Olshen, R., and Stone, C. *Classification and Regression Tress.* Monterrey, Ca.: Wadsworth, 1984.

[Clancey, 1985]
Clancey, W. "Heuristic Classification." *Artificial Intelligence.* 27 (1985) 289-350.

[Crawford, 1989]
Crawford, S. "Extensions to the CART Algorithm." *International Journal of Man-Machine Studies.* (1989) in press.

[Duda and Hart, 1973]
Duda, R., and Hart, P. *Pattern Classification and Scene Analysis.* New York: Wiley, 1973.

[Efron, 1982]
Efron, B. "The Jackknife, the Bootstrap and Other Resampling Plans." In *SIAM.* Philadelphia, Pa., 1982.

[Efron, 1983]
Efron, B. "Estimating the Error Rate of a Prediction Rule." *Journal of the American Statistical Association.* 78 (1983) 316-333.

[Fisher, 1936]
Fisher, R. "The Use of Multiple Measurements in Taxonomic Problems." *Ann. Eugenics.* 7 (1936) 179-188.

[Fu and Buchanan, 1985]
Fu, L. and Buchanan, B. "Learning Intermediate Concepts in Constructing a Hierachical Knowledge Base." In *International Joint Conference on Artificial Intelligence.* Los Angeles, 1985, 659-666.

[Fukunaga, 1972]
Fukunaga, K. *Introduction to Statistical Pattern Recognition.* New York: Academic Press, 1972.

[Galen and Gambino, 1975]
Galen, R. and Gambino, S. *Beyond Normality: The Predictive Value and Efficiency of Medical Diagnoses.* New York: John Wiley and Sons, 1975.

[Ginsberg, Weiss, and Politakis, 1988]
Ginsberg, A., Weiss, S., and Politakis, P. "Automatic Knowledge Base Refinement for Classification Systems." *Artificial Intelligence.* (1988) 197-226.

[Highleyman, 1962]
Highleyman, W. "The Design and Analysis of Pattern Recognition Experiments." *Bell System Technical Journal.* 41 (1962) 723-744.

[Jain, Dubes, and Chen, 1987]
Jain, A., Dubes, R., and Chen, C. "Bootstrap Techniques for Error Estimation." *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 9 (1987) 628-633.

[James, 1985]
James, M. *Classification Algorithms*. New York: John Wiley & Sons, 1985.

[Kanal and Chandrasekaran, 1971]
Kanal, L. and Chandrasekaran, B. "On Dimensionality and Sample Size In Statistical Pattern Classification." *Pattern Recognition*. (1971) 225-234.

[Kononenko, Bratko, and Roskar, 1986]
Kononenko, I., Bratko, I., Roskar, E. "ASSISTANT: A System for Inductive Learning." *Informatica*. 10 (1986) .

[Lachenbruch and Mickey, 1968]
Lachenbruch, P. and Mickey, M. "Estimation of Error Rates in Discriminant Analysis." *Technometrics*. (1968) 1-111.

[Lee and Mahajan, 1988]
Lee, K. and Mahajan, S. "A Pattern Classification Approach to Evaluation Function Learning." *Artificial Intelligence*. 36 (1988) 1-25.

[Marchand, Van Lente, and Galen, 1983]
Marchand, A., Van Lente, F., and Galen, R. "The Assessment of Laboratory Tests in the Diagnosis of Acute Appendicitis." *American Journal of Clinical Pathology*. 80:3 (1983) 369-374.

[McClelland and Rumelhart, 1988]
McClelland, J. and Rumelhart, D. *Explorations in Parallel Distributed Processing*. Cambridge, Ma.: MIT Press, 1988.

[Michalski, Mozetic, Hong, and Lavrac, 1986]
Michalski, R., Mozetic, I., Hong, J., and Lavrac, N. "The Multi-purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains." In *Proceedings of the Fifth Annual National Conference on Artificial Intelligence*. Philadelphia, Pa., 1986, 1041-1045.

[Narenda and Fukunaga, 1977]
Narenda, P. and Fukunaga, K. "A Branch and Bound Algorithm for Feature Subset Selection." *IEEE Transactions on Computers*. C-26 (1977) 917-922.

[Quinlan, 1986]
Quinlan, J. "Induction of Decision Trees." *Machine Learning*. 1 (1986) 1.

[Quinlan, 1987a]
Quinlan, J. "Simplifying Decision Trees." *International Journal of Man-Machine Studies*. 27 (1987) 221-234.

[Quinlan, 1987b]
Quinlan, J. "Generating Production Rules from Decision Trees." In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*. Milan, Italy, 1987, 304-307.

[Quinlan, 1987c]
Quinlan, J. "Induction, Knowledge and Expert Systems." In *Australian Joint Conference On AI*. Sydney, Australia, 1987, .

[Roberts, 1984]
Roberts, S. "A Branch and Bound Algorithm for Determining the Optimal Feature Subset of Given Size." *Applied Statistics*. 33 (1984) 236-241.

[Spackman, 1988]
Spackman, K. "Learning Categorical Decision Criteria in Biomedical Domains." In *International Conference on Machine Learning*. Ann Arbor, 1988, 36-46.

[Steen, 1988]
Steen, L. "The Science of Patterns." *Science*. 240 (1988) 611-616.

[Stone, 1974]
Stone, M. "Cross-Validatory Choice and Assessment of Statistical Predictions." *Journal of the Royal Statistical Society*. 36 (1974) 111-147.

[Weiss and Kulikowski, 1984]
Weiss, S. and Kulikowski, C. *A Practical Guide to Designing Expert Systems*. Totowa, New Jersey: Rowman and Allanheld, 1984.

[Weiss, Galen, and Tadepalli, 1987]
Weiss, S., Galen, R., and Tadepalli, P. "Optimizing the Predictive Value of Diagnostic Decision Rules." In *Proceedings of the Sixth Annual National Conference on Artificial Intelligence*. Seattle, Washington, 1987, 521-526.

[Wilkins and Buchanan, 1986]
Wilkins, D. and Buchanan, B. "On Debugging Rule Sets When Reasoning Under Uncertainty." In *Proceedings of the Fifth Annual National Conference on Artificial Intelligence*. Philadelphia, Pa., 1986, 448-454.

