Combining Symbolic and Connectionist Learning Methods to Refine Certainty-Factor Rule-Bases

by

J. Jeffrey Mahoney, MS, MM, BA

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May, 1996

To all the great people who have been such tremendous sources of inspiration, and have so significantly enriched my life. These include Mariah Carey, John Coltrane, Edsgar Dijkstra, Albert Einstein, Annette Funicello, Judy Garland, Anatoly Karpov, Stan Kenton, David Letterman, Robert Mahoney, John Newcombe, Arnold Palmer, Charlie Parker, Dave Pelz, Harvey Penick, Oscar Peterson, Monica Seles, Tracy York, Jeff Vandiver, Pepper Von, Zig Ziglar, as well as all of the beautiful women of Austin aerobics! To you this work is dedicated...

Acknowledgments

I am indebted to many people for helping me with the research presented here, as well as with the completion of this document. I must first acknowledge Ray Mooney for his assistance, insights, persistence, and unending patience in guiding me through this most trying process. Thanks also go to Geoff Towell of Seimens Corporate Research, Jude Shavlik, Mark Cravens, and David Opitz of The University of Wisconsin for their help with the DNA datasets, as well as with KBANN results. Also thanks to Yong Ma of The University of Illinois for supplying the MYCIN rule base. Finally, I must thank many of my colleagues for their support, proofreading, as well as keeping my spirits high when things appeared low. Special thanks go out to Paul Baffes, John Zelle, Dan Clancy, Cindi Thompson, Dave Moriarty, Siddarth Subramanian, Xiang-Seng Lee, Hiow-Tong See, Mike Hewett, Tara Estlin, Sowmya Ramachandran, and Bert Kay. I owe each of them a round of golf! This research was supported in part by the National Science Foundation under grant IRI-9102926, the NASA Ames Research Center under grant NCC 2-629, and the Texas Advanced Research Program under grant 003658114.

J. JEFFREY MAHONEY

The University of Texas at Austin
May 1996

Combining Symbolic and Connectionist Learning Methods to Refine Certainty-Factor Rule-Bases

Publication No. ____________________

J. Jeffrey Mahoney, Ph.D.
The University of Texas at Austin, 1996

Supervisor: Raymond J. Mooney

This research describes the system RAPTURE, which is designed to revise rule bases expressed in certainty-factor format. Recent studies have shown that learning is facilitated when biased with domain-specific expertise, and have also shown that many real-world domains require some form of probabilistic or uncertain reasoning in order to successfully represent target concepts. RAPTURE was designed to take advantage of both of these results.

Beginning with a set of certainty-factor rules, along with accurately-labelled training examples, RAPTURE makes use of both symbolic and connectionist learning techniques for revising the rules, in order that they correctly classify all of the training examples. A modified version of backpropagation is used to adjust the certainty factors of the rules, ID3's information-gain heuristic is used to add new rules, and the Upstart algorithm is used to create new hidden terms in the rule base.

Results on refining four real-world rule bases are presented that demonstrate the effectiveness of this combined approach. Two of these rule bases were designed to identify particular areas in strands of DNA, one is for identifying infectious diseases, and the fourth attempts to diagnose soybean diseases. The results of RAPTURE are compared with those of backpropagation, C4.5, KBANN, and other learning systems. RAPTURE generally produces sets of rules that are more accurate than these other systems, often creating smaller sets of rules and using less training time.

Contents

Chapter 1

Introduction

In nearly all walks of life, there exist problems that require some degree of expertise in order to solve. This includes things from as simple as how to effectively remove a spot from the living room carpet or which college courses to take during the freshman year, to more complex problems such as how to repair a malfunctioning automobile or which medication to take to remedy an ailment. When problems such as these occur, a good strategy is to consult with an expert in the particular problem area (domain). This expert will generally have a great deal of experience in solving the problem at hand, and will ideally be able to offer an effective solution.

Unfortunately, for many complex domain areas, such experts are often either in short supply, very expensive, or non-existent. One obvious solution is to attempt to automate the expertise, and make it readily available. The ability to precisely specify necessary problem-solving expertise, and put it "in a bottle" would be of significant value (both economic and social) to everyone.

This desire has led to a great deal of research into developing *expert systems*(Buchanan & Shortliffe, 1984), which are simply automated systems that aid, enhance, or replace the human expert. The use of expert systems has had a significant impact upon everyday society for a number of years, and is helping many corporations achieve higher profits and productivity (Feigenbaum, McCorduck, & Nii, 1988).

The major difficulty in creating such systems, however, is in properly extracting the necessary information from the expert. Not only is it difficult for experts to translate their reasoning abilities into concrete sets of rules, but once done, the result is usually inaccurate and inconsistent. This is no reflection on the expert, but rather on the difficulty of the rule-base construction problem. This difficulty has become prevalent enough to have been named the *knowledge-acquisition bottleneck* (Feigenbaum & McCorduck, 1983). This name refers to the fact that the bottleneck in creating an effective expert-system lies in the acquisition and specification of the knowledge.

1.1 The Knowledge-Acquisition Bottleneck

A real-world example of this bottleneck comes from the IRS. Out of an ongoing attempt to automate the task of processing tax-returns as much as possible, a team of five was given the duty of precisely specifying the rules that were currently in place for identifying tax returns that needed to be audited. In other words, given a tax return, specified by key entries on the form, decide whether or not the return should be audited. Once created, this set of rules would enable an automated

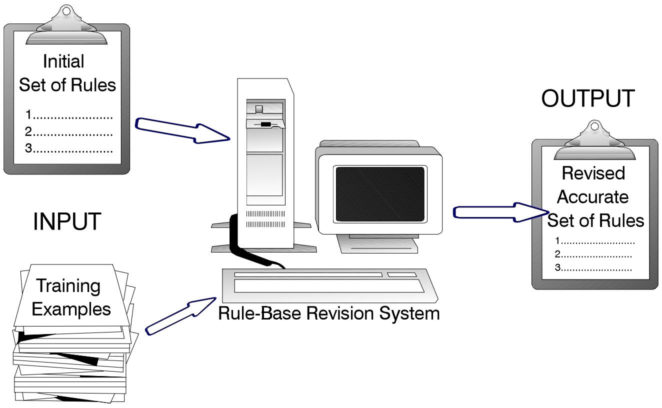

Figure 1.1: Rule Base Revision Overview

system to quickly and accurately process these returns and decide which ones needed more careful examination.

The team members individually had little difficulty in determining the proper classification of each return (i.e., audit or non-audit), and there was little disagreement. Much to their surprise, however, the task of writing down the rules that coincided with their decision-making process proved quite difficult. It took them slightly more than two years to complete the task, and they admitted that the resulting rules were still not perfect, though they were able to correctly classified returns over 90% of the time (Davis, 1995).

Out of an attempt to alleviate this bottleneck, researchers (Ginsberg, 1988; Mooney & Ourston, 1991; Towell, Shavlik, & Noordewier, 1990; Pazzani & Kibler, 1992) have sought ways of automating the process of fine-tuning a rule base. Throughout this document, I refer to this process as *rule-base revision*, though other authors have called it *theory revision* (Mitchell, Keller, & Kedar-Cabelli, 1986; Baffes & Mooney, 1992; Cain, 1991; Ginsberg, 1990), or *theory refinement* (Buntine, 1991; Ourston & Mooney, 1994). I prefer the former in an attempt to avoid any confusion with the scientific usage of the word *theory*.

Given a set of rules, the rule-base revision task is to rapidly revise these rules in a way that improves their performance on a given domain. By automating this task, orders-of-magnitude gains in completion time are expected, greatly relieving the bottleneck.

This process is applicable to any sets of rules that are capable of reaching conclusions. Possible examples include rules for picking football winners, forecasting the weather, buying a car, or diagnosing diseases. Rule bases can come from any domain where expert knowledge is of use in the decision making process.

1.2 Rule-Base Revision

Research in the area of rule-base revision has grown out of the related areas of inductive learning (Quinlan, 1993; Langley & Simon, 1995) and explanation-based learning (DeJong & Mooney, 1986; Mitchell et al., 1986). Induction attempts to build decision-making systems from scratch, using a set of examples of target concepts to guide the process. There is no use of background knowledge in solving the problem. Research has shown, however, that biasing learning with expert-supplied knowledge leads to better performance on novel examples (Ginsberg, 1990; Pazzani & Kibler, 1992; Ourston & Mooney, 1994; Towell & Shavlik, 1994). This is especially true in complex domains where training data is limited. Though, as has been noted, experts have difficulty in precisely specifying their expertise, they are very effective at identifying essential bits and pieces of their domain knowledge. Because of this, many revision systems attempt to modify the expert supplied knowledge as little as possible, only modifying rules when they fail to properly classify an example.

Recently, both connectionist and symbolic methods have been developed for providing learning systems with this knowledge bias (Towell & Shavlik, 1994; Fu, 1989; Ourston & Mooney, 1994; Pazzani & Kibler, 1992; Cohen, 1992). Many of these methods revise an imperfect rule base to fit a set of empirical data. Some of these methods have been successfully applied to real-world tasks, such as recognizing promoter sequences in DNA (Towell et al., 1990; Ourston & Mooney, 1990; Thompson, Langley, & Iba, 1991). These results demonstrate that revising an expert-given rule base produces more accurate results than learning from training data alone.

This process of rule-base refinement to correctly classify a set of training examples has proven to be an important task that lends itself to integrating analytical methods, that is methods that rely on theory, and empirical learning methods, that is methods that have demonstrated their effectiveness through example. Though several refinement systems have been successfully applied to real-world problems (Ginsberg, 1990; Ourston & Mooney, 1994; Towell & Shavlik, 1994), to date these have focused largely on revising logical Horn-clause theories. Many real-world domains, however, require some type of probabilistic or uncertain reasoning (Shafer & Pearl, 1990). The primary advantage of these methods is their ability to combine evidence from several different sources and draw conclusions based on the total combined evidence for each possible decision. Consider the following hypothetical rule, expressed in standard logical notation

$$a \wedge b \wedge c \wedge d \rightarrow C_1$$

for classifying an instance into category C_1. An expert looking at the same problem might, however, still be inclined to categorize a given instance into C_1 despite lack of evidence for d, provided sufficient evidence existed for $a \wedge b \wedge c$. This is particularly true in cases where instances do not exactly fit into *any* predefined category. An automated classification system using pure Horn-clause logic can not easily reason in this manner, since lack of evidence for d results in falsifying the antecedent condition, and no conclusion can be drawn (from this rule). In order for a Horn-clause based system to model this kind of reasoning, allowing one or more antecedents to be false, and still providing evidence for the conclusion, would require an exponential number of Horn-clauses. In this way, each Horn-clause would represent one of the possible combinations of true antecedents. This is clearly an undesirable solution since it greatly increases the size of the rule base, makes it more difficult to revise, as well as less understandable for the human.

Flexible matching methods, which have been used successfully in inductive rule-learning (Michalski & Chilausky, 1980; Michalksi, Mozetic, Hong, & Lavrac, 1986) and rule-base revision (Mooney & Ourston, 1991), are one way of dealing with this problem. In this approach, a score is calculated measuring how well an example matches each symbolic rule, and the rule with the greatest score is invoked. Another approach is to use M-of-N rules (Towell & Shavlik, 1991; Ginsberg, 1988), which fire if at least M of their N antecedents are satisfied, e.g. 3-of $a, b, c, d \rightarrow C_1$. Unfortunately, this approach does not consider the relative strengths of each of the antecedents. Neural networks on the other hand, use connection weights to encode relative strengths. By representing rules as a neural network, standard backpropagation (Rumelhart, Hinton, & Williams, 1986) can be used to modify the weights representing rule strengths (Fu, 1989; Towell & Shavlik, 1994; Lacher, 1992). Since a unit's activation-level depends upon a linear sum of all incoming activations, this is an effective approach to combining evidence.

1.3 Our Solution to the Bottleneck

In this dissertation, the system RAPTURE is described, which is an acronym for *Revising Approximate Probabilistic Theories Using Repositories of Examples*. This system combines both symbolic and neural learning methods. RAPTURE begins by converting a symbolic rule-base into a connectionist network. RAPTURE requires that rules be initially expressed using certainty factors (Buchanan & Shortliffe, 1984). Literals in the rules map to units in the network, and the certainty factors become the weights on the connections between units. Unlike standard neural networks, in which the total input to a node is determined by a linear sum of all incoming activations, for a certainty-factor network, the total input is the *probabilistic sum* of incoming activations. This is calculated using the formula $x + y - xy$ for pairs of values. This is the standard formula for calculating the probability that either of two events will occur, given their individual probabilities. The two probabilities are added, and then the cross-product term is subtracted, since it has been added twice. No thresholding output function is needed for units in the certainty-factor network, since the probabilistic sum already provides the required non-linearity.

Next, the network is modified to correctly classify a set of training examples. Network training is performed in three phases. First, a modified version of backpropagation (Rumelhart et al., 1986) is used to adjust the certainty-factors on existing rules. The normal backpropagation equations are modified in order to perform gradient descent for certainty-factor combining functions (e.g., probabilistic sum, min, and max). This is similar to methods employed in Fu (1989) and Lacher (1992), and throughout this text I refer to this technique as Certainty-Factor Backpropagation (CFBP), though others have called it expert-network backpropagation (Lacher, 1992). Once all of the training examples are classified correctly, the network is considered trained, and learning is complete.

It is often the case, however, that CFBP plateaus before achieving 100% accuracy on the training data. When this occurs, architecture modification routines are called. These routines borrow from both symbolic and connectionist methods common to machine learning. The first of these is the feature-addition routine. Specifically, features are added into the network that best discriminate between incorrectly handled examples using ID3's information gain (Quinlan, 1986). Backpropagation and feature addition continue in a cycle until either all of the training examples are correctly classified, or training accuracy again reaches a plateau.

In the latter case, a call to the final RAPTURE routine is made. This is a call to the UPSTART algorithm (Frean, 1990). UPSTART is a connectionist technique for building new hidden nodes in a neural network, and can be thought of as creating new hidden features for the rule-base. For each output unit that is incorrectly classifying examples, two new hidden units are created. One of these is designed to learn the false-negative examples, and the other the false-positives examples. These new units are trained using the RAPTURE algorithm recursively, and linked appropriately to their corresponding output units.

One elegant feature of the RAPTURE algorithm is that once a network is trained, the revised rules can be read directly off of the network. Unlike the KBANN system (Towell & Shavlik, 1992) (see section 2.5.3), where revised networks are mapped back into rules to improve the comprehensibility of the final result, no retranslation in necessary for RAPTURE. This is because of the direct correspondence between weighted links and probabilistic rules which removes any distinction between the symbolic and connectionist representations. They are equivalent ways of looking at the same information. This approach therefore combines the effectiveness of connectionist learning methods with the interpretability of rules. Comprehensibility is important since it has been found that users will generally not accept a system's conclusions unless it can present meaningful explanations for them (Swartout, 1981).

RAPTURE has been tested on revising several real-world knowledge bases with encouraging results. In particular, we present results for revising rule bases for recognizing promoter sequences in strands of DNA (Towell et al., 1990), identifying splice-junction sites in DNA (Towell & Shavlik, 1994), diagnosing diseased soybeans (Michalski & Chilausky, 1980), and diagnosing bacterial infections in humans (Buchanan & Shortliffe, 1984). This last domain is based on a version of the MYCIN knowledge-base, which RAPTURE has successfully revised. We compare our results to those obtained for purely inductive methods (C4.5 (Quinlan, 1993), standard backpropagation, and RAPTURE given no initial knowledge), a purely connectionist method for knowledge-base refinement (KBANN), a purely symbolic method for knowledge-base refinement (EITHER (Ourston & Mooney, 1994)), as well as various RAPTURE ablations which demonstrate the effects of each of the components. Results demonstrate that RAPTURE generally produces more accurate results from fewer training examples than competing approaches, as well as producing less complex sets of rules.

1.4 Thesis Statement

We have created the rule-base revision system, RAPTURE, as an aid towards the development of successful expert systems. In order for an expert system to be successful, however, it must meet certain criteria. Without the criteria listed here, it may be of only limited use.

First and foremost, successful expert-systems must perform at a very high level. If they are unable to perform the task as well as, or nearly as well as an expert, and certainly better than a non-expert, they will be of little value.

Second, they must be readily available for use. If an expert system remains in the development and debugging stages for long periods of time, its usefulness will undoubtedly be limited.

Third, a successful expert-system should be reasonably understandable. If the system is filled with rules that make little or no sense, rules that are clearly relying on insignificant observations, or rules that are difficult to comprehend, there will be little confidence in any results that are produced.