Models for Measuring Performance of Medical Expert Systems

Nitin Indurkhya *and* **Sholom M. Weiss**
Department of Computer Science, Rutgers University, New Brunswick, NJ 08903

Abstract

Scoring schemes for measuring expert system performance are reviewed. Rule-based classification systems and their error rates on sample data are considered. We present several models of measurement that are categorized by four characteristics – mutual exclusivity of classes, unique answers provided by the system, known correct conclusions for each case and use of confidence factors to weight the system's conclusions. An underlying model of performance measurement is critical in determining which scoring strategy is appropriate for a system and whether a comparison of different medical expert systems can be made.

1. Introduction

Although many medical expert systems have been developed, relatively few have been formally evaluated [8, 12, 21]. Different methods of measuring performance can be used for each system, making it difficult to compare their performance. In this report several underlying models of measuring performance are considered. Here *performance* refers to the error rates of systems on sample data.

A survey of strategies for measuring performance of medical expert systems can be found in [12] which proposes three levels of evaluation:

1. The subjective assessment of the research contribution of a developmental system.
2. The validation of a system's knowledge prior to possible clinical use.
3. The evaluation of the clinical efficacy of an operational consultation system.

In terms of these levels, this report deals with the second level of evaluation – validation of system performance prior to possible clinical use.

Rather than surveying performance measurement strategies in existing systems, the focus, in this paper, is on underlying models. Classification type rule-based expert systems [2, 18] are considered, although the results can be carried over to other classification systems [7]. The discussion is confined to scoring and measuring performance, as opposed to techniques (such as *train and test*) for estimating bias in the measured error rates [4]. A review of performance measurement models is critical in determining which scoring strategy is appropriate for a system and whether a comparison of different medical systems can be made.

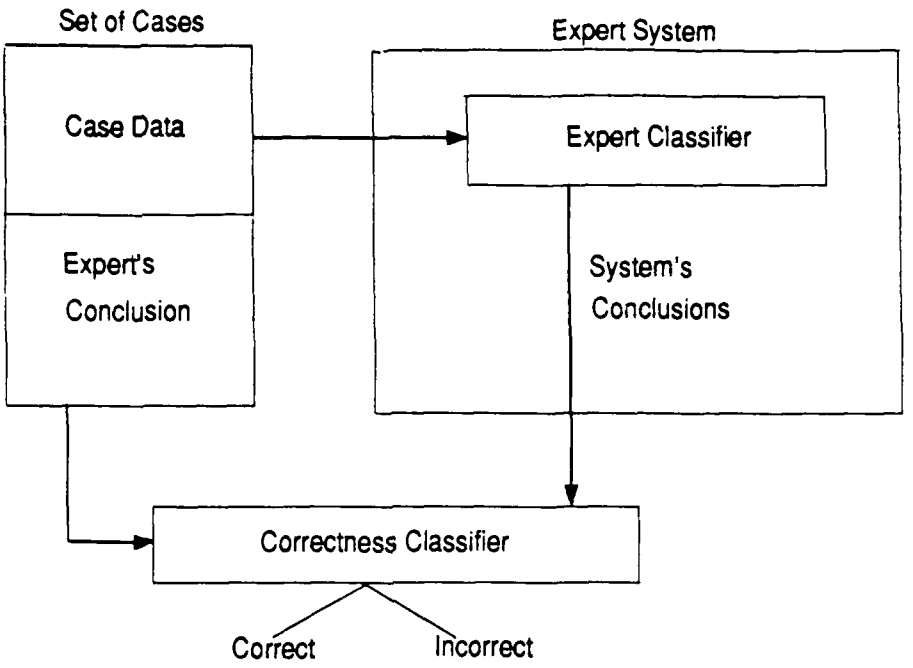


Figure 1: Measuring Expert System Performance

2. Measurement Models For Classifiers

A performance measurement model for a classification expert system requires several basic elements. Figure 1 illustrates the components of a standard performance measurement model. In Figure 1, the scoring of a case depends on the system's conclusion and the expert's conclusion. The structure of the classifier is not critical – it could be a set of rules, a decision tree or even a procedural algorithm. The only thing that is of concern is that the classifier takes case-data as input and outputs an answer. Thus, for evaluation purposes, the expert classifier is treated as a black box. In fact the view taken is that of an examiner who is given a system and told to measure its performance. The examiner does not care how the system is designed internally and focuses on the output of the system for a given set of input values.

For the purposes of measuring performance, the following characteristics are of interest:

Unique Answer This attribute refers to whether the system's conclusion for a case is unique. Thus, if a classifier always outputs a single class as its conclusion for all the test cases, the system is said to be *UA restricted*. Otherwise, the system is *Not-UA restricted*. As an example, consider the conclusions listed in Table 1 for two systems classifying among (*class1*, *class2*, *class3*) over the same set of cases. For the given set of cases, System A is *UA restricted* and System B is *Not-UA restricted*. This will affect what kind of scoring strategy is best suited for evaluating these systems. The scoring scheme for System A will be different from that for System B.

Case	System A	System B
case1	*class1*	*class1*, *class2*
case2	*class3*	*class2*
case3	*class1*	*class1*, *class3*

Table 1

Mutual Exclusive Classes This attribute refers to whether, for a given case, only one class can be satisfied. This will be true if the classes considered are mutually exclusive. It is assumed that the classes span the space of alternatives[1]. Thus, if every case can be classified in one and only one of the classes, then we say that the domain is *ME restricted.* If there are cases that can be classified in more than one class, then we say that the domain is *Not-ME restricted.*

Known Case Conclusion This attribute refers to whether the correct conclusion for a case is *known* and *undisputed.* The correct conclusion can be obtained from an expert or the true conclusion can be obtained by observation over time until the event is over. Thus, for a system which predicts rain for the next day, the true conclusion can be obtained by waiting until the next day and observing whether it rains or not. This will be the correct conclusion for the case. If the true conclusion is obtainable, then it is also *undisputed.* However, if the correct conclusion is obtained from an expert, then it may be disputable. For a case, two different experts may diagnose differently.

Weighted Answers This attribute refers to whether the system's answers are classified by uncertainty factors. If the system gives unweighted answers without any certainty factors, its answers are considered definite. For example, given a three class situation of *class1*, *class2* and *class3*, an expert system with unweighted answers might give the following output – (*class2*, *class3*). This will be taken to mean that the case can be classified as both *class2* and *class3* and that both the classifications are equally definite. A system gives weighted answers if it makes use of an uncertainty model in its reasoning. Such a system typically provides a likelihood measure or a Confidence Factor (CF) with each class – the CF reflecting how strongly the system supports this class as an answer. For example, for the three class situation described above, an expert system with weighted answers might have output as given in Table 2. Thus, the system has no confidence that the case can be classified as *class3* and has a higher confidence in classifying the case as *class1* than as *class2*. In the above example, note that the confidence factors do not sum up to 1.0. This means that the uncertainty model is not based on probability theory. This is not unusual. Many systems use non-probabilistic uncertainty models. For purposes of measuring performance, the theory underlying the uncertainty model is not significant.

[1] If the classes do not span a complete set of possibilities, then an additional class *None-of-the-above* can be added to the list of classes.

class1	0.7
class2	0.5
class3	0.0

Table 2

The above features will be used to categorize scoring strategies and describe models for measuring performance.

3. Models For Systems Restricted To Mutual-Exclusivity And Unique Answers

The simplest situation occurs when the correct conclusion for the case is *undisputed* and *known*, the classes are mutually exclusive and span the possibilities, and the system's conclusion is unique. A classifier for a ME-restricted domain, usually gives a unique answer for any input data[2]. Thus for a ME-restricted problem, most classifiers would be UA-restricted and for any input data would give a unique answer. In a rule-based system, where no rules or multiple rules can be satisfied, a *unique answer* is equivalent to conflict resolution in which a single answer is selected by some criteria such as highest confidence factor, etc. These criteria can be encapsulated into a *unique answer generator*. Because of this, it does not matter whether the answers are weighted or not. The situation for UA-restricted classifiers in ME-restricted domains is shown is Figure 2. Performance measurement models for such systems are examined in the rest of this section.

3.1. The Correctness Model

The most common measurement strategies are based on measuring the total number of correct decisions made by a system. This measure is popular among machine learning systems [10] where performance improvement is demonstrated by counting the number of correct responses before and after learning. The underlying model is one in which the number of correct/incorrect responses are being noted. Thus, for example, for a two class situation (mutually exclusive classes):

$$\text{System's response} = C_{system}$$
$$\text{Correct response} = C_{correct}$$

A case is classified as correct or incorrect using Table 3. With this information, Table 4 is filled up. The metric of percentage of errors is calculated as:

$$\%\ \text{of errors} = \frac{Incorrect}{Correct+Incorrect} \times 100.$$

[2]When classifiers for ME-restricted domains are not UA-restricted, this leads to analytical difficulties. This issue will be discussed later on in this report.

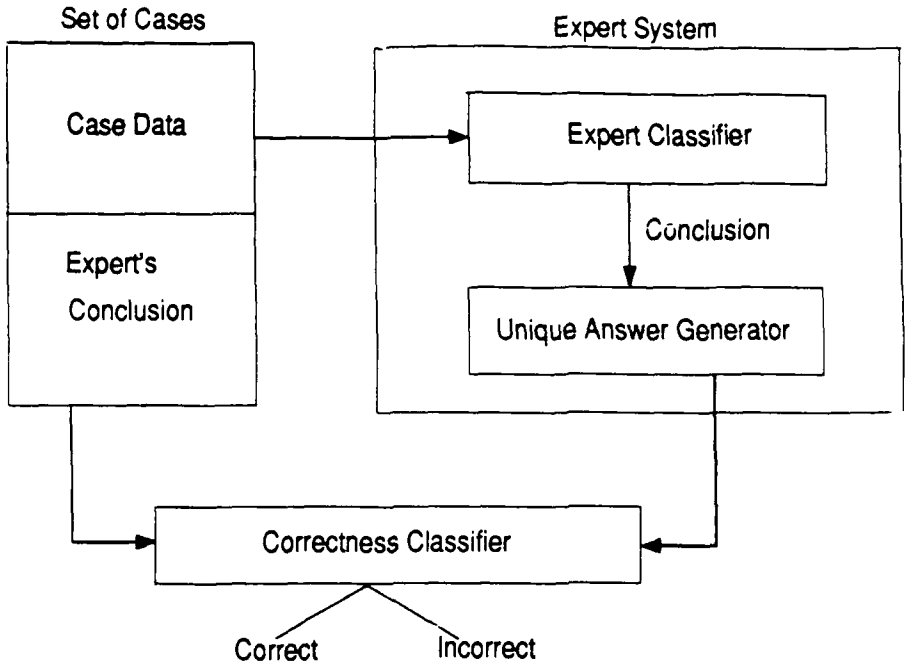


Figure 2: Systems Restricted To Unique Answers And Mutual Exclusivity

	C_{system}	
$C_{correct}$	Class1	Class2
Class1	Correct	Incorrect
Class2	Incorrect	Correct

Table 3

Alternatively, one could use a *confusion matrix* [7]. A confusion matrix is a N×N table of *actual class* against *classified class* where N is the number of classes. This is illustrated by the following example. Suppose for a three class situation, a classifier produces the confusion matrix given in Table 5 in scoring a set of 50 cases: Note that each entry in the confusion matrix refers to the number of cases having the corresponding C_{system} and $C_{correct}$. Thus, for example, the entry in the first row and second column means that in 2 cases C_{system} was Class2 and $C_{correct}$ was Class1. Correctly classified cases fall on the diagonal moving from the upper left to the lower right. Based on the the confusion matrix, the overall error rate can be determined as in Table 6. This gives the following result for the system:

$$\% \text{ of errors} = \frac{14}{50} \times 100 = 28\%$$

This model is the simplest measurement model and compares performance at a coarse level. It does not distinguish between different types of errors.

Correct	Incorrect

Table 4

$C_{correct}$	C_{system}		
	Class1	Class2	Class3
Class1	13	2	5
Class2	2	14	1
Class3	3	1	9

Table 5

Correct	Incorrect
36	14

Table 6

3.2. The Positive/Negative Correctness Model

Some systems use a more detailed model than the correctness model described above. Instead of classifying a case as correct, it is now classified as *True Positive* or *True Negative*. Instead of classifying a case as incorrect, it is now classified as *False Positive* or *False Negative*. This classification is done with respect to a class.