Finally, an expert system should be as simple as possible. This does not mean that all expert systems should be trivial, but rather that they should not be needlessly complex. If rules can be represented in an efficient manner, a more understandable system results.

The claim of this thesis is that by using the described method, such expert systems will result. By carefully making use of symbolic and connectionist machine-learning techniques, such expert systems can be created much more rapidly and more accurately than can be done manually. Beginning with "crude" expertise, using an uncertain-reasoning framework, and attempting to modify the original expertise only as dictated by training examples, a rule-base system can be developed that is more accurate, less complex, more easily understood, and more quickly obtained than other approaches.

1.5 Overview of Experimental Results

One of the strengths of the research presented here is the thoroughness of testing. Whereas other systems may be content to present results from one or two "toy" domains, we present in Chapter 4 extensive results from four real-world domains. These are all rule bases designed by domain experts in an attempt to solve real problems. Results demonstrate that RAPTURE performs at least as well, and usually better than any other current system.

RAPTURE was tested in two domains involving DNA sequencing. One of these is designed to recognize promoter sequences, and the other to recognize splice-junction sites. In both of these domains, RAPTURE outperforms all of the other systems, with the possible exception of KBANN. Results suggest, however, that RAPTURE is producing significantly smaller rule bases, and likely requires less time to train.

In the experiments involving the MYCIN rule-base, which is designed to diagnose infectious diseases, RAPTURE outperforms all other systems. This is clear evidence of the advantage given by the background knowledge. Training time is somewhat slowed down in this domain by the extensive usage of the UPSTART algorithm, which results in slightly larger rule bases than RAPTURE without UPSTART.

In the soybean disease-diagnosis experiments, RAPTURE performs significantly better than other systems when there are fewer numbers of training examples. This advantage is the result of the background knowledge. With large numbers of training examples, however, most systems perform at nearly equivalent levels.

1.6 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 presents relevant background information, and Chapter 3 discusses the details of the RAPTURE algorithm. Chapter 4 presents results on revising several real-world knowledge bases, Chapter 5 discusses related work, Chapter 6 discusses future work, and Chapter 7 presents the conclusions.

Chapter 2

Background

This chapter presents some of the background information that is assumed throughout the remainder of this document. This includes insight into human decision-making, reasoning under uncertainty, and the certainty-factor formalism upon which RAPTURE is based. Also presented is an overview of BACKPROPAGATION, and two recent revision systems—KBANN and EITHER.

2.1 Human Decision-Making as Reasoning under Uncertainty

One of the assumptions of this research is that reasoning under uncertainty is the method by which humans most commonly make everyday decisions. It is rare when one makes a decision with 100% certainty that no error has been made. Though it is possible to become quite confident of a decision after careful analysis of the various options, there will usually remain some doubt that perhaps another course of action would have produced better results. Examples include a physician diagnosing a patient, a young couple deciding on a new house, or a jury determining the guilt or innocence of O.J. Simpson. Typically such decisions begin by examining all of the available evidence. Evidence supporting or refuting each particular conclusion is collected, examined, and weighed against all evidence supporting or refuting other possible conclusions. Whenever a preponderance of evidence supporting one particular conclusion exists, a decision can be made with a great deal of confidence. Less preponderance of evidence results in less confidence in any given conclusion.

Looking at the above examples, it is noted that the doctor must carefully examine each patient, and observe symptoms as they present themselves. Each observed symptom will cause certain classes of diseases to be considered, and others to be ruled out. The doctor's diagnosis is further aided by background knowledge, which should include a medical history of the patient and immediate family, as well as a working knowledge of diseases that may be currently prevalent in the community. As more evidence is gathered, certain patterns should begin to present themselves, and ideally a successful diagnosis can be made.

This is very similar to the way a young couple decides on the purchase of their first house. Each house they consider will have features that they like, as well as those that they dislike. These pros and cons must be carefully weighed in order to come to a successful decision.

Finally, the decision-making process used by the jury deciding the fate of O.J. Simpson is

Figure 2.1: Weighing all of the evidence

considered. This case was 1995's "trial of the century," and due to the extensive publicity of this case, I will assume that all readers are familiar with the basic decision that had to be made by the jury. Much of their time was spent hearing evidence presented by trial lawyers. The evidence presented by the prosecution was designed to increase belief that Mr. Simpson did in fact commit the crimes in question. It was their goal to demonstrate that the accumulation of evidence removes reasonable doubt as to the guilt of the accused. The defense, on the other hand, presented evidence that either refuted the claims of the prosecution (which decreases belief in the *guilt* of the accused), or presented their own evidence that is designed to increase belief that the accused is *innocent.* The difficult task of the jury was to weigh all of this evidence, and make their own determination as to which evidence was the strongest.

It is worth noting that in each of these examples, there have been no hard and fast rules for reaching a conclusion. A rule such as

$$\begin{gathered}(\textsf{has-motive OJ}) \wedge (\textsf{has-opportunity OJ}) \wedge (\textsf{has-weak-alibi OJ}) \wedge \\ (\textsf{has-history-of-abuse OJ}) \wedge (\textsf{physical-evidence-linking OJ}) \wedge \\ (\textsf{witness-saw OJ}) \rightarrow (\textsf{GUILTY OJ})\end{gathered}$$

may seem reasonable at first glance, but is actually far too inflexible to be of any value. The antecedent condition is far too specific, and will fail to be true if any one the conjuncts is false. If for example, there were no witnesses, the final condition is false, and the rule fails to provide any evidence towards a verdict. If this were this the only rule for determining guilt, the accused must go free, despite any other evidence linking him to the crime scene. This does not seem to coincide well with the manner that a human comes to a conclusion.

Finally, it should be noted that there has been no mention of probability. In each of the above scenarios, there is no apparent concern over the probability that a correct decision has been made. The only important factor is the weighing of the evidence. Whenever the weight of evidence for one conclusion outweighs all others, this conclusion is deemed to be the best one.

2.2 The Certainty-Factor Formalism

Certainty-factors are one approach to modeling the uncertain-reasoning method described above, and is the approach used in the RAPTURE system. Certainty-factors grew out of MYCIN—an expert system for diagnosing infectious diseases, which was a part of the Stanford Heuristic Programming Project (Buchanan & Shortliffe, 1984).

All MYCIN rules are of the form $A \stackrel{0.8}{\rightarrow} D$, indicating that belief in proposition A gives additional reason for believing in proposition D. Unlike Horn-clause rules where antecedents prove consequents, certainty-factor rules work towards increasing or decreasing the level of belief in a particular proposition. The amount of increase or decrease of this belief is indicated by the certainty-factor (number) associated with each rule.

These certainty-factors are real-valued numbers that range from -1.0 to $+1.0$, and indicate the degree to which belief in the antecedent propositions effects belief in the consequent proposition. The greater the magnitude of the certainty-factor, the greater the effect. A certainty-factor of $+1.0$ indicates that the consequent is definitely true given its antecedents, whereas one of -1.0 indicates that the consequent is definitely false. A certainty-factor of 0.0 indicates that the antecedents have no bearing upon belief in the consequent. Examples include:

$$\mathsf{Age}{>}\ 21 \stackrel{+1.0}{\rightarrow} \mathsf{Of\text{-}Drinking\text{-}Age} \qquad \mathsf{Age}{<}\ 18 \stackrel{-1.0}{\rightarrow} \mathsf{Of\text{-}Drinking\text{-}Age}$$

$$\mathsf{Hair\text{-}Blonde} \stackrel{0.0}{\rightarrow} \mathsf{Of\text{-}Drinking\text{-}Age}$$

The last example indicates that the color of one's hair has no bearing on being of drinking age (and is therefore a superfluous rule).

Certainty-factor rules allow incremental updating of beliefs based upon newly acquired evidence. As an example, assume that there is currently no evidence for a belief in proposition D (e.g., has certainty-factor 0.0). Observing A to be true, along with the rule $A \stackrel{0.8}{\rightarrow} D$, changes belief in D to 0.8. This is calculated by multiplying the certainty-factor of A (1.0) by the certainty-factor of the rule (0.8). If A is observed to be only partially true, lesser belief in D results. If, for example, A's certainty-factor is 0.5, the rule gives D a measure of belief of 0.4 ($= 0.5 \times 0.8$).

Furthermore, several rules can independently offer evidence for D, allowing evidence to accumulate. Assume that D currently has a 0.4 certainty-factor, along with the existence of a second rule, $B \stackrel{0.6}{\rightarrow} D$. If B is observed true, this rule gives an increased belief in D of 0.6. Evidence from separate rules combine using probabilistic sum, which is defined (for positive evidence) as $a \oplus b \equiv a + b - ab$. Therefore, the original certainty-factor of 0.4 combines with the increased belief of 0.6 resulting in a new certainty-factor of 0.76 ($= 0.4 + 0.6 - 0.4 \times 0.6$). Another way of thinking about this is that the new evidence (0.6) takes the original certainty-factor (0.4) 60% towards certainty (1.0).

Certainty-factors can also be negative, providing evidence that tends to decrease belief in a proposition. Negative certainty-factors combine using $a \oplus b \equiv a + b + ab$. Therefore a certainty-factor of -0.4 combines with a certainty-factor of -0.5 to give a certainty-factor of -0.7.

Thus far, the words belief and certainty-factor have been used interchangeably, but they are in fact distinct concepts. The *measure of belief* of a proposition is the (probabilistic) sum of all positive evidence. Similarly, the *measure of disbelief* of a proposition is the sum of all negative evidence. In order to compute a propositions certainty- factor, the formula

$$CF = \frac{(MB + MD)}{(1 - \min(MB, MD))}$$

is utilized. Using this formula, it can be seen that any proposition with equal measures of belief and disbelief will have a resulting certainty-factor of 0.0. This coincides with intuition. The careful reader may want to convince themselves that this equation is independent of the order in which beliefs and disbeliefs are accumulated.

Certainty-factor rules may also contain multiple antecedents, as in $A \wedge B \wedge C \xrightarrow{.7} D$. Conjunction is handled using the min function. The minimum certainty-factor from among A, B, and C combines with the 0.7 to determine D's measure of belief. Similarly, the max function is used with antecedent disjunction.

It should be noted that propositions with negative certainty-factors offer no evidence (positive or negative) for rules using this proposition as an antecedent. To see this, consider the rule $\textsf{Has-Phd}\xrightarrow{0.6}\textsf{Intelligent}$. This rule is meant to offer positive evidence for someone's intelligence if it is known that this person has obtained a Ph.D. degree. If nothing is known about the person's academic background, this rule will have no effect. If, however, it is known for certain that the person does not have a Ph.D. (certainty-factor -1.0), this should not be used as negative evidence that the person is intelligent. It is simply no evidence that they are intelligent by this rule. The distinction between the two rules:

$$\textsf{Has-Phd}\xrightarrow{0.6}\textsf{Intelligent} \qquad \textsf{NOT(Has-Phd)}\xrightarrow{-0.6}\textsf{Intelligent}$$

is significant, and needs to be carefully understood. The former makes an inference about someone who has a Ph.D., whereas the latter applies only to those that do not. Because of this distinction, the exact rule for calculating the measure of belief of a consequent proposition is max(0,CF of antecedent) multiplies by the certainty-factor of the rule. Negative certainty-factors do not pass their values forward.

In the original Mycin experiments, a 0.2 threshold was used to block certainty factors with low values. Any literal with certainty factor less than this threshold would not propagate any value forward. This was done with the notion that such low values should be essentially ignored, as they are close to 0. Rapture does not use this approach. One of Rapture's strengths is its ability to allow many seemingly small pieces of evidence to combine to produce a great deal of belief in a proposition. An arbitrary threshold prevents this from happening.

Finally, one of the most common misconceptions about certainty-factors is that they represent probabilities. They do not. The numbers themselves have only a subjective meaning, which may or may not be intuitive. Numbers closer to 1.0 represent strong and conclusive evidence, whereas numbers closer to 0.0 represent weaker evidence. A key distinction between certainty factors and true probabilities is the manner in which they interpret the negation of a proposition based on evidence. Consider again the rule $\textsf{Has-Phd}\xrightarrow{0.6}\textsf{Intelligent}$. As stated, this rule is designed to increase belief in one's intelligence level, given that they have obtained a Ph.D. Were this a probability, the obvious deduction would be that there is now a $0.4(= 1.0 - 0.6)$ probability that the person is not intelligent. In other words, any evidence for believing in a proposition automatically provides evidence for disbelieving it. Domain experts agree that this is not the intent of such rules

(Buchanan & Shortliffe, 1984). For certainty factors, evidence supporting a proposition has no direct bearing upon belief in its negation. Measures of belief and disbelief are calculated separately and combined to determine overall certainty factor as described above.

There have been successful attempts at providing probabilistic interpretations for certainty-factors, most notably by Heckerman (1986). Unfortunately, such interpretations generally require unreasonably restrictive assumptions upon the data, which will generally not be true for real-world domains. Primary in these assumptions is that all evidence supporting a conclusion must be conditionally independent. This is not true in most applications.

2.3 A Case For Certainty Factors

RAPTURE grew out of a desire to build a revision system that was capable of employing uncertain-reasoning, while being able to take full advantage of available domain knowledge. This was due to the existence of a number of domains where it was clear that some sort evidence summation would be necessary in order to successfully represent goal concepts. Examination of the DNA domains, for instance, has shown that it exhibits an M-OF-N property (von Heijne, 1987). This indicates that certain DNA strings can be identified whenever M-OF-N consecutive nucleotides match a given pattern. This is clearly evidenced in the original expert rules for determining splice-junction points (see Appendix B).

Also, a set of rules for diagnosing diseased soybeans (described fully in Chapter 4 (Michalski & Chilausky, 1980)) was created that made use of uncertain reasoning. The rule base contains two different types of rules. Rules labelled *significant* are suggested to carry 0.9 weight (out of a possible 1.0), while those labelled *confirmatory* carry the remaining 0.1.

Certainty factors were chosen as the formalism for RAPTURE for several reasons. First, it is perhaps the simplest method that retains the desired evidence-summing aspect of uncertain reasoning. As each rule fires, additional evidence is contributed towards belief in the rule's consequent. All evidence can then be combined giving an overall degree of confidence in the consequent. Since the value of a proposition's certainty factor is independent of the order in which beliefs and disbeliefs are collected, there is no concern over which evidence to consider first. This modularity of evidence, as well as the use of probabilistic sum enables many small pieces of evidence to add up to significant evidence. This is lacking in formalisms that use only `MIN` or `MAX` for combining evidence (Ling & Valtorta, 1991).

Second, probabilistic sum is a simple, differentiable, non-linear function. This is crucial for implementing gradient descent using backpropagation. Further, other formalisms for uncertain reasoning (e.g. Bayesian networks) have been shown to be NP-hard to evaluate in the general case (Cooper, 1990), and require the specification of exponentially many conditional probabilities in the fan-in of a node (Schwalb, 1993).

Even more significantly, however, is the widespread use of certainty factors. Despite recent criticism of certainty factors (Shafer & Pearl, 1990), there have been numerous knowledge-bases implemented using the certainty-factor model, which immediately gives our approach a large base of applicability.

One of the conclusions drawn in Buchanan and Shortliffe (1984) was that the performance of their certainty-factor system (MYCIN) was relatively insensitive to the precision of the weights on the rules. This has been one reason for much of the recent criticism of certainty factors. However, most of the experiments that were done with MYCIN rules only involved adjusting the precision of the weight of a rule, and not on the relative ranking of the rules. This is not the case with RAPTURE. By examining the revised sets of rules presented in Chapter 4 and Appendix B it is clear that RAPTURE is able to greatly change the values and relative strengths of certainty factors when deemed necessary.

Also of importance is the relative ease with which certainty-factor rule bases can be created out of Horn clauses. If domain knowledge is not initially represented in certainty-factor form, it is a simple matter to write an equivalent set of rules using certainty factors. One such approach is described in the next chapter. Despite their lack of true probabilistic semantics, certainty factors borrow a great deal of their meaning from probability theory. Because of this, the impact and effects of a rule's certainty-factor is quite intuitive.

Finally, and perhaps most importantly, are the empirical results. Presented fully in Chapter 4, results to date indicate that this approach leads to very accurate rule bases. In all of the domains tested, RAPTURE consistently performs as well as or better than other learning systems. This is a clear indication that this approach is one that is worth pursuing.

2.4 Backpropagation

BACKPROPAGATION, or simply BACKPROP, is a standard connectionist technique for training neural networks to correctly categorize sets of labelled examples (Rumelhart et al., 1986). This is a gradient-descent method that gradually shifts all of the weights in the network in a direction that will decrease network mean-squared error.