As an example, the tables involved for a two class situation are shown in Table 7. Note that the tables are with respect to class1. For N classes, the table with respect to Class1 would be as

$C_{correct}$	C_{system}	
	Class1	Class2
Class1	True Positive	False Negative
Class2	False Positive	True Negative

Table 7

shown in Table 8. Using the above classification for each case, Table 9 is calculated for each class: For each table, H+ and H- add up to the total number of cases as do T+ and T-. The tables for the classes are related. A FP case for *class1* will be a FN case for *class2*. The sum of all the TP's and FN's from all the classes equals the total number of cases. Since FN's for a class are FP's for other classes, it is enough to report TP and FP values for each class.

$C_{correct}$	C_{system}				
	Class1	Class2	Class3	...	ClassN
Class1	TP	FN	FN	...	FN
Class2	FP	TN	FN	...	FN
Class3	FP	FN	TN	...	FN
⋮	⋮	⋮	⋮	⋱	⋮
ClassN	FP	FN	FN	...	TN

Table 8

	T+	T-
H+	TP	FN
H-	FP	TN

Table 9

A number of metrics are possible using this model. Some of them are listed in Table 10. A system which uses this model is described in [5]. The metrics *specificity* and *sensitivity* are

Sensitivity	TP/H+
Specificity	TN/H-
Predictive value(+)	TP/T+
Predictive value(-)	TN/T-
Accuracy	(TP+TN)/TOTAL

Table 10

commonly used in scoring laboratory tests in medicine. It is important to note that there are two kinds of errors being distinguished: *false positive* errors and *false negative* errors. They could be considered of unequal importance.

The computations can be done easily using a *confusion matrix* similar to the example given for the correctness model. The tables described above can provide useful information for improving the performance of the classifier. For example, it might indicate that the classifier does very well on all the classes but one – thereby indicating that the performance can be improved by trying to improve the results for this one class. SEEK2 [6] is an example of a system that uses the matrix of True Positives, False Negatives, False Positives and True Negatives as a source of information for improving performance which it measures using the correctness model.

3.2.1. The Correctness And Cost Model

This is a generalization of the positive/negative correctness model. There is now a cost factor associated with making a FP or FN judgment. This kind of model is useful when it is known that the cost of making a FP decision is different from making a FN decision. For example, in a situation of approving credit card requests, the cost of a FN decision (not approving a good applicant) is relatively low (the annual card fee and interest payments) while the cost of a FP decision (approving an applicant who is unable to pay his bills) is high (the average annual purchases of a credit card holder). Thus one calculates Table 11. The positive/negative correctness model is a special case

	T+	T-
H+	TP	FN × FNcost
H-	FP × FPcost	TN

Table 11

of this model with $FNcost = FPcost = 1$. Note that the cost function need not be linear. For example, if it is important that the number of FN's be not more than 50% of the FP's, then the non-linear cost function which needs to be calculated would be $f(FP, FN) = FP/FN$. The same set of metrics as used with the positive/negative correctness model are relevant here.

A number of optimality measures for classifiers can be understood as special cases of this model. Thus, the criteria described in [7, Pg 63] - *Minimum cost, Minimax error* and *Fixed error rate* can be all seen to be within the scope of this model. The reader is referred to [7] for details.

4. Models For Systems Restricted To Mutual-Exclusivity And Non-Unique Answers

In all the models discussed so far, three important assumptions have been made:

Known Case Conclusion: The correct conclusion was *known* and *undisputed.*

ME Restricted Domain: The classes were mutually exclusive.

UA Restricted Classifier: The expert system incorporated a unique answer generator thereby ensuring that for any input data, the system always gave a unique answer.

These assumptions simplify considerably the factors affecting measurement. We now relax the third of the abovementioned constraints and consider *Not-UA restricted* expert systems. So the situation is that the classes are mutually exclusive and the classifier can give more than one class as its conclusion[3].

[3]The conclusion for each case is still known and undisputed.

It was mentioned earlier that when a classifier for a ME restricted domain is not UA restricted, this leads to analytical difficulties. The reason for this is that having non-unique answers when only one of the classes can be a correct answer leads to difficulties in evaluation. The strategies for dealing with these problems can be categorized on the basis of whether the answers are unweighted or weighted[4].

4.1. Models For Systems With Unweighted Answers

In this situation non-unique answers arise because rules for more than one class are satisfied and the system does not have a strategy for picking among the classes. Sometimes, this indicates that the system is not properly constructed. For the purposes of evaluating such a system, the system designers must provide schemes for picking one of the multiple answers as the answer to be used in scoring the system. In effect, the system must be augmented with methods which act as filters to the system output and make it UA-restricted. A good example of this can be found in AQ15 [11]. In reporting performance of AQ15, Michalski and his colleagues describe the use of special routines of the kind mentioned above which force the system to be UA-restricted.

There are situations, however, when one could measure performance in the presence of multiple answers. For example, let there be 100 classes and for a particular case, suppose system A narrows its diagnosis down to 3 classes while system B narrows its diagnosis down to 20 classes. Let the correct conclusion be among the multiple answers of system A as well as those of system B. Clearly, system A has done better than system B. Such an evaluation is reported by Spackman [16] who compares the performance of his learning system CRLS with AQ15. Refer to Section 5.1.2. where the technique is discussed. However, when there are only two mutually exclusive classes, accepting multiple answers is awkward.

4.2. Models For Systems With Weighted Answers

When the system provides weighted answers, this can give rise to non-unique answers if the system itself does not select one of the classes as a conclusion. There are two main strategies for dealing with this situation.

4.2.1. Using Models Restricted To Mutual-Exclusivity And Unique Answers

Given the standard interpretation of confidence factors and given the knowledge that the classes are mutually exclusive, for purposes of scoring the system, the class in which the system has highest confidence can be picked as the C_{system}[5]. This makes the system *UA-restricted* and so the ME/UA restricted models can be used to score the system. For example, if for a case, a system gives

[4] Because the systems being considered are Not-UA restricted, confidence factors do make a difference now, as compared to the previous section where UA restricted systems were considered.

[5] There is the problem of how to handle *ties* - more than one class having the highest confidence, but this complication shall be ignored for the moment.

class1	0.7
class2	0.5
class3	0.0

Table 12

Table 12 as output, then, for the purposes of scoring the system over the case, *class1* will be picked as C_{system}.

4.2.2. Measurement By Averaging Weights

By picking the most certain answer, in effect, the uncertainty in the answers is being ignored. A situation where the correct conclusion has second-highest confidence may be better than a situation where the correct conclusion has fourth-highest confidence. By picking the class with highest confidence, both the above situations are marked as wrong and classified equivalently. This may not be desirable. The solution is to use techniques which measure partial correctness in these situations by averaging over the weights. Two such techniques are described below:

Measurement Using Distance Metrics

One way of measuring partial correctness is to use a closeness measure rather than forcing the system to choose between correct and incorrect classification. For N diagnoses, consider an n-dimensional space. Each axis is the confidence factor of a diagnosis (hence it is scaled from 0 to 1). The expert's conclusion is a point in this space and so is the system's conclusion. Consider an appropriate distance metric between these two points in this space. The mean or variation over all cases could be taken as a performance metric.

Thus, for example, consider the case of the earlier section. Let $C_{correct}$ for the case be *class2*. Since there are 3 classes, points in 3-dimensional space need be considered. Let *class1* lie along the *x-axis*, *class2* along the *y-axis* and *class3* along the *z-axis*. Then, $C_{correct}$ can be represented by the point *(0,1,0)*. Similarly, C_{system} is the point *(0.7,0.5,0.0)*. If the squared distance is taken as a metric, then the case is assigned a score of 0.74.[6] Once scores are calculated for a number of cases, the average squared distance can be reported as a performance metric. This kind of performance measurement using a distance metric is popular among neural net systems [15].

Measurement Using ROC Curves

This measurement strategy, described in [17], is useful in the two class situation. An ROC curve is a plot of sensitivity against 1-specificity. The values of specificity and sensitivity are calculated for various levels of decision confidence of the system. These levels are chosen along the confidence

[6] The smaller this score, the better the performance on the case.

range. The area under the ROC curve is calculated as an performance metric. A 100% accurate system having 100% sensitivity and 0% specificity, has an area of 1.0. This measure is independent of the occurrence of class members in the test data as well as of the decision bias of the system.

The strategy can be best illustrated by an example. Consider the data in Table 13 obtained from an expert system that classifies patients as having Rheumatoid Arthritis (RA) or not. 121 cases are classified by the system in four different categories. These categories are obtained by dividing the confidence range into four intervals and classifying cases into these four categories instead of the original two categories (RA or not-RA)[7]. This gives us the results of columns 2 and 3. The

Rating category	$C_{correct} = RA$	$C_{correct} \neq RA$	$Sensitivity$	$1 - Specificity$
definitely RA	21	4	0.50	0.05
probably RA	10	5	0.74	0.11
possibly RA	7	13	0.90	0.28
definitely not RA	4	57	1.0	1.0
total	42	79		

Table 13

sensitivity and specificity are calculated by the "rating procedure". This involves viewing each rating category as representing a threshold. All cases in higher categories are viewed as positive decisions. Thus, viewing *Possibly RA* as representing a threshold, we compute the following:

Number of true positive decisions = 21 + 10 + 7 = 38
Total number of positive decisions = 42
Number of false positive decisions = 4 + 5 + 13 = 22
Total number of negative decisions = 79
Sensitivity = True Positive decisions/Total Positive decisions = 0.90
1 - Specificity = False Positive decisions/Total negative decisions = 0.28

The other values in columns 4 and 5 are calculated similarly. Thus rating information is used as though decision thresholds were being used instead. Once we the results of columns 4 and 5, we now have enough information to plot five points. The resulting ROC curve is shown in Figure 3 with an area of 0.87.

5. Relaxing Mutual Exclusivity Of Classes

In the previous section, it was shown that relaxing the UA restriction while keeping the ME restriction leads to problems. Now the ME restriction is relaxed while still keeping the constraint of having a known undisputed conclusion for each case.

[7] The choice of four categories is arbitrary. It could have been anything else.

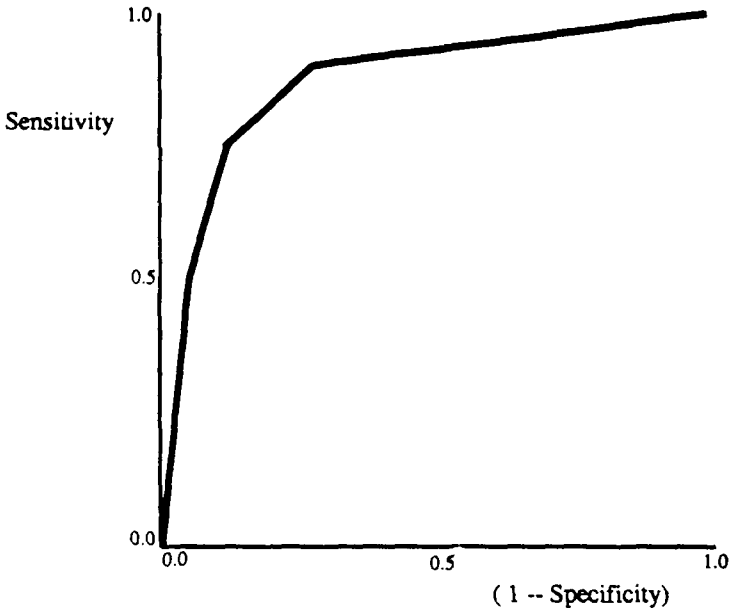


Figure 3: An ROC Curve

If the classes are not mutually exclusive, multiple classes may apply to the same case data. For example, a patient may have a bad throat, fever, diabetes and lung cancer at the same time. Any system that diagnoses this set of health disorders cannot make assumptions about mutual exclusivity of these classes. Thus, from Figure 2, the unique answer generator can be removed. The new situation is depicted in Figure 4.

It is immediately clear that a large part of performance measurement now rests with the correctness classifier. In the earlier models, since there was only one expert conclusion and one system conclusion for each case, the correctness classification was trivial – just check to see if expert's conclusion was equal to the system's conclusion or not. Now, however, the problem is complicated because there are multiple classes for the conclusions of both the expert system and the expert. How does one score a case for which the expert system and the expert agree of some conclusions and disagree on some others ? Any model that deals with measuring performance of such systems must necessarily address this problem and have a scheme for resolving it.