Each time a training example is processed by the network, any errors occurring at the output layer are noted. This error is then *propagated* back down the network towards the input layer. The weights on all links connecting into output nodes are modified slightly (as dictated by a learning-rate parameter) in the direction that will reduce the error at the output node. Estimates of remaining error are made for each node exactly one layer beneath the output layer. This modification of the weights on links beneath nodes producing error continues, until the input layer is reached. This terminates the processing of this example. All training examples are processed by the network, one at a time, repeatedly, until no further error reduction is possible. This marks the end of training. The mathematical formulae used in BACKPROP are presented in Section 3.6.1.

One of the properties of BACKPROP is that it is a hill-climbing technique. Because of this, it is possible that a local error-minimum will be reached during training, instead of a global minimum. This will result in inferior performance compared to an optimal weight setting.

2.5 Previous Rule-Base Refinement Systems

In the two sections that follow, other refinement systems are described. Results of these systems will be compared with those of RAPTURE in Chapter 4. One of these is a purely symbolic system, and another is predominantly connectionist.

2.5.1 EITHER

EITHER (Ourston & Mooney, 1994) is a rule-base revision system that uses propositional Horn-clause logic to represent its rule bases. It begins with background knowledge in the form of a Horn-clause rule base, along with a set of training examples.

EITHER revises rules whenever they incorrectly classify training examples, and terminates training only after all examples are processed correctly. EITHER contains mechanisms for specializing and generalizing the rule base. False-positive examples occur whenever the system places an example into a category to which it does not belong. This is an indication that the rules for this category need to be specialized. Similarly, false-negative examples occur whenever the system fails to place an example into a category to which it does belong. This is an indication that the rules for this category need to be generalized. The bias that EITHER works with is to make as few changes to the initial rules as possible, while classifying all training examples correctly. This is based on the assumption that the original rules contain much useful information, and are not terribly far from being correct.

EITHER is a modular system that contains independent modules for performing deduction, abduction, and induction. The analytical methods (deduction and abduction) are used to identify those portions of the rule base that are producing errors, and to carefully choose those training examples that are representative of this error. Induction is used to specify possible corrections to the rule base that will eliminate the errors caused by these examples.

EITHER supports two means for specializing its rule bases—removing rules, and adding conjuncts to existing rules. Both of these mechanisms strictly decrease the number of examples from the example-space that will be able to prove a given rule-consequent. This strictly specializes the rule-base. Specialization in EITHER is done in a manner guaranteeing that negative examples will not be provable in the given (incorrect) category, while simultaneously insuring that no positive examples become unprovable.

Generalizing the rule base proceeds similarly for false-negative examples. EITHER contains two mechanisms for achieving this. These are adding new rules to the rule base, and deleting conjuncts from existing rules. Each of these mechanisms facilitates rule firing, allowing more examples to be classified into a given category. These mechanisms are also designed to prevent false-positives from being produced. This strictly generalizes the rule base.

The bias of minimal modification to the original rule-base is designed to allow as much of the original background knowledge as possible to remain intact. Since changes to the rule base occur incrementally, there is no guarantee that the final result will be as close to the original as possible, though each step attempts to apply the smallest change that accomplishes its task. This ideally enables the rules to perform at a superior level of accuracy on unseen examples. Ourston illustrates the success of this heuristic with results from several domains (Ourston & Mooney, 1994).

2.5.2 NEITHER

NEITHER, or New-EITHER, is an updated version of EITHER developed by Baffes and Mooney (1993a). While still a purely symbolic-learning system, it allows M-OF-N style rules that allow consequents to be concluded whenever M of the N antecedents are true. By varying the value of M, rules with evidence-summing properties are created.

One of the most significant changes to the EITHER algorithm was a simplification of the search algorithm for potential changes to the rule base. Each time a modification to the existing rules is called for, EITHER entertains exponentially many possible modifications. NEITHER on the other hand, only considers one possible change to the rules for each misclassified example, determined heuristically to be the best change for that example.

NEITHER incorporates all of the generalization and specialization techniques of EITHER, plus adds its own through the modification of the M-OF-N rules. By *lowering* the value of M, rules become active more easily, since fewer antecedent conditions must be satisfied. This generalizes the rule base. Similarly, by *raising* the value of M, more antecedent conditions must be satisfied, making it more difficult for rules to become active. These properties gives NEITHER the ability to perform better in domains such as Promoter that benefit from partial matching (Baffes & Mooney, 1993b).

2.5.3 KBANN

As previously mentioned, KBANN (Towell & Shavlik, 1994) is a refinement system which translates a rule base into a neural network and then refines it using backpropagation. The translation into a neural network proceeds in a straightforward manner. First, a logical circuit is created using the AND-OR graph of the theory, and the weights of the units in the network are set to simulate AND and OR gates. In addition, all remaining features in the data are added to the input layer. The network is then fully connected by adding low-weighted links from every node in layer n to every node in layer $n + 1$.

Once built, the network is trained using backpropagation. To help minimize the size of the network, weight-decay (Hinton, 1986) is utilized. By adjusting each weight in the network slightly towards zero after each weight update, links that are not contributing to the network are eliminated.

After training, symbolic rules can be extracted from the network. By analyzing the weights of the incoming links, each unit is translated into a set of M-OF-N rules, that are satisfied if at least M of their N antecedents are true. The resulting rule base is generally much simpler than the revised network; however, there is no guarantee that the two representations are semantically equivalent.

KBANN has been tested in several domain areas, most of them concerning DNA sequencing, such as the promoter recognition and splice-junction determination domains discussed in Chapter 4. Results presented in Towell and Shavlik (1994) demonstrate the effectiveness of this approach, with generally superior performance to other learning systems.

2.6 Symbolic vs. Connectionist Debate

During the last several years, a rift has developed in the machine learning, as well as the general AI community. This rift has come about as a consequence of differing opinions regarding intelligence and learning. On the one side are the symbolic AI people, who tend to think of learning as being performed strictly by manipulating humanly-understandable symbols in intelligent ways. This enhances human comprehensibility, and allows one to evaluate, verify, and modify all manipulations.

On the other side are the connectionists, who believe that learning is best achieved by building networks of units that can be trained to learn perceived correlations between these units. Each unit can represent an observable feature from the domain, or a non-observable hidden feature. This is analogous to the way neurons in the brain work.

One of the results of the research presented in this dissertation is that RAPTURE bridges this gap quite effectively. From one perspective, RAPTURE's rule bases can be viewed in a purely symbolic manner. They are simply certainty-factor rules for solving the problem. Alternatively, these rule-bases can be viewed as connectionist networks. The certainty-factors are nothing more than weights on links that connect features from layer to layer, and their values are adjusted through the use of gradient descent. Rule (or architecture) modification takes place using a combination of symbolic and connectionist techniques.

Unlike (most) connectionist systems, all units in a certainty-factor network represent true features from the domain (which may be non-observable). Similarly, unlike (most) symbolic systems, there exists real-valued correlations between many of the features.

Furthermore, this system brings to light the similarity between the symbolic and connectionist approaches, as they can really be perceived as different ways of looking at the same problem. Much research is currently underway for developing hybrid systems which combine pieces of symbolic and connectionist learning techniques. This has led to the birth of several workshops, books, and a mailing list devoted to the development and discussion of hybrid approaches (Fu & Lacher, 1994; Kandel & Langholz, 1992). Many current results indicate that this combined approach holds much promise in many domains.

Chapter 3

The Refinement Algorithm

This chapter describes the algorithm used by RAPTURE for revising certainty-factor rule bases. The overall structure of this algorithm is illustrated in Figure 3.1. The key idea of this revision process is to use *training examples* to guide the revision. These are examples that have been carefully examined and labelled by a domain expert. Whenever the rule base classifies one of these examples differently than the expert, an error is deemed to have occurred. By carefully noting these mishandled examples, revisions can be made to appropriate areas of the rule base in an attempt to eliminate these errors. Once all of the training examples are handled correctly, it is hypothesized that the modified set of rules will be more successful at classifying novel examples than the original rule base.

3.1 Overview of the Algorithm

Given a rule base designed to partition examples into useful categories, along with a set of training examples, the task of RAPTURE is to revise these rules in order that they correctly categorize all of the training examples. It is this training phase of the algorithm that is the main focus of this chapter. After a brief overview of the algorithm, each learning component will be fully described in the following sections. There are also minor pre- and post-processing tasks that take place before and after training, and these too are described in turn.

Before RAPTURE can begin its training phase, the rule base needs to be converted into a connectionist network. This enables RAPTURE to borrow from existing connectionist techniques for its learning components. As the original rule base is expressed in a certainty-factor format, the network built from it uses certainty factors as weights on the links between nodes. Each node in the network represents a unique literal from the rule base, and antecedent nodes are connected to consequent nodes via a certainty-factor link. Throughout this document, this structure is referred to as a *certainty-factor network*.

Once built, this network is revised using a set of training examples to guide the revision. The revision process terminates when the network correctly classifies all of these examples. RAPTURE contains three training mechanisms to accomplish this task.

The first of these is Certainty-Factor backpropagation, or CFBP. This is the gradient descent algorithm developed for RAPTURE for adjusting the weights on the links between nodes. Gradient descent is a standard connectionist technique for training a (neural) network. As each

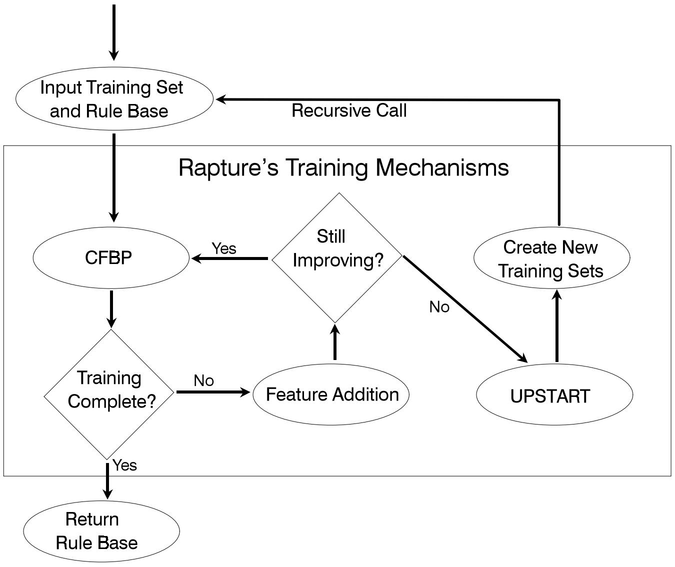

Figure 3.1: The Rapture Algorithm

training example is given to the network at the input layer, certainty-factor values are propagated forward towards the output layer. Once all values have reached the output layer, the network output can be compared with the "correct" output. A correct output is generally a 1.0 value at the output node corresponding to the correct category, and 0.0's at the remaining output nodes. This information is readily available since all of the training examples have been properly classified by a domain expert. Errors at the output level are then *backpropagated* back down towards the input layer. This has the effect of modifying some or all of the certainty factors associated with each rule. Backpropation (Rumelhart et al., 1986) is a hill-climbing technique that slightly adjusts the weight on each link in a direction that will most directly minimize the error caused by the particular training example. Examples are processed by the network one at a time in this manner until all examples have been processed. Processing all examples exactly once is termed an *epoch*, and any number of epochs may be necessary before the overall network error reaches a minimum. Ideally, CFBP will modify the network sufficiently so that all of the training examples will be correctly classified.

It is usually the case, however, that CFBP alone is not enough to completely train the network. This signals a need to modify the architecture of the network, which calls upon the second stage of Rapture's training algorithm. In this stage, new features (nodes and links) are added into the network, in an attempt to correct remaining classification errors. New links are connected to every output category that is incorrectly identifying training examples. These links connect appropriate nodes to the output categories in question. Appropriate nodes are those with the highest information-gain in discriminating the misclassified examples. These may be nodes already in the network, or new nodes built from features from the domain not currently in the network. Each new link is given a relatively low weight. Once all new links and nodes are in place, the network is sent back for another round of CFBP.

The process of CFBP followed by feature addition continues until either all of the training examples are successfully classified, or no further progress is being made. The latter case signals the final RAPTURE training mechanism, which is to use the UPSTART (Frean, 1990) algorithm. This is a connectionist technique for constructing hidden units. These hidden units act as intermediate terms in the symbolic rule base. For each output category with either false-negative or false-positive examples, two new hidden units are created. One of these is created to learn the false-negative examples, and is positively connected to the output category, while the other is designed to learn the false positives, and is negatively connected. Assuming that these hidden units are trained correctly, the network will now correctly classify all training examples. The training of the hidden units is done recursively, using RAPTURE. New links are created, CFBP is performed, and UPSTART is called as necessary.

Once the network is fully trained, the revised rule base can be read directly from the network. There is no translation necessary, as the rule base is exactly equivalent to the certainty-factor network. Each of the above steps is fully described below.

3.2 Pre-processing the Rule-Base

The first step in using RAPTURE is to obtain a rule base that is expressed in certainty-factor format. While there exist any number of rule bases that utilize some kind of uncertain reasoning, many of these are not directly expressed using certainty factors. In fact, of the four domains studied in this work, only one initial theory was originally described using certainty factors. For the others, modifications to the original rule base were required in order to convert them into certainty-factor form.

The basic idea in this rule-base conversion is to take a propositional Horn-clause rule such as $A \wedge B \wedge C \wedge D \rightarrow E$, and break it up so that each antecedent can independently contribute evidence for concluding the consequent. In this manner, observing one or more true antecedents will result in increased belief in the consequent proposition. This is unlike standard Horn-clause logic, where a false antecedent prevents a rule from concluding the consequent. Using the above rule as an example, four new rules will be created to replace the original—one for each antecedent. These are

$$A \stackrel{0.44}{\rightarrow} E \quad B \stackrel{0.44}{\rightarrow} E \quad C \stackrel{0.44}{\rightarrow} E \quad D \stackrel{0.44}{\rightarrow} E.$$

Each rule is given the same certainty factor value, calculated to produce a combined increase of belief in the consequent of 0.9. Thus, if all of the antecedents from the original rule are true, each rule will individually contribute the same amount of increased belief in the consequent, and these values will combine to give a total increased belief of 0.9. This value of 0.9 was suggested in Michalski and Chilausky (1980) in his study of diagnosing soybean diseases. In order to find the correct certainty-factor value for each rule, the formula $(1 - .1^{(1/n)})$ is used, where n is the number of antecedents in the original rule.

As an example, the original Promoter Recognition rule base (see next section) contains the following rule for identifying a Promoter:

$$\textsf{Contact} \wedge \textsf{Conformation} \rightarrow \textsf{Promoter}.$$

Translating this into certainty-factor form yields the two rules

$$\textsf{Contact} \stackrel{0.68}{\rightarrow} \textsf{Promoter} \qquad \textsf{Conformation} \stackrel{0.68}{\rightarrow} \textsf{Promoter}$$

Note that $0.68 \oplus 0.68 = 0.9$. This does not deny the possibility of a certainty-factor rule containing more than one antecedent. It may be the case that two or more factors must be present in order for there to be any evidence suggesting a conclusion. In fact, the Mycin rule base contains many rules in exactly this format. RAPTURE handles these using the standard certainty-factor combining rules as described earlier. RAPTURE only uses rules such as these when they are explicitly stated in the initial theory, and will never create them from a Horn-clause rule.

3.2.1 Special Rule Handling

During the process of converting non certainty-factor rule bases into certainty-factor networks, a few special cases arose. These included the ability to model weighted Horn-clause rules, the ability to handle negated literals, and M-OF-N rules.

Weighted Horn-Clause Rules

The original rule base for Soybean Disease Diagnosis was developed with the help of a soybean expert, and contained two types of rules that were given different levels of importance (Michalski & Chilausky, 1980). Though described in Horn-clause form, rules labelled as significant were suggested to carry approximately nine times as much weight as those labelled confirmatory. In other words, in order to be 100% certain of a diagnosis, 90% of the certainty would be derived from the significant rules, and the remaining 10% from the confirmatory. This clearly suggested the use of a certainty-factor of 0.9 for significant rules, and 0.1 for the confirmatory. These were the values that were used in translating these rules into certainty-factor format. Significant rules were handled exactly as the standard Horn-clause rules described above. Conjuncts are broken up into separate rules, and each is given the same certainty-factor value such that a 0.9 certainty factor will be produced whenever all of the antecedents are true. The only difference for translating the confirmatory rules was to give them a much smaller certainty factor. If all of the antecedents from a confirmatory rule are true, the resulting combined certainty factor will be only 0.1. This translation appears to model the original rules accurately.