5.1. Models for Systems With Unweighted Answers

As before, the simplest situation is considered first. Assume that the correct conclusions are known and undisputed and that the system's conclusions are unweighted answers Note that $C_{correct}$ and C_{system} are sets of classes. Thus for a case one could have $C_{correct}$ = (*class1*, *class3*) and C_{system} = (*class1*, *class2*). The models described below are proposed as possible ways of scoring such systems.

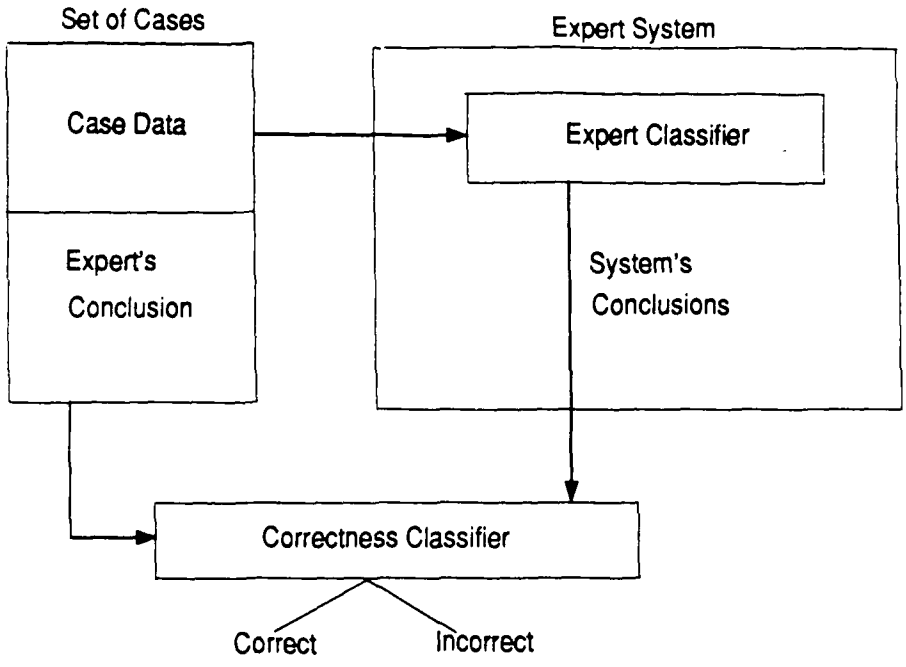


Figure 4: Measurement With Multiple Diagnoses

5.1.1. The Case Correctness Model

The simplest measure of performance is to measure how many *cases* are incorrect. The definition of an *incorrect* case may not be domain-independent. Consider the following examples:

1. If the domain is medicine and we are dealing with a set of diseases all of which are very expensive to test and at the same time all the diseases are life threatening, then for a case in question, the system must match the expert exactly in its conclusions. If it guesses more disease classes than the expert, then the further testing required to weed out the false positives would make the system not very cost effective. On the other hand, if the system does not guess all the diseases, then the patient will not have much confidence in the system. These criteria translate into the following correctness classifier:

   If $C_{correct}$ and C_{system} match exactly, then the case is correct or else it is incorrect.

2. Again consider a set of diseases which are extremely expensive to test for precisely. Assume that an expert system is being installed for checking the presence of these diseases during regular medical examinations. For the system to be effective and useful, the system must not raise false alarms which would necessitate the expensive testing. For such a situation, the following correctness criteria would be appropriate:

   If C_{system} is a subset of $C_{correct}$ then the case is correct or else it is incorrect.

Given that the correctness classifier classifies each case as correct or incorrect (possibly by some domain-dependent scheme), the percentage of errors can be computed as a measure of the system performance. The metric of percentage of errors is calculated as before:

$$\% \text{ of errors} = \frac{Incorrect}{Correct+Incorrect} \times 100.$$

In many situations it is often important to determine the degree of error. Otherwise one cannot distinguish between the performance of systems such as systems M and N listed in Table 14. Assume

	Conclusions	
System	Case A	Case B
Expert	*class1, class2*	*class2, class3*
System M	*class2*	*class1, class2, class3*
System N	*class3*	*class1*

Table 14

the correctness criteria is exact match. Although, both systems M and N got the cases wrong, it seems, intuitively, that System M did better than System N. This is not reflected at all in the measurement by case correctness.

5.1.2. The Partial Correctness Model

It was seen above that the case correctness model was relatively coarse. Another way of measuring correctness would be to employ some kind of closeness measure rather than to force the system to choose between correct and incorrect classification. This can be achieved by allowing partial correctness of cases. The following example illustrates this strategy. Suppose that a system reports performance on two cases as shown in Table 15. Thus for Case1, $C_{correct}$ contained 2 classes and

Case	$C_{correct}$	C_{system}
Case1	*class1, class2*	*class2, class3*
Case2	*class2, class3*	*class1, class2, class3*

Table 15

the system got 1 of them right, so it is given a score of 1/2 on correctness. While this handles the case of partial correctness on cases, it is inadequate unless it is coupled with some measure on how precise the system was. One such measure is the positive predictive value[8]. Thus, for the above example, we get the following two measures:

[8] The ratio of the number of classes correctly identified by the total number of classes in C_{system}.

$$\text{Accuracy} = \frac{1+2}{2+2} = 0.75$$
$$\text{Predictive Value } (+) = \frac{1+2}{2+3} = 0.60$$

These metrics were in fact used in [11] to measure performance of AQ15 for cases with multiple diagnoses.

5.1.3. The Diagnostic Performance Model

The case correctness model does not specify a system's performance relative to a class. This may be important for a rule-based system, for example, to determine the performance of the rules for a particular class (so that they can be refined). One way of measuring performance by diagnostic class is as follows:

Split each case into *p-cases* corresponding to the number of correct classes for that case. Thus, each *p-case* has one conclusion from the $C_{correct}$'s for the case which generated it. This is the $C_{correct}$ of the *p-case*. Now we measure performance for each *p-case*. The system classifies a *p-case* as correct if the $C_{correct}$ of the *p-case* is among the system's conclusions. The notions of False Positive and True Negative can be defined similarly.