Negated Literals

Another special type of rule that RAPTURE is able to handle is those containing negated literals. In the Splice-Junction Recognition rule base, there are two high-level rules that use negated literals that are essential for properly identifying Splice-Junction sites. In order to handle this, RAPTURE creates special `NOT` nodes which alter the certainty-factor values that propagate through them. These `NOT` nodes alter the certainty factor of a literal as described below.

A true literal, with certainty factor 1.0, will give its negation a certainty factor of 0.0. Remembering that negative certainty factors never propagate forward (see Chapter 2), this is the smallest value that can usefully be assigned. A literal with certainty factor 0.0 gives its negation a value of 1.0. This is in accordance with the methodology of *negation as failure* (Clark, 1978).

This was adopted for strictly intuitive reasons. If an expert is checking for a certain characteristic, she may have several tests (rules) that check for this. If each test fails, she is likely to be inclined to conclude that the particular characteristic is absent. This is despite lack of evidence for this conclusion, which is only obtained through tests that check for the *absence* of the characteristic. Any literal with a certainty-factor value between 0.0 and 1.0 will result in the negated literal receiving a certainty factor of (1.0 − CF). Any literal with a certainty-factor value of less than 0.0 results in the negated literal being true with certainty factor 1.0.

M-of-N Rules

In the rule base for identifying Splice-Junction sites in DNA strands, several M-OF-N rules are expressed. These are easily simulated with proper setting of certainty factors. Analogous to the translation for a standard Horn-clause, where all of the original antecedents must be true in order that the corresponding certainty-factor rules give a value of 0.9, an M-OF-N rule can be represented using slightly higher certainty factors. Assuming a 3-OF-5 rule, by giving each of the 5 certainty-factor rules the certainty factor corresponding to a Horn-clause with 3 antecedents, whenever any three of the five antecedents become true, a certainty factor of 0.9 will result. This also has the nice feature that if all five of the antecedents are true, an even greater certainty factor will result, corresponding with intuition.

3.2.2 Preparing to Construct the Network

Once a rule base has been obtained in certainty-factor format, RAPTURE can begin the revision process with the use of training examples. Although no formal study has been made to determine the precise relationship between an original propositional rule base, and its certainty-factor counterpart, it is evident through examination of the learning curves of Chapter 4 that both theories have very similar original predictive accuracies.

3.3 Converting the Rule Base into a Network

Before training begins, RAPTURE's first task is to convert the given certainty-factor rule base into a connectionist network. Unlike the translation of the previous section, this conversion is completely information preserving. The rule base and constructed network both process examples in exactly the same manner. The main idea in building this network is to create one node for each unique proposition in the rule base. All identical propositions map to the same node. Input features (those only appearing as rule-antecedents) become input nodes that are placed at the bottom (input layer) of the network. Output symbols (those only appearing as rule-consequents) become output nodes, and are placed at the top (output layer) of the network. These output nodes represent the various categories into which an example can be classified. Links are created to connect antecedent nodes with consequent nodes. One link is created for every rule in the rule-base, and the certainty factor of the rule becomes the weight of this link. As examples are presented to the network, values are propagated forward, and the measure of belief of each output node is computed. Each example is classified into the category with the highest output value.

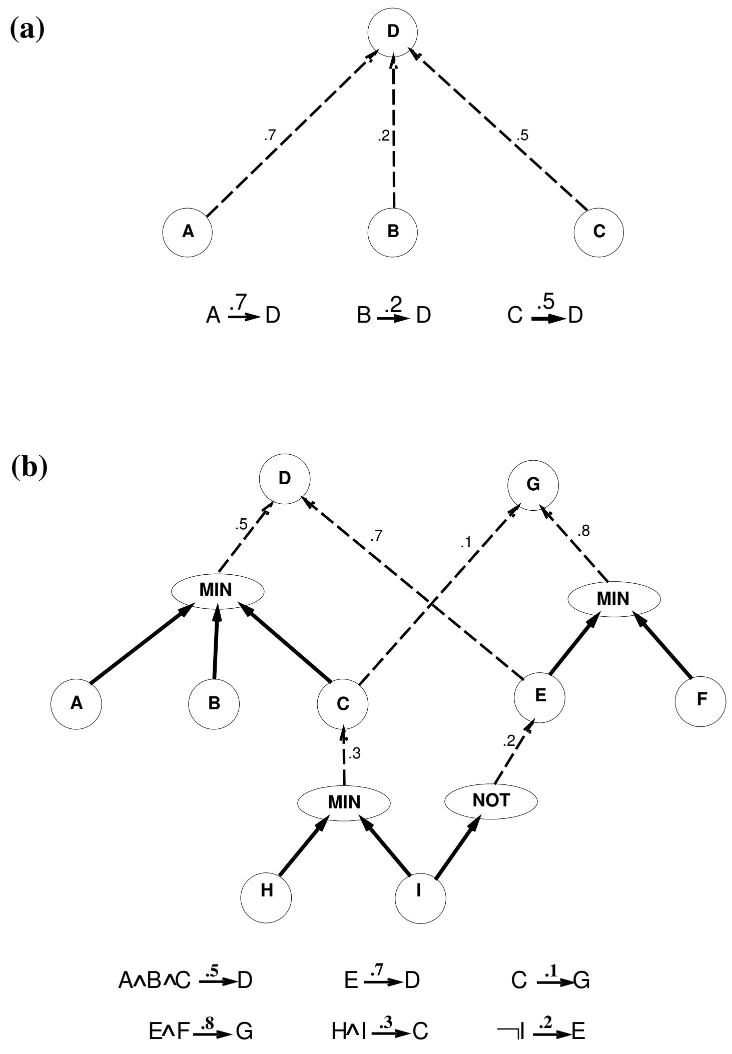

Figure 3.2: Converting Rules into Certainty-Factor Networks

3.3.1 Domains with a Default Category

In both of the DNA domains examined in this work, there exists a default *negative* category. Whenever the given rules fail to classify a particular example into any of the defined categories, it is labelled as a negative example. The rule base for Promoter Recognition only contains rules for one category, namely Promoter, and examples not satisfying the conditions for this category are labelled negative (or non-Promoters).

This is easily handled with the use of a threshold. The value of 0.9 is used in all experiments requiring a threshold. Any example receiving certainty factors less than 0.9 in all defined output categories is classified as a negative example. This threshold value remains constant throughout training, though may be altered before testing begins (see below).

Consider a simple example of three rules:

$$A \xrightarrow{.7} D \qquad B \xrightarrow{.2} D \qquad C \xrightarrow{.5} D$$

The network of Figure 3.2(a) is the certainty-factor network for this set of rules. These rules state that A, B, and C all contribute evidence towards D in varying degrees. Note the difference between these rules, and the single rule

$$A \wedge B \wedge C \overset{0.9}{\rightarrow} D$$

which states that all of A,B, and C must be true (to some degree) in order for there to be any evidence for D. As mentioned previously, RAPTURE handles these two cases differently—the former uses probabilistic sum to combine certainty-factor values, whereas the latter uses `MIN`.

Figure 3.2(b) illustrates the following more complete set of rules.

$$\begin{array}{ccc} A \wedge B \wedge C \overset{.5}{\rightarrow} D & E \overset{.7}{\rightarrow} D & C \overset{.1}{\rightarrow} G \\ E \wedge F \overset{.8}{\rightarrow} G & H \wedge I \overset{.3}{\rightarrow} C & \neg I \overset{.2}{\rightarrow} E \end{array}$$

As shown in the network, conjuncts must first pass through a `MIN` node before any activation reaches the consequent. This permits only the minimum value from among all inputs to propagate up the network. This is in agreement with the certainty-factor combining rules (Buchanan & Shortliffe, 1984). Note that each of the conjuncts is connected to the corresponding `MIN` mode with a solid line. This represents the fact that the link is non-adjustable, and simply passes its full activation value onto the `MIN` node. Similarly, solid lines are used to connect `NOT` nodes with their non-negated counterparts. The full activation is passed to the `NOT` node, which then negates this value as described earlier. The standard (certainty-factor) links are drawn as dotted lines indicating that their values *are* adjustable.

The above constructions illustrate the close correspondence between the certainty-factor rule-base, and the certainty-factor network. Each representation can be converted into the other, without loss or corruption of information. They are two equivalent representations of the same set of rules.

3.4 Pre-processing the Data

Once the network has been constructed, RAPTURE's task is to revise it in order that it can correctly classify all of the training examples. One of the biases of RAPTURE is that it continues training until 100% of the training examples are classified correctly. This is done with the hope of enabling the learned rule base to be more successful at classifying novel examples. Attempts are made to prevent overfitting (see Section 3.7.2), at least during node and link addition, though this is certainly one area for future work. The downside of this bias is the impossibility of successfully training on sets containing conflicting examples. These are examples that have identical feature vectors, yet have been given different classifications by the expert. In order to remedy this, all conflicting examples are removed from the training set before training begins. This is done by placing all examples with identical feature vectors and conflicting classifications into a conflict set. One example from this set is placed back into the training set, and the rest are discarded from training. The example that is selected is one that has been classified into the same category as the majority of the examples in the conflict set. In case of a tie, an example is picked at random from among tied categories.

3.5 Processing the Training Examples

Before any training can occur, examples must be input into the network. This is done by assigning each node in the input layer a certainty-factor value based on the example in question. Since each node in the input layer by definition represents one observable feature from the domain, these values can generally be taken directly from the example. RAPTURE is designed to handle several different types of observable features.

The standard feature is one with a finite number of discrete values. An input node representing the feature `(COLOR RED)` will receive a certainty factor of 1.0 if the incoming example has the value `RED` for feature `COLOR`. A certainty factor of 0.0 results if the value is any other color from the domain. If the example is missing the value for this feature, a certainty factor of $1/n$ is used, where n is the number of possible values for this feature. This has been shown to be an effective encoding for missing features in neural networks (Shavlik, Mooney, & Towell, 1991).

In many domains, there also occur features with continuous values, such as a patient's age. An input node corresponding to the feature `(AGE-OF-PATIENT < 50)` will receive a certainty factor dependent upon how closely the example satisfies the condition of the rule. This is evaluated using the formula $1/(1+e^{AGE-50})$. This has the nice property that the certainty factor will be higher for younger patients, yet still have a small positive value for patients slightly over 50. A patient exactly 50 years old will produce a certainty factor of 0.5, and this value will decrease rapidly towards 0.0 for older patients. This type of evaluation is typical of the method used to determine set membership in fuzzy set theory (Zadeh, 1965). All examples with no value for this feature are assigned the average value for the feature.

Finally, when creating RAPTURE–KBANN networks (discussed in Section 4.2), which are the certainty-factor counterpart of a standard KBANN network, and where every feature from the domain must be represented in the network, there arises the need to represent continuous features without a specific threshold. An example of this is a node representing the feature `TEMPERATURE`. A feature such as this signifies that only the magnitude of the value from the example is important, and the larger the value, the greater the resulting certainty factor. All training examples with values for this feature are scaled linearly between 0.0 and 1.0. The example having the smallest value will produce a certainty factor of 0.0 for the input node, while the example with the largest value will produce one of 1.0. All other values are scaled in between. If a test example occurs with a value outside this range, it will produce a certainty factor of 0.0 if below the training minimum, and a 1.0 if above the training maximum. An example with a missing value will receive the average temperature, and in the case of a missing value where no training examples have values, a certainty factor of 0.5 is produced. This is one distinction between RAPTURE and RAPTURE–KBANN as the former never makes use of such rules.

With the network and the training data in order, RAPTURE can begin the process of revising the rule base. RAPTURE contains three separate mechanisms for achieving this—certainty-factor backpropagation (CFBP), feature addition, and UPSTART hidden-node creation. These are fully described in the following sections.

3.6 Certainty-Factor Backpropagation

CFBP is the gradient-descent algorithm developed for RAPTURE. Gradient descent, or backpropagation (Rumelhart et al., 1986), is a standard connectionist algorithm for adjusting weights in a neural network. The basic idea is to systematically adjust all of the weights on the links in the direction that minimizes the mean-squared error of the network. Each time an example is processed by the network, a comparison is made between the actual output and the desired output. The desired output is generally a certainty factor of 1.0 at the correct output node, and 0.0 at all others. These differences are noted at each output node, and represent the error produced from the example. These errors are propagated back towards the input layer, and the certainty factor of each link is slightly adjusted in the direction that will decrease this error for this example. By cycling through all the examples one at a time, and comparing the network output with the desired output, the overall network error can be diminished. Each complete pass through the training examples, where each example is processed by the network exactly once, is termed an *epoch*. Epochs continue until the overall network error reaches a minimum. Specifically, RAPTURE continues this backpropagation until classification accuracy ceases to increase, and mean-squared network error improves by less than 0.001 after a complete epoch.

3.6.1 Standard Backpropagation

In order to train a *neural* network using backpropagation, the standard formula for adjusting the weight linking node i to node j (w_{ji}) after seeing pattern p is

$$\Delta_p w_{ji} = \eta \delta_{pj} o_{pi} \tag{3.1}$$

where η is the user-defined learning rate, o_{pi} is the output of unit i for input pattern p, and δ_{pj} is the output error of unit j for pattern p. The output of a node j is $f(net_{pj})$ where the net input $net_{pj} = \sum w_{ji} o_{pi}$ for all input connections i, and where f is a nondecreasing, differentiable function (generally a logistic function). The value of δ_{pj} is determined by the type of unit. If j is an output unit, then

$$\delta_{pj} = (t_{pj} - o_{pj}) f'_j(net_{pj}). \tag{3.2}$$

where t_{pj} is the correct output value for unit j. If j is not an output unit, then

$$\delta_{pj} = f'_j(net_{pj}) \sum_k \delta_{pk} w_{kj}. \tag{3.3}$$

3.6.2 CFBP

The above formulas, however, do not apply to certainty-factor networks since they model only the standard linear-summation of neural networks. In order to achieve gradient descent in a certainty-factor network, it is first necessary to derive the corresponding formulas for Certainty-Factor Backpropagation.

The main distinction between these two types of networks is the manner in which values are combined to produce total net input (net_{pj}). Certainty-factor networks use probabilistic sum to determine this. This probabilistic sum directly determines the certainty factor of the node since

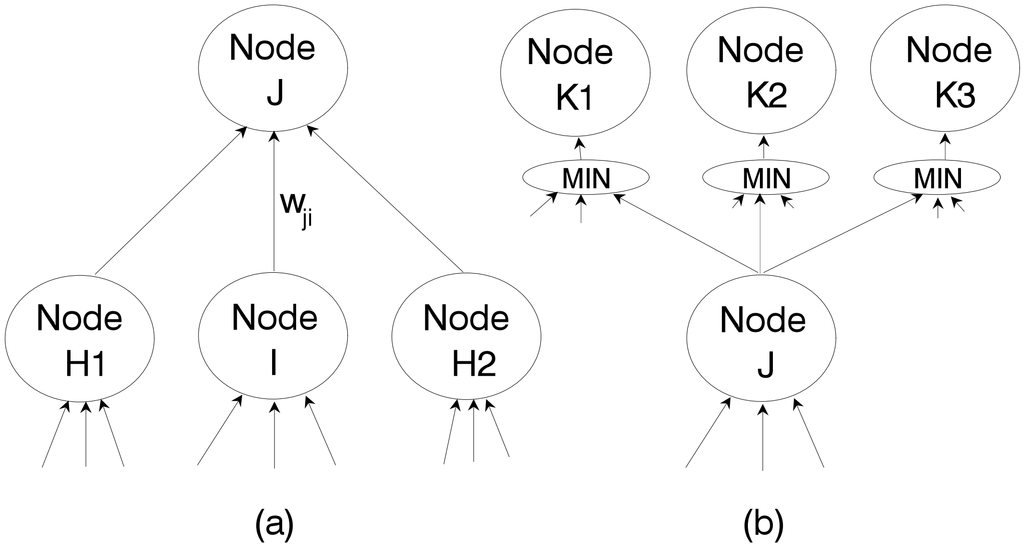

Figure 3.3: CFBP Diagrams

we have defined the output function to be identity— ($o_{pj} = net_{pj}$). In order to minimize mean-squared error in a certainty-factor network, CFBP is performed using the following equations. For a derivation of these equations, see Appendix A.

$$\Delta_p w_{ji} = \frac{\eta \delta_{pj} o_{pi} (1 - \sum_{+h \neq i} w_{jh} o_{ph})}{1 - \mathtt{MD_j}} \tag{3.4}$$

If u_j is an output unit

$$\delta_{pj} = (t_{pj} - o_{pj}) \tag{3.5}$$

If u_j is not an output unit

$$\delta_{pj} = \sum_{k_{min}} \frac{\delta_{pk} w_{kj} (1 - \sum_{+i \neq j} w_{ki} o_{pi})}{1 - \mathtt{MD_k}} \tag{3.6}$$

The essential variables are diagrammed in Figure 3.3. The first equation is for adjusting the weight (certainty factor) on the link connecting node i with node j (w_{ji}), after processing training example p. The w_{ji} notation represents the weight coming into node j from node i, indicating that node j is exactly one layer higher in the network (closer to the output layer) than i. The "Sigma with circle" notation is meant to represent probabilistic sum over its index value. The measures of belief and disbelief for node j are labelled as $\mathtt{MB_j}$ and $\mathtt{MD_j}$ respectively. In this equation, this probabilistic sum is indexed over $+h \neq i$. This is a probabilistic sum of all the positive links coming into node j, except for i itself. The h nodes are the other nodes also exactly one layer beneath, and connected to, node j. See Figure 3.3(a).

The second and third equations are used for determining the amount of output error that is caused by node j after processing training example p. In the third equation, the entire formula is summed over index k_{min}. This notation represents the fact that some k nodes receive no value from node i when processing example p. This is due to the fact that the output value from each node may have to pass through a MIN (or MAX) node before any value is passed onto node k. This makes it possible that j's value will not reach k, and any error caused by node k can not be attributable to node j. Thus the summation index k_{min} indicates that the summation is over only k nodes that received a value from node j. The probabilistic sum in this equation is similar to that of the previous equation. The index is over $+i \neq j$, which represents all positive links i coming into node k except for j itself. See Figure 3.3(b).

Using these equations, CFBP performs gradient descent on certainty-factor networks in a manner identical to standard backpropagation for a neural network. Examples are fed into the network one at a time through the input nodes. Weight adjustment occurs with each example as network error *propagates* down towards input lines. Epochs are run until network error reaches a minimum.

Assuming the target output value is 1.0 for the correct category and 0.0 for all other categories, it is virtually impossible to achieve an overall mean-squared error of zero. When combining evidence using probabilistic sum, an output of 1.0 is only possible with the existence of a rule that concludes the consequent with certainty (i.e., 1.0). The existence of such a rule still requires that all antecedents have certainty factors of 1.0. Such cases are not common in probabilistic classification problems. Because of this, RAPTURE deems a classification correct when the output value for the correct category is greater than that of any other category. Since this is considered a correct diagnosis, no error propagation takes place ($\delta_{pj} = 0$), even though it is unlikely that the mean-squared error for this example is 0.0. This is similar to the method used in Yu and Simmons (1990) and many others.

The remaining issue is when to terminate CFBP. The obvious stopping criteria is after all examples are classified correctly. This simple criteria is not used for a couple of reasons. First, since backpropagation is a hill-climbing algorithm, it is very possible that 100% training accuracy will never be reached. The network error may reach a local minimum, where further backpropagation will not help. Second, classifying all of the examples correctly does not mean that further weight adjustment is of no use. Correct classification does not imply 0.0 mean-squared error, as previously explained, and it is possible that further gradient descent would minimize error to an even greater extent. A more useful criteria for terminating weight adjustment is when overall mean-squared error is minimized. RAPTURE checks the total mean-squared error after every 10 epochs, and halts CFBP whenever training accuracy is observed to decrease, or mean-squared error decreases by less than 0.001, with no improvement in training accuracy.

If at this point, all of the training examples are being correctly classified, the network is considered trained, and RAPTURE terminates, returning the trained network. If not, a call to the RAPTURE's second training mechanism is made.

3.7 Feature Addition

This module is called whenever CFBP fails to train the network to 100% accuracy with the given examples. Since this is commonly a sign that the architecture of the network is insufficient for the current classification task, this module attempts to modify it in a way that will further improve training accuracy. Specifically, carefully selected nodes and links are added into the network in an attempt to bring training accuracy closer to 100%. The essential decision to be made is the

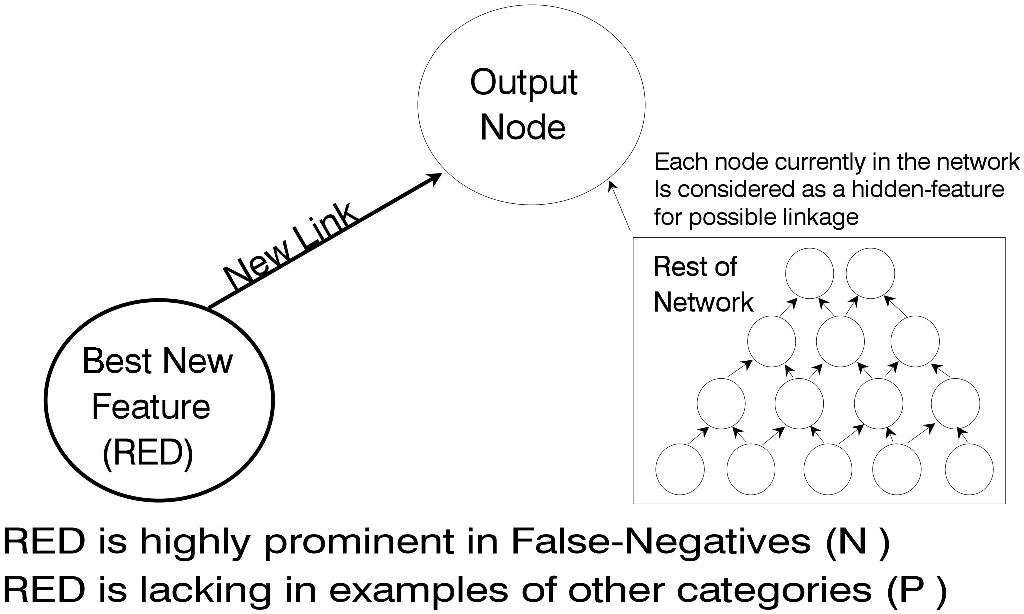

Figure 3.4: Adding New Features

selection of the appropriate feature from the domain to be added into the network, and this may come from an existing node in the network, or an observable feature from the domain not currently in use.

In order that the new feature has as great an impact as possible on the successful classification of training examples, it is directly connected to appropriate output nodes in the network. It is connected with a small positive link weight to one output node, and with a small negative link weight to others. The selection of the most appropriate feature to add uses the information-gain metric of ID3 (Quinlan, 1986). The following subsection describes this process in detail.

3.7.1 Choosing the Best Feature to Add

Assuming that the network has not reached 100% training accuracy, by definition there remain training examples that are being classified into incorrect categories. Each such example is both a false-negative example, and a false-positive example. It is a false-negative example for the correct category, since it failed to be classified into this category. It is also a false-positive example, however, for the category into which it was classified. In fact, it is a false-positive example for every category whose output node obtained a higher certainty-factor value than did that of the correct category.

Formally, let us define C as the set of all possible categories into which an example may be classified, and C_i represent the ith category. We define N_i to be the set of false-negative examples for category C_i. These are those examples whose target (correct) category is C_i, yet the network has classified as $C_{j \neq i}$. In fact, the network may have given a higher certainty factor to several C_j for any given example in N_i. We can define M_i as the set of all mistaken categories $C_{j \neq i}$, such that for some example in N_i, C_j has a higher certainty factor than C_i. Finally, define P_i as the set of all examples whose target category is among the categories in M_i, namely the true positives for all of these categories, regardless of how the network handles them.

As shown in Figure 3.4, for each category C_i, we now have two disjoint sets of examples. The first of these, N_i, is the set of false-negative examples for this category. The second is P_i, which is the set of true-positives for all categories that are being confused with C_i.

Since the network is classifying the false-negative examples into these incorrect categories, a new feature is needed that the network can use to better discriminate between these groups of examples. Quinlan's ID3 metric (Quinlan, 1986) has been adopted by RAPTURE as the solution to this problem.

ID3 is designed to build decision trees that classify examples into pre-defined categories. Given a set of examples described using feature vectors, ID3 selects the feature that best partitions these examples into subsets that correlate with an expert's classification. This is done using an information-gain metric. Each node in the tree is the feature with the highest information-gain among examples at this node, and each branch beneath this node represents a particular value.

This same information gain can be used to help RAPTURE add beneficial features. RAPTURE has two sets of examples to work with (N_i and P_i), and needs to find the feature-value pair that is highly prominent in the former, yet lacking in the latter. Once this feature has been determined, it can be used as positive evidence for category C_i and negative for all categories in M_i.

By labelling each example in N_i as *negative*, and those from P_i as *positive*, RAPTURE can then use information gain to determine which feature maximally distinguishes these sets of examples. Specifically, this metric chooses the feature that minimizes

$$\frac{[(P_{yes} + N_{yes}) \times INFO(yes) + (P_{no} + N_{no}) \times INFO(no)]}{N}$$

This function calculates information gain by returning a number between 0.0 and 1.0, where 0.0 represents maximal gain, and 1.0 represents no information gain. The variable P_{yes} refers to the number of positive examples with value *yes* for the feature in question. N_{yes} is the number of negative such examples. $INFO()$ is a function of the value (yes or no) defined as $INFO(\texttt{value}) =$

- 0 `if` $P_{value} = 0$ `or` $N_{value} = 0$
- $-ulog_2u - vlog_2v$ `otherwise,`
  `where` $u = P_{value}/(P_{value} + N_{value})$, `and` $v = N_{value}/(P_{value} + N_{value})$.

The major task of this module is to check every possible feature from the domain, as well as the network. ID3's metric calculates the information gain for each node currently in place in the network. This excludes nodes that are already directly connected to the output node in question. This is then compared with the information gain of every feature from the domain not currently in the network. RAPTURE does this by considering every feature-value pair from the domain as a binary feature. Instead of having a feature `COLOR` with values `RED, BLUE`, and `GREEN`, we have 3 features `COLOR=RED, COLOR=BLUE, and COLOR=GREEN`, with values `NO` or `YES`. For features with continuous values, a set of potential threshold values is created. This set is created by listing all of the values for this feature from the training examples. This list is sorted, and midpoints are created between each successive numeric value. Each of these midpoint values is then used, and tested as a potential threshold for this feature. This has been shown to be a successful technique for creating values to test against (Quinlan, 1993).

All intermediate (non-observable) features appearing in the network are also tested for information gain. Every example when processed by the network will give some certainty-factor value to every node in the network. If a particular example gives a 0.4 certainty factor to the the node in question, this represents 0.4 of an example containing this hidden feature, and 0.6 of an example that does not contain this feature. Since the information-gain formula requires counts of the number of examples with and without this feature, these values are totalled over all of the training examples, and given to the information-gain equation.

Any information-gain ties are broken at random. Once a feature has been selected, it is added as new positive evidence for C_i. For newly inserted features, it may be the case that the selected feature is *negative* evidence for C_i, meaning that it is highly prominent in P_i while lacking in N_i. To correct this, the value of the feature is simply inverted (e.g., from `COLOR=RED` to `COLOR≠RED`). The alternative method of creating a `NOT` node, rather than inverting the value has the undesirable side effect of increasing the size of the network. For intermediate features currently in the network, where inversion is not possible, only positive evidence is considered. Placing a small positive weight (0.1) on the rule produces a rule of the form $\texttt{COLOR=RED} \xrightarrow{0.1} C_i$, which serves as new positive evidence for C_i. Similar rules (using the same feature pair) are built for each of the categories in M_i with negative weights (-0.1). This gives new negative evidence for all of these categories. The examples in N_i will now produce higher certainty-factor values at the output node for C_i, and smaller values at the output nodes for M_i.

This feature addition is performed for each category $C_i \in C$. Note that for some categories, the false-negative list will be empty. This happens whenever every example of this category is classified correctly. This produces no new nodes in the network. False positives are not dealt with explicitly, since these will turn up as false negatives for some other category. Once there are no further false-negative examples, the network is fully trained.

Note that is may be possible that none of the output nodes have any false-negative examples, yet network accuracy is less than 100%. This can occur with the existence of a default category, described in Section 3.3.1. This category is never represented in the network with an output node, but rather a threshold value is used to determine when an example falls into this default category. If no output node has a certainty factor greater than this threshold, the example is categorized into this category (usually the *negative* category).

When this occurs, a new feature is linked into the network exactly as described above. False negatives for the default category, along with the true positives of the mistaken categories are collected, and the highest information-gain feature-value is selected. The only difference now is that it is impossible to connect this node as positive evidence for the default category, since it has no output node. But, as before, negative link connections can still be made to other categories, and this produces the desired effect.

With these new nodes in place, RAPTURE returns to CFBP in order to train the weights on the new links. Ideally, this will allow more training examples to be successfully classified. This entire process (CFBP followed by adding new nodes and links) repeats until either all training examples are correctly classified, or training accuracy begins to decline.

Once all of the examples are classified correctly, the network is considered trained. Before returning the network, all links with certainty factors within 0.0001 of 0.0 are deleted, simplifying the complexity. If network accuracy begins to decline before 100% accuracy occurs, the final RAPTURE training module is called. Links near zero are similarly deleted before the next module is called.

3.7.2 Avoiding Overfitting During Feature-Addition

Early tests with Rapture showed that by continuing to add features as described in the previous section until 100% training accuracy was obtained, generally poor generalizations resulted. For larger numbers of examples in some domains (e.g. soybean), 100% accuracy is never achieved. After adding a certain number of links into the network, mean-squared error often got considerably worse. Continuing to add links beyond this point, despite often ending with 100% training accuracy, poor test performances resulted. If, however, training terminated as soon as mean-squared error began increasing, even with less than 100% training accuracy, better test performance resulted. This is a clear sign that overfitting is occurring.

In order to prevent this, a call to the Upstart algorithm is made. Rapture terminates the feature-addition portion of the algorithm as soon as network accuracy ceases to improve, and mean-squared error begins to increase. This is noted individually for each output category, as well as for the entire category. Whenever the addition of a new feature causes no improvement in overall network accuracy, no improvement in the accuracy of the particular category being worked on, and no improvement in network mean-squared error, the particular category is marked as ready for Upstart, and no further feature addition is performed for this category. It is occasionally the case that the addition of a particular feature may actually create more errors in the particular category, yet still improve overall accuracy by helping out other categories. This does not cause a call to Upstart, and features continue to be added for this category. Once every category is either completely trained, or ready for Upstart, the feature addition module terminates.

3.8 The UPSTART Algorithm

The final module used by Rapture is the Upstart algorithm (Frean, 1990). This is a connectionist technique for creating hidden nodes in a neural network. The basic idea is to create new hidden units directly beneath any output nodes that are producing classification errors. These new nodes are designed to learn the incorrectly classified examples, and are linked directly to the problematic output nodes. In essence, these nodes learn the exceptional cases that the current network is unable to classify.

Specifically, the algorithm looks at every output node, and notes which examples it is misclassifying. Since there are two types of errors that can be made (i.e., false-negatives and false-positives), two new hidden units are created. One of these is designed to learn the false-negatives for this category, while the other is designed to learn the false-positives. Clearly, if a particular category does not have both of these types of errors, then the corresponding hidden unit is not necessary. Once trained, the node representing the false-negative examples is linked positively to the output unit, whereas the one representing the false-positives is linked negatively. Once these new nodes are trained correctly, and linked into the network, no false-negative or false-positive examples will remain for this output category. Once this is done for each category, training is complete, as now the network categorizes each training example correctly.

The problem is now reduced to one of properly training the new hidden units. Fortunately, a very elegant solution is available—namely to train the new units using the Rapture algorithm recursively. This means applying all of the above modules with a new problem. CFBP followed by feature addition again repeats until termination is signalled. If all examples are now correct, the

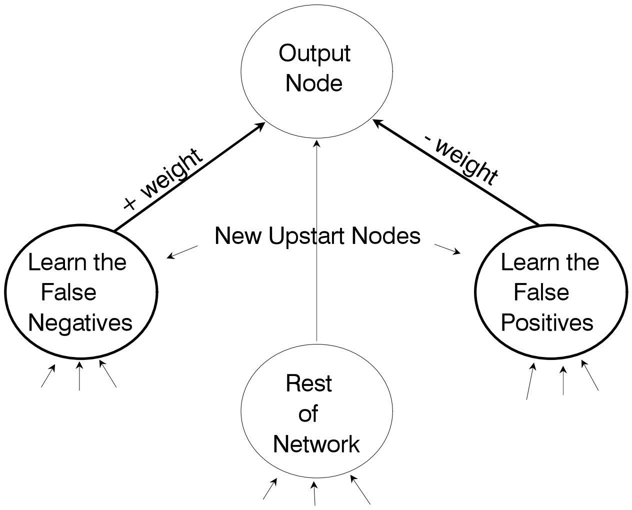

Figure 3.5: Adding Hidden Units – Upstart

algorithm terminates, otherwise UPSTART is called again a new set of misclassified examples.