Thus for example, consider a domain where there are three classes: *class1, class2* and *class3.* Consider the two cases shown in Table 16. Table 17 shows the performance measurement listed by

case number	$C_{correct}$	C_{system}
1	class1, class2	class1, class3
2	class1, class3	class2, class3

Table 16

diagnostic class. What has been done, effectively, is that the system is being viewed as a ME/UA

class	TP/H+	False Positives
class1	1/2	0
class2	0/1	1
class3	1/1	1
total	2/4	2

Table 17

restricted system. For each *p-case*, each of the C_{system}'s is the system's answer and is either equal to the one and only one $C_{correct}$ for that *p-case* or not. Thus, all the metrics discussed earlier for such systems are applicable here. Costs can be attached to the various false positive and false negative errors of each class in order to get a metric which measures performance more accurately. SEEK2 [6] uses this model for handling cases with multiple diagnoses.

The drawback of this model is that it cannot be used to estimate the performance of the system over *cases*. If one tries to do so, the performance measures will be biased towards cases with more multiple conclusions.

5.2. Measurement Models For Handling Weighted Answers

So far non-ME restricted systems without uncertainty have been examined. This simplifies the interpretation of system output. Any class that gets fired by the system rules, is a part of C_{system}. In this section scoring of systems with weighted answers for non-mutually exclusive classes is considered. In an earlier section, the effects of adding an uncertainty model to systems classifying among mutually exclusive classes were analyzed. The main problem had been in defining what should be considered as C_{system}. The same problem is encountered when the classes are not mutually exclusive. It is assumed that the expert's conclusions are undisputed and definite, and the system produces an ordered list of classes (ordered by some measure like confidence factor CF). The CF attached to a class reflects the system's belief that the case can be classified in the class. For example, consider a case for which the expert's conclusions are *(class1, class2)*. Let the system produce the following list of classes ordered by confidence factors: *((class1, 0.9) (class3, 0.7) (class2, 0.4) (class4, 0.3))*. Leaving aside $C_{correct}$ for the moment, the difficulty is in assigning an interpretation to C_{system}. There are several strategies that can be used here, each giving rise to a different scoring scheme:

1. One strategy might be to ignore certainty factors for performance measurement. Thus, for the above example, one would determine C_{system} = (class1, class2, class3, class4). All classes with non-zero CF's would be considered as the system's answers. This strategy ignores the reasoning under uncertainty employed by the system and is thus not a very precise measure of performance.

2. Another alternative would be to ask the domain expert to define what an answer should be in terms of the confidence factors of the system. This would depend on how the confidence factors are being interpreted by the domain experts. Thus, one could decide to treat every class with a CF greater than 0.5 as a part of C_{system}. For the example, we get C_{system} = ((class1, 0.9) (class3, 0.7)). Having used the CF's to determine what should be the answer, they are ignored for scoring the system response. In general, there can be no domain-independent way of using CF's to fix thresholds for answers because the confidence factors have different interpretations in different domains. Thus, by this strategy, the system can use uncertainty models for reasoning but is obligated to also provide a mechanism for determining acceptable answers for performance measurement.

3. Another strategy might be to thrust the responsibility of interpreting the system's output to the correctness classifier. Thus the correctness classifier looks at the system output (which is a list of classes with confidence factors) and the $C_{correct}$ (which is a set of classes) and determines whether the system's output is acceptable as an answer or not. One way the

correctness classifier might do so is to count the number of classes in $C_{correct}$ and accept an equal number of classes from the system's output as C_{system} and then do the performance measurement using any of the schemes described for the situation when C_{system} is definite. Thus, for the example, since $C_{correct}$ contains two classes, the two classes with highest CF would be assumed to be the system's answer and C_{system} would be taken to be (class1, class3). This is the strategy used by SEEK2 [6] in scoring cases with multiple diagnoses for the AI/RHEUM knowledge base [8] which uses an uncertainty model.

4. Another way of measuring correctness would be some kind of closeness measure rather than to force the system to choose between correct and incorrect classification. This has been discussed earlier for systems with mutual exclusivity and non-unique answers. The technique is applicable even when the classes are not mutually exclusive.

6. Measurement When True Case Conclusion Not Known

In the preceding sections, techniques for measuring performance were described for conclusions that are undisputed and known. In some domains the answer for each case is not truely known. The expert himself may be wrong and so his conclusion may not be the true conclusion. Also, two expert may give differing conclusions for the same case data. The problem here is how does one find the *gold standard*, the conclusion against which to compare performance of the system. Several methods for solving this problem can been used:

Find the true conclusion This means that the expert opinion is ignored and an attempt is made to get the real answer by observing real-world events. For example, if one is building a system which forecasts the next day's weather, then the true conclusion can be found by waiting until the next day and observing the weather. As another example, consider a system trying to determine life expectancy of cancer patients.

Compare with human expert's performance Sometimes it may not be possible to obtain the true conclusion. In such situations, the strategy used is to measure the performance by comparing with the performance of human experts.

This strategy was used to validate a version of MYCIN in the domain of meningitis [21]. For each of ten cases, six treatment recommendations were obtained – one was obtained using MYCIN, one was the actual therapy used in treating the patient and the other four were obtained from experts at four different experience levels. For each case, the six recommendations were randomly ordered and presented to eight outside experts. Each outside expert gave his own recommendation and rated the six recommendations as identical to his own, acceptable or unacceptable. A majority of the evaluators approved MYCIN's choice 70% of the time. None of the other four experts achieved better approval than MYCIN.

Allow experts to attach confidence factors What if the expert refuses to commit himself ? Or if he cannot commit himself ? The strategy here is to allow the expert to attach confidence

factors to his conclusions. This strategy is particularly useful if the performance data is not very extensive, and the expert attaches confidence factors which are known to be compatible with the confidence ratings assigned by the system.[9] For each case, one obtains the conclusions from the expert with confidence factors for each class. For each class the signed difference between the CF value of system and the expert is computed. For a case, the average difference is computed over all classes to obtain a metric of performance over a case. Alternatively, the average difference could be computed for each class over all cases to obtain a metric of performance with respect to each class. Thus, for example, consider the data in Table 18 for a single case. The average difference between the confidence values for the above data

class	$CF_{correct}$	CF_{system}	Difference
class1	0.80	0.61	0.19
class2	0.40	0.50	-0.10
class3	0.30	0.30	0.00

Table 18

is 0.03 which is 3% of the 0 to 1 scale. Such a performance measurement was reported for PROSPECTOR [3].