In practice, it is rare to see more than one level of UPSTART being called, though it does occur when there are hundreds of training examples. Once all hidden units are fully trained, they are linked into the original network with just enough weight to force all examples to be classified correctly. This weight is determined by performing a binary search over possible weights, until a certainty factor within 0.01 of the lowest possible value is obtained. This lowest possible value is the minimal value that can be assigned to the new link that enables all of the misclassified examples in question to be handled correctly. Any value greater than this minimum value will correct these examples. Using too high a value, however, results in requiring remaining UPSTART link-weights to have an even higher value in order to correct misclassified examples. For this reason, it is desirable to choose a link weight not far from the minimum value.

Finally, as an implementation note, generally only one new UPSTART unit is created below each output unit with misclassified examples. This is because I chose to create all of the false-negative UPSTART nodes first before creating any false-positive UPSTART nodes. Since as soon there are no false-negative examples, all examples are correct, and false-positive UPSTART nodes are only needed in domains with a default category. This was decided because for each category, there were generally many fewer false-negative examples than false positives, and this simplified the work of UPSTART.

3.9 Post-Processing

One of the elegant features of this algorithm is the minimal need for post-processing. One the network is trained, the new rule base can be read directly off of it. This is the reverse process of building the network, and can be done directly with no loss or corruption of information. An overview of the entire algorithm is depicted in Figure 3.6.

- Loop until training accuracy ceases to improve.
  1. Perform CFBP on the network. Use given training examples, and as many epochs as necessary until mean-squared error ceases to improve.
  2. If not 100% training accuracy, use ID3 information gain to add new features (or link existing ones). Create one new link for each output unit misclassifying positive examples.
- Remove all near zero-weighted links
- For All Output Units Producing Errors
  1. Build one unit to learn the false-negative examples.
  2. Build one unit to learn the false-positive examples.
  3. Train each of these units using RAPTURE
  4. Link new units to the Output Unit with appropriate weight.

Figure 3.6: Overview of the RAPTURE Algorithm

Chapter 4

Experimental Results in Real-World Domains

In order to test the performance of RAPTURE, rule bases from various domains were sought. A successful domain requires training examples of all target concepts, along with a set of rules for classifying the examples. An ideal rule base for RAPTURE is one that is originally expressed using certainty-factors, though to date, only one such rule base has been obtained. The other rule bases examined in this chapter are originally expressed as sets of Horn-clauses, and manually converted into certainty-factor format for use by RAPTURE. This chapter presents results from four unique domains, one of which contains two sets of examples.

Two of these rule bases come from problems of genetics that attempt to identify certain regions of DNA sequences. The first of these is identifying promoter sequences, which are portions of DNA that precede gene sequences. The second attempts to identify exon-intron and intron-exon borders in DNA sequences. These borders distinguish the parts of a DNA sequence that code genetic information, from those that do not.

The third rule base that RAPTURE has been tested on is a version of the original MYCIN rule base that came from the Stanford Medical Library. This rule base was ideal for RAPTURE, as it was designed using certainty-factors in its rules, and was created to provide consultative advice on diagnosis and therapy for infectious diseases. The fourth rule base is one that classifies diseased soybeans into one of fourteen common diseases. This rule base comes from Michalski and Chilausky (1980). These domains are discussed in detail in the sections that follow.

4.1 The Train and Test Methodology

In the following sections, numerous graphs are presented. These are *learning curves* that show the performance of various learning systems in the given domains. All results were run on Sun Sparc 5's. The horizontal axes represent the number of training examples that were given to each system for learning, and the vertical axes plot either test accuracy on novel examples, the time it took to train the system, or the complexity of the learned rule base (measured as the number of rules or the number of links in the network). Except where noted, all of the systems are given the same sets of examples for training, and are tested on the same sets of novel examples. During each trial, beginning with no training, the number of training examples increases by adding new novel

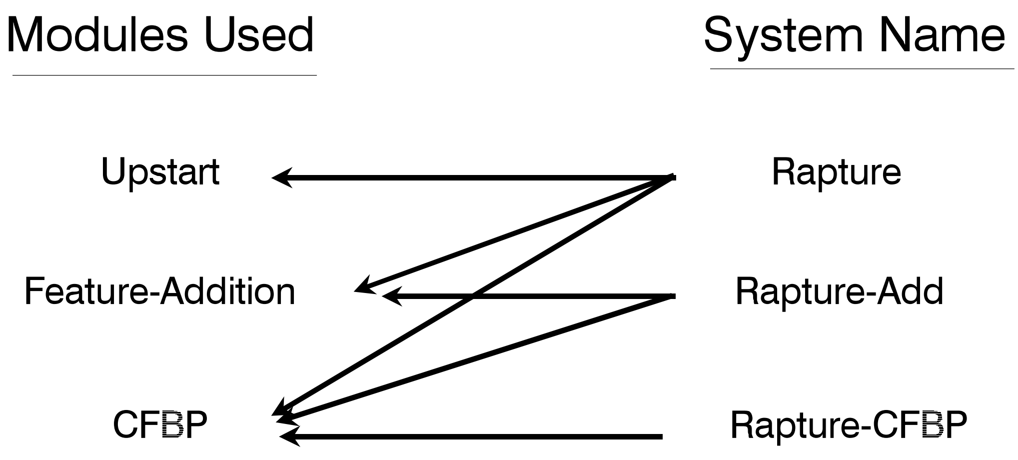

Figure 4.1: Modules Used in Rapture Ablations

examples to the current training-set. The test set of examples remains static throughout a given trial, and is created from examples not used for training. All of the graphs represent values averaged over at least 20 independent trials. Statistical significance is measured by running two-tailed paired t-tests on the data.

In the learning-curve plots that follow, I have made attempts to be as consistent as possible in labelling each of the systems. For example, RAPTURE is consistently plotted with a slightly thicker line than the others, and uses black dots to mark individual points. KBANN is consistently plotted as a thinner solid line, using an asterisk for individual points. Systems that do not begin with any domain knowledge use non-filled-in marks for their data points. This makes it much easier to quickly pick out these systems.

4.2 Versions of Rapture used in Testing

In order to fully test RAPTURE, observe the contribution of each of the component algorithms, and make comparisons with other learning algorithms, several ablated versions of RAPTURE were created, and tested on the above domains. Results of these are compared with results from standard learning systems.

The first two of the ablations are simply RAPTURE with one or more training modules disabled. The simplest of these is RAPTURE–CFBP, which uses only the first algorithm of RAPTURE-—the weight modification routine. The initial theory is converted into a certainty-factor network, and CFBP is performed until training accuracy is maximized. Similar to this is RAPTURE–ADD, which is RAPTURE without the UPSTART algorithm. CFBP is performed on the initial theory, and new links are added repeatedly until no further improvement is observed. These main RAPTURE ablations are diagrammed in Figure 4.1

In order to test the significance of the initial theory, RAPTURE–NULL was also created. This is the complete RAPTURE algorithm beginning with no background knowledge. The initial network simply consists of a set of output nodes with no input connections. New nodes are created and linked through feature-addition, CFBP is performed, and the UPSTART algorithm is called to create hidden units. RAPTURE–NULL terminates when training accuracy reaches 100%.

The variant RAPTURE–KBANN, is an attempt to replicate the KBANN algorithm (Towell & Shavlik, 1994) using RAPTURE's certainty-factor combining functions. Beginning with an initial theory represented as a certainty-factor network, all features from the domain not currently represented in the network are added to the input layer. The network is then fully connected by linking all nodes in one layer, to all nodes in the next higher layer. These new additional links are given a minimal link-weight. CFBP and weight decay (Hinton, 1986) are performed over this network until training accuracy ceases to improve.

Finally, CF-NET is CFBP performed over an initially random-weighted two-layer certainty-factor network. This system is created out of an attempt to discover any potential differences between standard BACKPROPAGATION and CFBP. CF-NET uses an architecture that includes an input layer, an output layer, and one hidden layer. The input layer consists of one input unit for each possible value for each feature of the domain. The output layer has one output unit for each output category, and the hidden layer contains the number of units equal to *(number-input-nodes + number-output-nodes) / 10.* This is the architecture that was used for standard backpropagation in Shavlik et al. (1991). CFBP is applied to this network until training accuracy is maximized.

4.3 Experimental Hypotheses

RAPTURE was designed to be able to work well in a framework involving uncertainty, and be able to take advantage of available background knowledge. Because of this, we believe that RAPTURE should be able to outperform systems that are unable to make use of background knowledge, or have difficulty expressing uncertain target-concepts. Since the given background knowledge should ideally be not too far from a *correct* set of rules, RAPTURE should be able to train much faster, both in terms of number of training examples required to reach a certain performance level, as well as CPU time necessary to train over a given number of examples.

Because of this, it is hypothesized that RAPTURE will be able to produce more accurate sets of rules than C4.5, which is unable to take advantage of background knowledge, and can not easily represent uncertain concepts. Similarly, RAPTURE should be able to outperform BACKPROP, which despite being able to represent uncertain target-concepts, allows no use of background knowledge.

The system EITHER, on the other hand, is designed to make use of background knowledge, but with its use of Horn-clauses to represent rules, it is unable to accurately represent uncertain concepts. This should lead to performance that is inferior to that of RAPTURE.

Systems that are able to make use of background knowledge, and represent uncertain target-concepts, such as KBANN and NEITHER, should be competitive with RAPTURE. Since KBANN requires a fully-connected network containing all available input features, it should take more time to train, and may produce larger rule bases. Since NEITHER, on the other hand, can only represent uncertainty through the use of M-OF-N rules, which are less general than certainty factors, it may produce slightly less accurate sets of rules.

The separate modules of RAPTURE were designed to improve the accuracy of the initial rule bases as much as possible with respect to training examples. Because of this, it is hypothesized that the performance of ablated versions of RAPTURE will suffer. Further, increasing the number of inactive modules should result in increasingly poor performance. Hence RAPTURE should outperform RAPTURE–ADD, which should outperform RAPTURE–CFBP. The system RAPTURE–NULL will demonstrate the utility of the background knowledge, as it begins with none. The performance of this system should suffer because of this. Also, since RAPTURE–KBANN begins with a much larger initial set of rules, RAPTURE should be able to train faster, and produce less complex rule bases, which should give it better generalization performance.

Due to their very similar techniques, BACKPROP and CF-NET are hypothesized to produce nearly identical results. The only difference between these two systems are their combining functions, and corresponding backpropagation formulas. Since those of CF-NET are more complex, and of a higher order, it will almost certainly take more time to train these networks, and they may converge at a slower rate.

4.4 DNA Primer

Since two of the four domains presented here deal with analyzing DNA sequences, a brief introduction to this domain is in order. More detailed information can be found in (Frank-Kamenetskii, 1993; von Heijne, 1987; Rosenfield, Ziff, & van Loon, 1983; Portugal & Cohen, 1977; Kornberg, 1974).

DNA, short for Deoxyribonucleic Acid, is the molecule that contains the genetic information in all living creatures. It is made up of two polynucleotide strands that form a double helix. Each of these strands consist of a long chain of nucleotides, and each nucleotide can be one of four types (known as bases). These bases are adenine, guanine, cytosine, and thymine, abbreviated as A,G,C, and T. Genetic information is encoded via the linear order in which bases are arranged in a given strand of DNA. It is estimated that a typical strand of human DNA contains more than 3,000,000,000 nucleotides.

Genes, the smallest physical units of heredity, are actually sequences of DNA that code for some functional product—commonly protein. Each linear sequence of three bases defines a codon, and these specify a particular amino acid in the protein. Since there are only 20 amino acids found in proteins, the $64 = 4^3$ possible base triplets are easily sufficient. There is much ongoing research into decoding the information contained in strands of DNA, and more is being learned continuously. The New York Times (Wade, 1995) reports on rapid progress currently being made, and quotes two scientific reports that estimate that as much as 99% of human-DNA genetic-information should be fully decoded by the year 2002.

One current unknown in the decoding process is the determination of the starting points of gene sequences. Knowing exactly which nucleotides encode any given gene greatly simplifies the decoding process. This problem is made all the more difficult by the fact that the majority of nucleotides in a DNA strand don't code for anything, and are simply referred to as "junk" DNA. It is estimated that as much as 90% of animal-cell DNA is junk.

Fortunately, there are special sets of base-sequences that indicate where genes begin and end. Sequences that precede a gene are known as *promoters*, and those that terminate them are referred to as *terminating codons*. Identifying the base sequences that code for promoters is the subject of one of the test domains of this work.

Research has shown that the complete set of codons for any one gene are not contiguously placed in the DNA strand. Codons and junk DNA are interspersed throughout all gene sequences, further complicating the decoding process. Contiguous areas of codons are known as Exons, and the interspersed junk areas are known as Introns. Identifying the borders between Exons and Introns is the subject of another one of the test domains presented in this work.

4.5 DNA Promoter Experiments

As discussed in the previous section, Promoters are short DNA base-sequences that precede the beginning of genes. The ability to identify these regions greatly simplifies the task of genetic decoding. While much is known about the structure of Promoter sequences, there is no known procedure for identifying them solely by examining base sequences. Physical tests do exist, however, that can be performed on strands of DNA to identify these regions. This is due to the fact that a promoter is the location where protein RNA polymerase binds to the DNA structure. While further explanation can be found in Frank-Kamenetskii (1993), suffice it to say that such a test will identify Promoters with very high accuracy. This allows one to examine the base sequences that precede genes, in order to search for regularities.

Through analysis of the biological literature of O'Neill and Chiafari (1989), Noordewier, Towell, and Shavlik (1991) developed a set of rules for recognizing Promoters given a base sequence.

The certainty-factor rule base for identifying Promoters is illustrated in Figure 4.2. All original (non-translated) rule bases can be found in Appendix B. Each line represents one rule– consisting of the consequent, the certainty factor, and the antecedents. There are thus two rules that conclude `promoter`. One of them uses `contact` as evidence, and the other uses `conformation`. All of the rules for this domain have antecedents of length one.

Though these rules contain a great deal of accurate information, they are unfortunately overly-specific. In fact, these rules never classify an example as a Promoter (i.e., no examples produce a promoter certainty-factor of 0.9 or greater). Attempts to revise this rule base are discussed in the following sections.

4.5.1 The 106-example Promoter Data Set

The 106 example Promoter data set is available from the Irvine machine-learning data repository (Merz, Murphy, & Aha, 1996), and because of its wide availability has been tested on a number of learning systems. This data set consists of 53 examples of Promoters (i.e. positive example), and 53 examples of non Promoters. There are no conflicting examples, and there are no examples with missing features.

Each example is represented as a sequence of 57 contiguous nucleotides taken from a strand of DNA, with each nucleotide being one of the four base values—A, C, G, or T. The 57 nucleotides in the examples represent a window carefully positioned around a possible Promoter site, so that nucleotides 51 through 57 actually represent the first nucleotides of the gene that follows the possible Promoter (in positive examples). The first 50 features of the examples are labelled P-50 through P-1, representing DNA positions -50 through -1. These are the positions which contain the potential Promoter. The final 7 positions are labelled P+1 through P+7, representing positions +1 through +7, which is the beginning of a potential gene. Note that there is no position P0. The classification

```
promoter <-0.68-- (contact)
promoter <-0.68-- (conformation)

contact  <-0.68-- (minus_35)
contact  <-0.68-- (minus_10)

minus_35 <-0.28-- (p-37 c)
minus_35 <-0.28-- (p-36 t)
minus_35 <-0.28-- (p-35 t)
minus_35 <-0.28-- (p-34 g)
minus_35 <-0.28-- (p-33 a)
minus_35 <-0.28-- (p-32 c)
minus_35 <-0.28-- (p-31 a)

minus_10 <-0.25-- (p-14 t)
minus_10 <-0.25-- (p-13 t)
minus_10 <-0.25-- (p-13 a)
minus_10 <-0.25-- (p-12 t)
minus_10 <-0.25-- (p-12 a)
minus_10 <-0.25-- (p-11 t)
minus_10 <-0.25-- (p-11 a)
minus_10 <-0.25-- (p-10 a)
minus_10 <-0.25-- (p-9 t)
minus_10 <-0.25-- (p-9 a)
minus_10 <-0.25-- (p-8 t)
minus_10 <-0.25-- (p-7 t)
```

```
conformation <-0.08-- (p-47 c)
conformation <-0.08-- (p-46 a)
conformation <-0.08-- (p-45 a)
conformation <-0.08-- (p-44 t)
conformation <-0.08-- (p-44 a)
conformation <-0.08-- (p-43 t)
conformation <-0.08-- (p-42 t)
conformation <-0.08-- (p-41 a)
conformation <-0.08-- (p-40 a)
conformation <-0.08-- (p-39 c)
conformation <-0.08-- (p-28 t)
conformation <-0.08-- (p-27 t)
conformation <-0.08-- (p-23 t)
conformation <-0.08-- (p-22 a)
conformation <-0.08-- (p-22 g)
conformation <-0.08-- (p-21 a)
conformation <-0.08-- (p-20 a)
conformation <-0.08-- (p-18 t)
conformation <-0.08-- (p-17 t)
conformation <-0.08-- (p-16 t)
conformation <-0.08-- (p-16 c)
conformation <-0.08-- (p-15 t)
conformation <-0.08-- (p-15 g)
conformation <-0.08-- (p-8 g)
conformation <-0.08-- (p-7 c)
conformation <-0.08-- (p-6 g)
conformation <-0.08-- (p-5 c)
conformation <-0.08-- (p-4 t)
conformation <-0.08-- (p-4 c)
conformation <-0.08-- (p-2 c)
conformation <-0.08-- (p-1 a)
conformation <-0.08-- (p-1 c)
```

Figure 4.2: The Initial Promoter Rule Base in Certainty-Factor Form

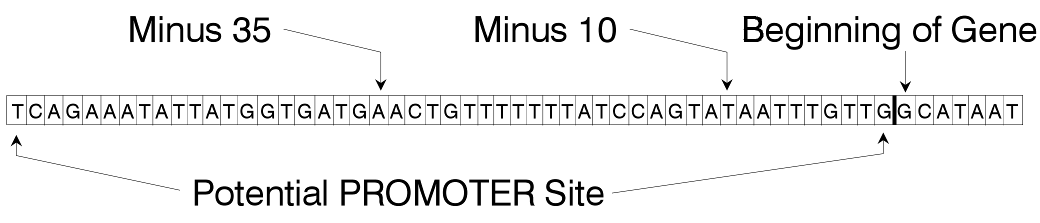

Figure 4.3: Promoter Example 53 From the Rule Base

task is to determine if a Promoter resides in positions P-50 through P-1, with a gene beginning in position P1. An actual example from this data set is shown in Figure 4.3.

These rules are based upon the fact that RNA-polymerase must bind to a DNA sequence at two distinct points, namely DNA positions -35 and -10. The values at these positions weighs heavily upon proper classification. Since this data set is a 50/50 split of Promoters and non-Promoters, the original rule base classifies exactly 50% of the examples correctly, by simply labelling every example negatively. This is precisely the accuracy of random guessing in this domain.

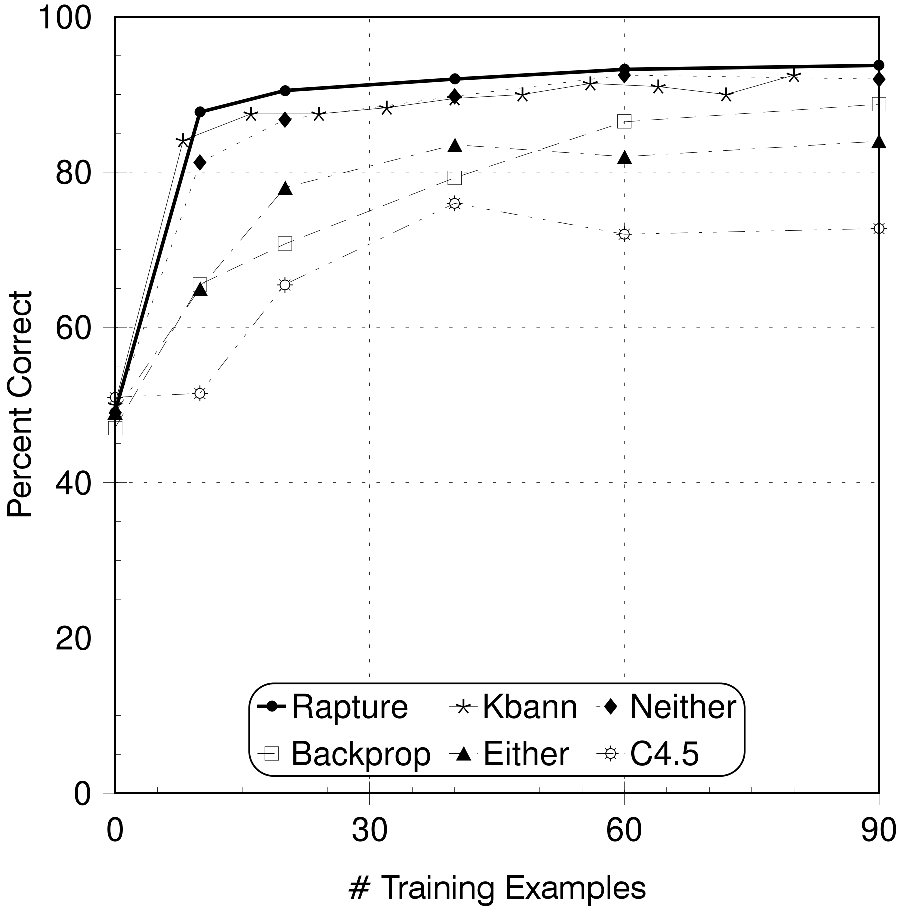

Figure 4.4: Promoter 106 Recognition Accuracy – Standard Systems

Since this problem is a *single-category* problem, where an example either is or is not a Promoter, RAPTURE uses a threshold value (0.9) during training and testing to distinguish positive and negative examples. As discussed in Chapter 3, an example producing an output certainty-factor of 0.9 or greater during training is classified as a positive example (e.g., Promoter). A certainty-factor less than 0.9 results in a classification of non-Promoter. Once the rule base is trained to 100% accuracy (using the 0.9 threshold), the threshold is adjusted before testing. By averaging the certainty-factor of the lowest-scoring positive example with the highest-scoring negative example, a threshold maximizing the distance between the two sets of examples is obtained. The effect of this was minimal in these tests, as the new threshold-value never differed from the original (0.9) by more than 0.05.

Performance in this domain is shown in the graph of Figure 4.4. This is a plot of accuracy in correctly identifying Promoter sites in DNA strands. In this domain, all plots, except those of KBANN are averaged over 25 trials. The KBANN results were obtained independently by Geoff Towell, and were run over differing sets of training examples. The graph clearly demonstrates the advantages of an evidence summing system like RAPTURE over a pure Horn-clause system such as EITHER, a pure inductive system such as C4.5, or a pure connectionist system, such as BACKPROP.

The reasons for this are apparent, upon further examination of the domain. It turns out that there are several potential sites at which hydrogen bonds can form between DNA and a protein. When enough of these bonds form, promoter activity can occur. In order for a set of rules to effectively model this knowledge, some sort of evidence-summing ability is required.

Before any learning takes place (i.e., 0 training examples), all of the systems perform at roughly the same 50% accuracy level, which as mentioned previously is the accuracy produced by random guessing. RAPTURE, however, is able to learn an accurate classification for Promoters much more quickly than the other systems. Whereas RAPTURE performs at a better than 90% level after only 20 training examples, EITHER, BACKPROP, and C4.5 *never* reach this performance level, even after as many as 90 training examples. These results are not terribly surprising, since both EITHER and C4.5 lack the ability to represent M-OF-N style rules, or any other evidence-summing ability. The weak performance of BACKPROP is likely due to the lack of domain knowledge in the neural network.

Though RAPTURE performs the best overall, systems KBANN and NEITHER perform comparably. Both of these systems contain mechanisms for representing M-OF-N style rules, and are given domain knowledge to guide their learning. RAPTURE does, however, learn more quickly as it is able to perform at a greater than 90% level after only 20 training examples, where it takes more than twice this number for both NEITHER and KBANN. Statistical t-tests show that RAPTURE is performing significantly better than the other systems through 40 training examples at at least the 0.05 level. Since the exact data that was used for training the KBANN system was not available to us, performing paired t-tests was impossible, though it does appear that KBANN is performing no better than NEITHER.

By examining the training time plots of Figure 4.5, it is clear that RAPTURE is able to learn this concept quite rapidly in terms of CPU cycles as well. Though C4.5 and NEITHER learn somewhat faster, RAPTURE is outperforming the neural network by more than an order of magnitude. The main reason for this good performance is the domain-knowledge that RAPTURE is able to use to good advantage.

This is one set of rules and examples that RAPTURE is able to train using only the CFBP mechanism. The feature-addition module, and the UPSTART mechanism are never needed to train the examples. This is a good indication that the structure of the domain knowledge is quite accurate. Figure 4.6 presents the *learned* rule base after training on the entire (106) training set. The main difference between this and the original is that each nucleotide has different amounts of influence on the terms in the rules. `Contact` has also been given a much higher certainty factor than `Conformation`, indicating a greater importance in classifying each example. The 0.99 certainty factor for `Contact` indicates that this condition is essentially conclusive for being a promoter.

The fact that this set of rules is much smaller than the complete network required by BACKPROP, RAPTURE–KBANN, and CF-NET also makes training much faster.

The performance of the RAPTURE-variants is plotted in figure 4.7, which further demonstrates the benefit of the initial theory. RAPTURE and RAPTURE–KBANN are both performing at similar levels, though RAPTURE is doing slightly better by the 90 example mark. RAPTURE–NULL, which is given no initial theory performs significantly worse ($\rho = 0.001$), and has performance very similar to that of BACKPROP.

CF-NET is the big disappointment of all of the variants. In this, as well as the other domains,

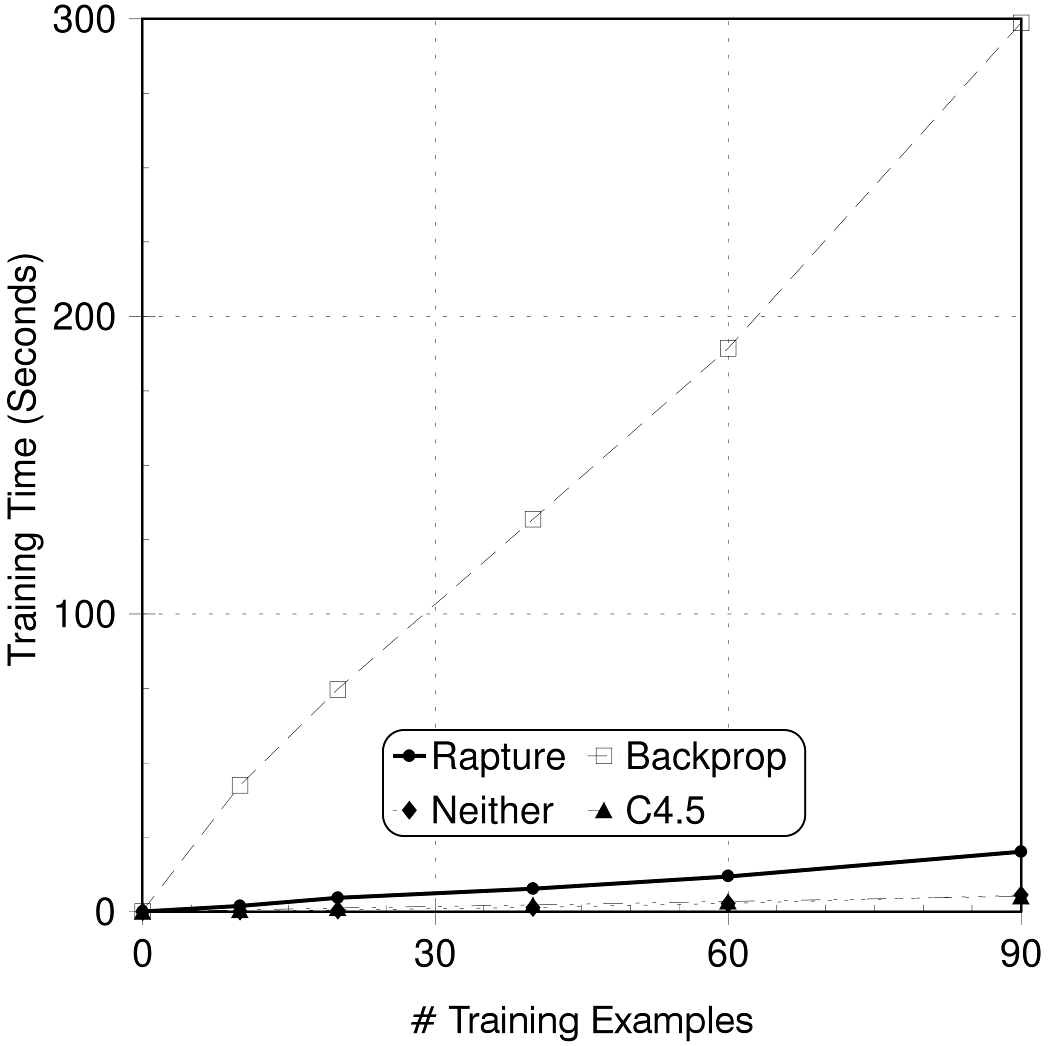

Figure 4.5: Promoter 106 Training Time –Standard Systems

it is largely unable to learn the target concepts. Remembering that CF-Net is CFBP performed over a certainty-factor network with a standard neural-network architecture, two reasons for this failure are apparent. First, this system does not begin with any domain knowledge, but rather is initialized with random certainty-factors. All features from the domain are present in this network, along with a hidden-layer of nodes, providing many degrees of freedom in the fully-connected network. One of the properties of a certainty-factor network and CFBP is that its weight-space contains many more local minima than the corresponding neural network using BackProp. This is due to the non-linearity present in the certainty-factor combining functions. Due to this property, CF-Net converges very rapidly to a local-error minimum not far from the original weights. This generally results in testing accuracy similar to the random-guessing accuracy. Though not plotted for the remainder the domains in this work, its performance is comparable for them all.

By examining the remainder of the plots for this domain (Figure 4.8 and Figure 4.9), many of the same observations are demonstrated. Both Rapture–Kbann and Rapture–Null require significantly more time to train their networks, and Rapture–Kbann produces significantly larger

```
promoter <-0.99-- (contact)          conformation <-0.08-- (p-47 c)
promoter <-0.84-- (conformation)     conformation <-0.11-- (p-46 a)
                                     conformation <-0.15-- (p-45 a)
                                     conformation <-0.05-- (p-44 t)
contact  <-0.79-- (minus_35)         conformation <-0.18-- (p-44 a)
contact  <-0.82-- (minus_10)         conformation <-0.14-- (p-43 t)
                                     conformation <-0.12-- (p-42 t)
                                     conformation <-0.12-- (p-41 a)
minus_35 <-0.25-- (p-37 c)           conformation <-0.07-- (p-40 a)
minus_35 <-0.35-- (p-36 t)           conformation <-0.10-- (p-39 c)
minus_35 <-0.29-- (p-35 t)           conformation <-0.16-- (p-28 t)
minus_35 <-0.37-- (p-34 g)           conformation <-0.10-- (p-27 t)
minus_35 <-0.31-- (p-33 a)           conformation <-0.09-- (p-23 t)
minus_35 <-0.28-- (p-32 c)           conformation <-0.06-- (p-22 a)
minus_35 <-0.31-- (p-31 a)           conformation <-0.11-- (p-22 g)
                                     conformation <-0.07-- (p-21 a)
                                     conformation <-0.14-- (p-20 a)
minus_10 <-0.20-- (p-14 t)           conformation <-0.18-- (p-18 t)
minus_10 <-0.23-- (p-13 t)           conformation <-0.05-- (p-17 t)
minus_10 <-0.27-- (p-13 a)           conformation <-0.06-- (p-16 t)
minus_10 <-0.29-- (p-12 t)           conformation <-0.09-- (p-16 c)
minus_10 <-0.31-- (p-12 a)           conformation <-0.08-- (p-15 t)
minus_10 <-0.24-- (p-11 t)           conformation <-0.07-- (p-15 g)
minus_10 <-0.30-- (p-11 a)           conformation <-0.06-- (p-8 g)
minus_10 <-0.28-- (p-10 a)           conformation <-0.09-- (p-7 c)
minus_10 <-0.25-- (p-9 t)            conformation <-0.06-- (p-6 g)
minus_10 <-0.27-- (p-9 a)            conformation <-0.13-- (p-5 c)
minus_10 <-0.25-- (p-8 t)            conformation <-0.14-- (p-4 t)
minus_10 <-0.29-- (p-7 t)            conformation <-0.12-- (p-4 c)
                                     conformation <-0.10-- (p-2 c)
                                     conformation <-0.12-- (p-1 a)
                                     conformation <-0.13-- (p-1 c)
```

Figure 4.6: The Revised Promoter Rule Base

rule bases. This is due to the fact that RAPTURE–KBANN begins with such a large initial network, and not enough links are able to decay away due to the convergence properties of certainty factors. Rule base complexities of all networks are measured by the number of links they contain. Interestingly, RAPTURE–NULL actually produces much smaller rule bases than RAPTURE. In fact, for 10 training examples, RAPTURE–NULL only requires slightly more than 1 rule in order to classify the examples correctly. This is a clear demonstration of the value of the domain knowledge, as these simpler rule bases result in worse performance.