7. Conclusion

This paper has explored the various issues involved in measuring expert system performance on the basis of errors made on sample case data. The simplest scenario was considered first: when the conclusion for each case is known and undisputed, the classes are mutually exclusive and the system generates a single answer. These assumptions were then relaxed one by one and various approaches used to the handle the problems arising therein were analyzed. Most of them have been shown to be efforts to make the assumptions applicable again. From the point of view of performance measurement, four features of expert systems were identified as interesting: (1) Unique Answer, (2) Mutual Exclusivity, (3) True Conclusion Known and (4) Weighted Answers. The categorization of measurement models has been done over these four features. In Table 19, some expert systems have been classified on the basis of these features. In the table, ME, UA and TCK and WA refer to the four features. A *y* refers to the presence of the feature and a *n* refers to its absence. Some combinations of these features are awkward. Thus, it is awkward to consider systems where the classes are not mutually exclusive but the system always gives a unique answer. Note that some of the systems mentioned in the table have been designed to handle cases from more than one category. This is because some categories are broader than others. Thus, systems which handle

[9] This would be the situation if a model of expert reasoning was being accurately designed and it was known that the expert can give indefinite answers. So the uncertainty model being used would have to conform to the uncertainty model used by the expert himself.

System	UA	ME	TCK	WA
INTERNIST-1 [13]	*n*	*n*	*y*	*y*
MYCIN [21]	*n*	*n*	*n*	*y*
PROSPECTOR [3]	*n*	*n*	*n*	*y*
AI/RHEUM [8]	*n*	*n*	*y*	*y*
AQ15 [11]	*y*	*y*	*y*	*n*
CRLS [16]	*n*	*y*	*y*	*n*
SPE [19]	*n*	*n*	*y*	*n*
CASNET [20]	*n*	*n*	*n*	*n*
PUFF [9]	*n*	*n*	*n*	*n*
Decision Trees [1, 14]	*y*	*y*	*y*	*y*
Classical Pattern Recognition [4]	*y*	*y*	*y*	*y*

Table 19 Categorization of some expert systems

cases with non-unique answers, can also handle cases with unique answers. Systems have been classified into the broader category that covers the cases over which measurement is likely to be done. Besides specific systems, two classes of systems have been categorized. The classical pattern recognition algorithms referred to are *supervised learning* algorithms. Note that classical pattern recognition algorithms and neural net systems tend to follow the simplest model of measurement. Medical expert systems, on the other hand, tend to diverge from the simplest model. Consequently, they need a more complex representation and are more difficult to evaluate.

Given the above categorization, one can more readily determine which set of measurement models are appropriate for a given medical expert system. Determining the underlying performance measurement model is critical in determining which scoring strategy is appropriate.

References

[1] Breiman, L. et al. (1984). *Classification and Regression Trees*, Wadsworth, Inc., Belmont, California.

[2] Clancey, W. (1984). *Classification Problem Solving*, Proc. of AAAI-84. Pgs 49-55.

[3] Duda, R.O. et al. (1979). *Development of the Prospector Consultation System for Mineral Exploration*, Final Report SRI Project 6415, SRI International, Menlo Park, California.

[4] Duda, R.O. and Hart, P.E. (1973). *Pattern Classification and Scene Analysis*, John Wiley and Sons, Inc., New York.

[5] Galen, R. and Gambino, S. (1975). *Beyond Normality: The Predictive Value and Efficiency of Medical Diagnoses*, John Wiley and Sons, Inc., New York.

[6] Ginsberg, A., Weiss, S.M. and Politakis, P. (1988). *Automatic Knowledge Base Refinement for Classification Systems*, Artificial Intelligence 35 (1988). Pgs 197-226.

[7] James, M. (1985). *Classification Algorithms*, John Wiley and Sons, Inc., New York.

[8] Kingsland III, L.C. (1985). *The Evaluation of Medical Expert Systems: Experience with The AI/RHEUM Knowledge-based Consultant System in Rheumatology.* Proc. Ninth Annual Symposium on Computer Applications in Medical Care. Pgs 292-295. IEEE Computer Society Press, Washington D.C.

[9] Kunz, J.C. et. al. (1978). *A Physiological Rule-based System for Interpreting Pulmonary Function Test Results.* Rept. HPP-78-19, Heuristic Programming Project, Computer Science Dept., Stanford University, Stanford, CA.

[10] Laird, J. (ed). *Proc. of the fifth International Conference on Machine Learning 1988*, Ann Arbor, Michigan.

[11] Michalski, R.S., Mozetic, I., Hong, J. and Lavrac,N. (1986). *The AQ15 inductive learning system: An overview and experiments.* Tech. Rept. ISG 86-20, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, Urbana, Illinois.

[12] Miller, P.L. (1986). *The evaluation of artificial intelligence systems in medicine*, Computer Methods and Programs in Biomedicine 22 (1986). Pgs 5-11.

[13] Miller, R.A., Pople, H.E. and Meyers, J.D. (1982). *INTERNIST-1, An experimental computer-based diagnostic consultant for general internal medicine.* N. Engl. J. Med. 1982. Pgs 307-468.

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

[15] Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986). *Learning Internal Representations by Error Propagation* in D.E. Rumelhart and J.L. McClelland (Eds.), "Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations". MIT Press.

[16] Spackman, K. (1988). *Learning Categorical Decision Criteria in Biomedical Domains.* Proc. of the fifth International Conference on Machine Learning. Pgs 36-46. Univ. of Michigan, Ann Arbor, Michigan.

[17] Swets, J.A. (1988). *Measuring the Accuracy of Diagnostic Systems.* Science (3) Jun 1988. pgs 1285-1293.

[18] Weiss, S.M. and Kulikowski, C. (1984). *A Practical Guide to Designing Expert Systems.* Rowman and Allanheld, Totowa, New Jersey.

[19] **Weiss**, S.M., **Kulikowski**, C.A. and Galen, R. (1981) *Developing Microprocessor-based Expert Models for Instrument Interpretation.* Proc. of IJCAI-81. Pgs 853-855.

[20] Weiss, S.M. et. al. (1978). *A Model-based Method for Computer-aided Medical Decision-making.* Artificial Intelligence (11) 1978. Pgs 145-172.

[21] Yu, V.L. et. al. (1979). *Antimicrobial selection by a computer: A blinded evaluation by infectious disease experts.* J. Amer. Med. Assoc. 242 (1979). Pgs 1279-1282.