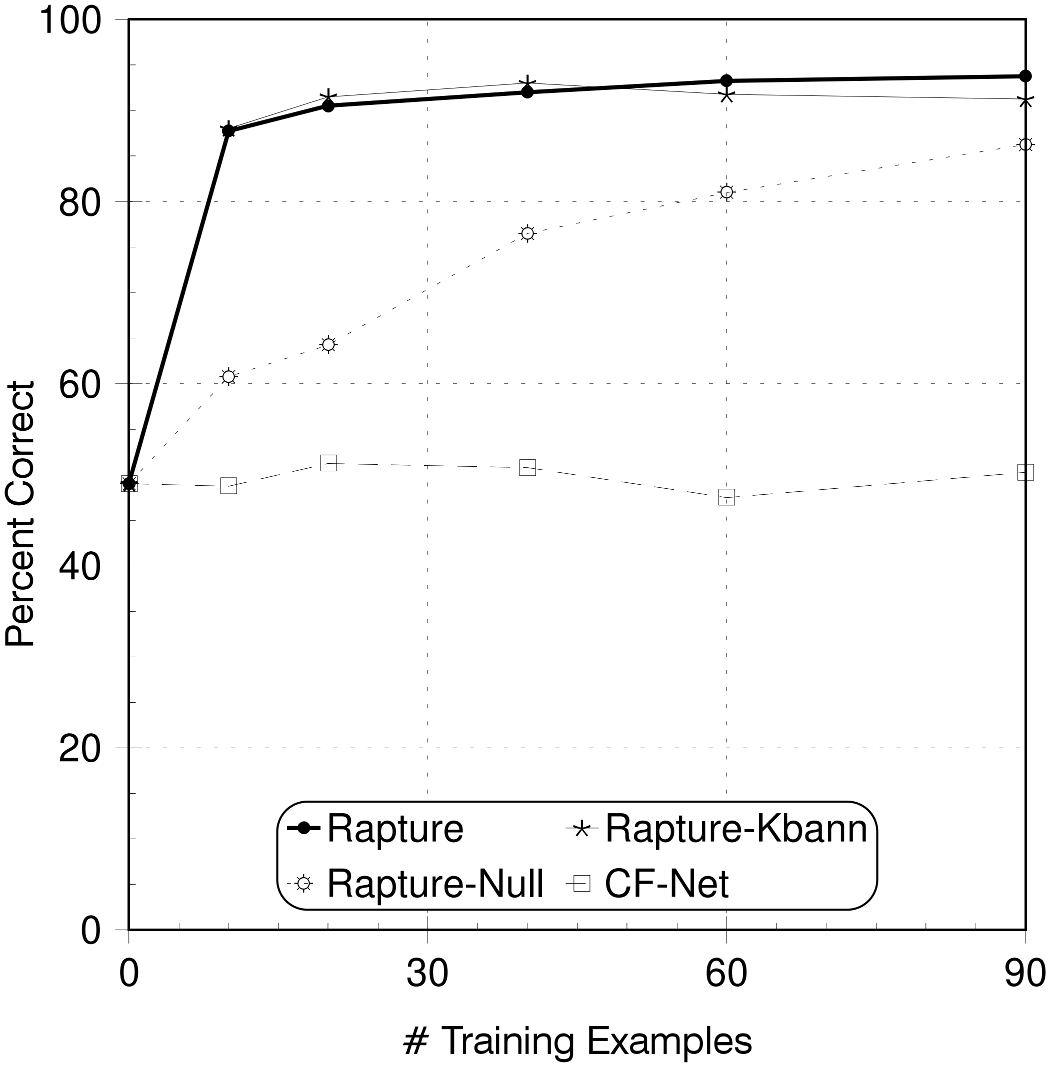

Figure 4.7: Promoter 106 Recognition Accuracy – Rapture Ablations

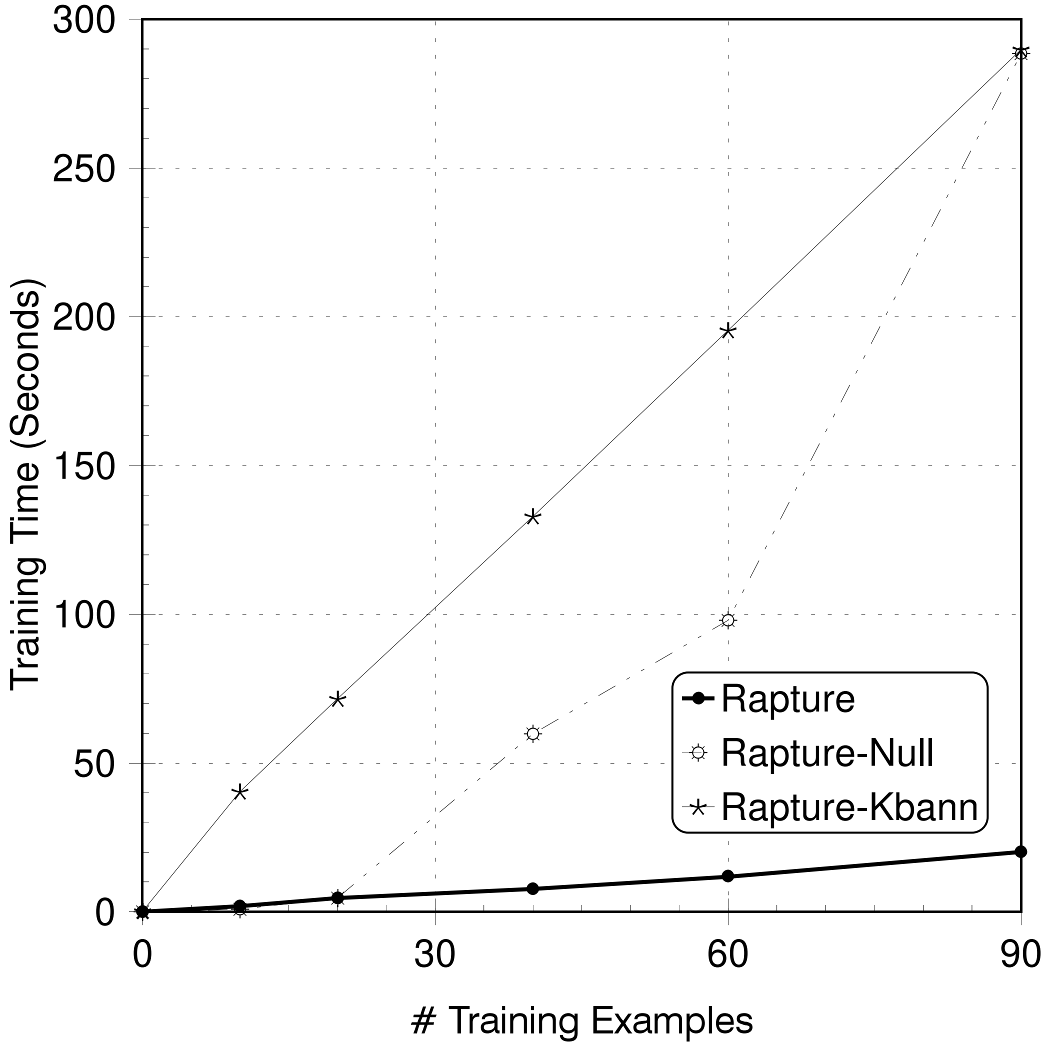

Figure 4.8: Promoter 106 Training Time – Rapture Ablations

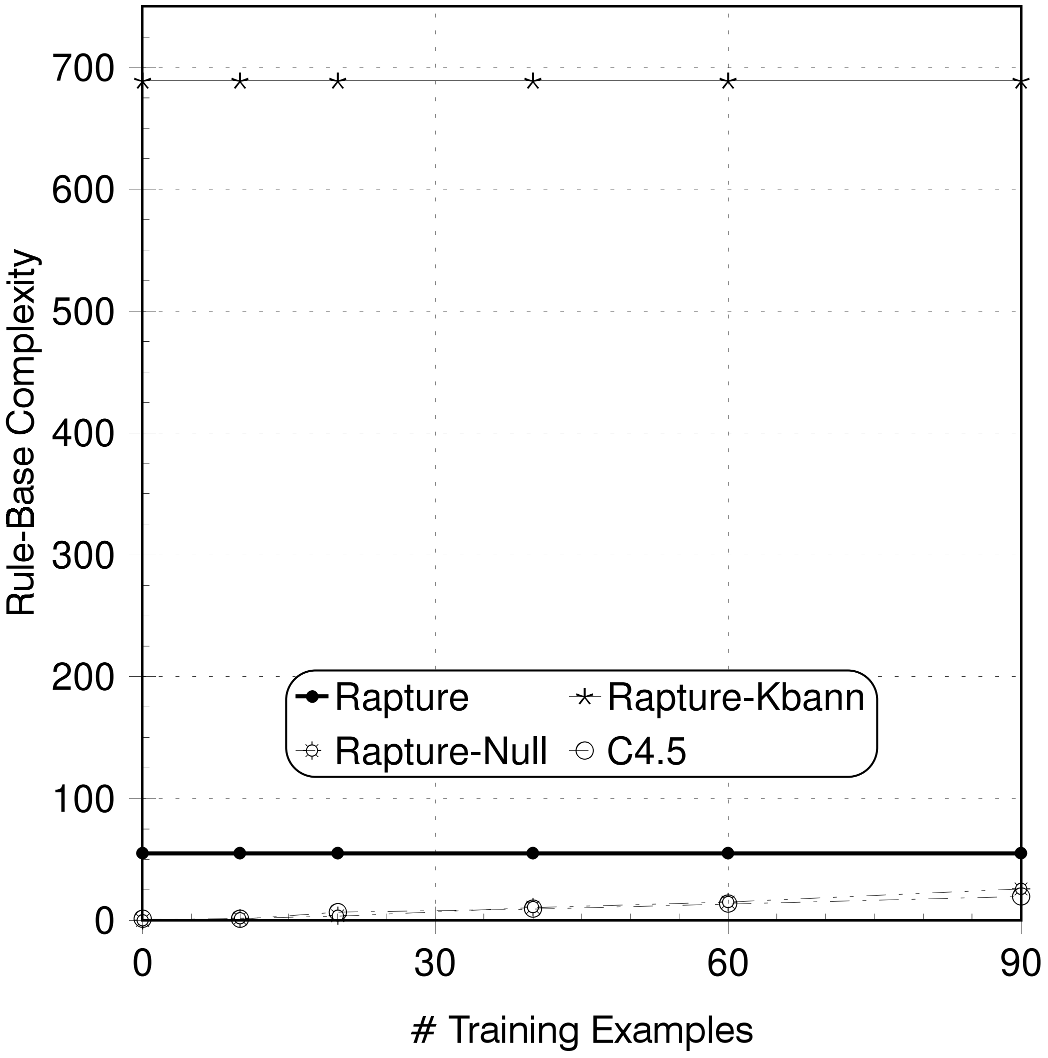

Figure 4.9: Promoter Rule-Base Complexity

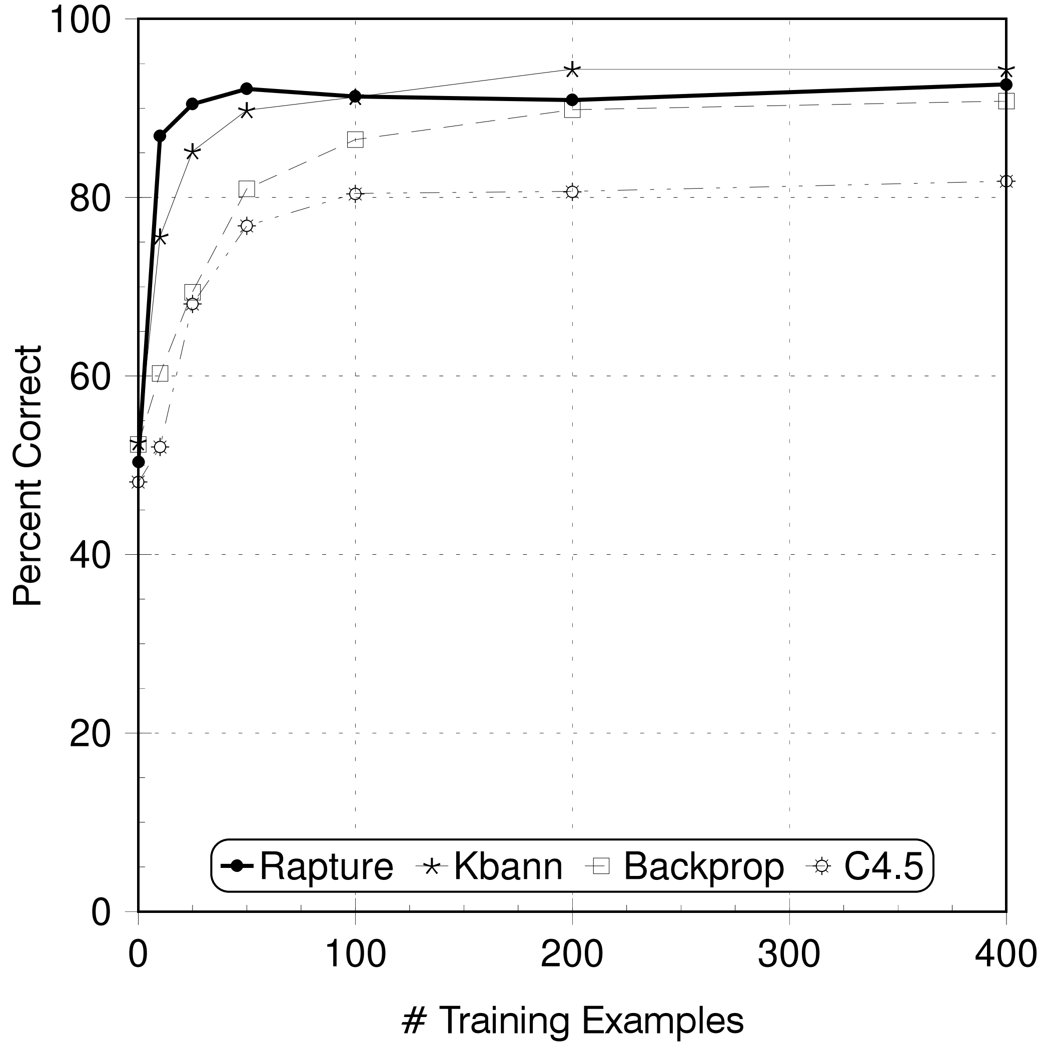

Figure 4.10: Promoter 468 Recognition Accuracy – Standard Systems

4.5.2 The 468-example Promoter Data Set

This data set is a more extensive set of examples of Promoters and non-Promoters that comes from the Human Genome Project (Alberts, 1988). Once again, the examples are an even mix of 234 Promoters and 234 non-Promoters, though the key difference in these examples is the presence of missing features. Approximately 15% of the examples have missing values for one or more of the features. Naturally this set of examples uses the same initial rule base as the previous set.

Performance results are presented in Figure 4.10. While RAPTURE outperforms the other systems up through 50 training examples, KBANN surpasses RAPTURE at the 200 example mark. While once again paired t-tests were not possible to run with KBANN, it does appear that RAPTURE and KBANN are performing roughly the same by 400 example mark. RAPTURE performs significantly better than BACKPROP up through 200 training examples ($\rho = 0.1$ level), and significantly better than C4.5 ($\rho = 0.001$ level) throughout all of training. These results seem to corroborate the need for an ability for evidence-summation. C4.5 is the only system presented here without this

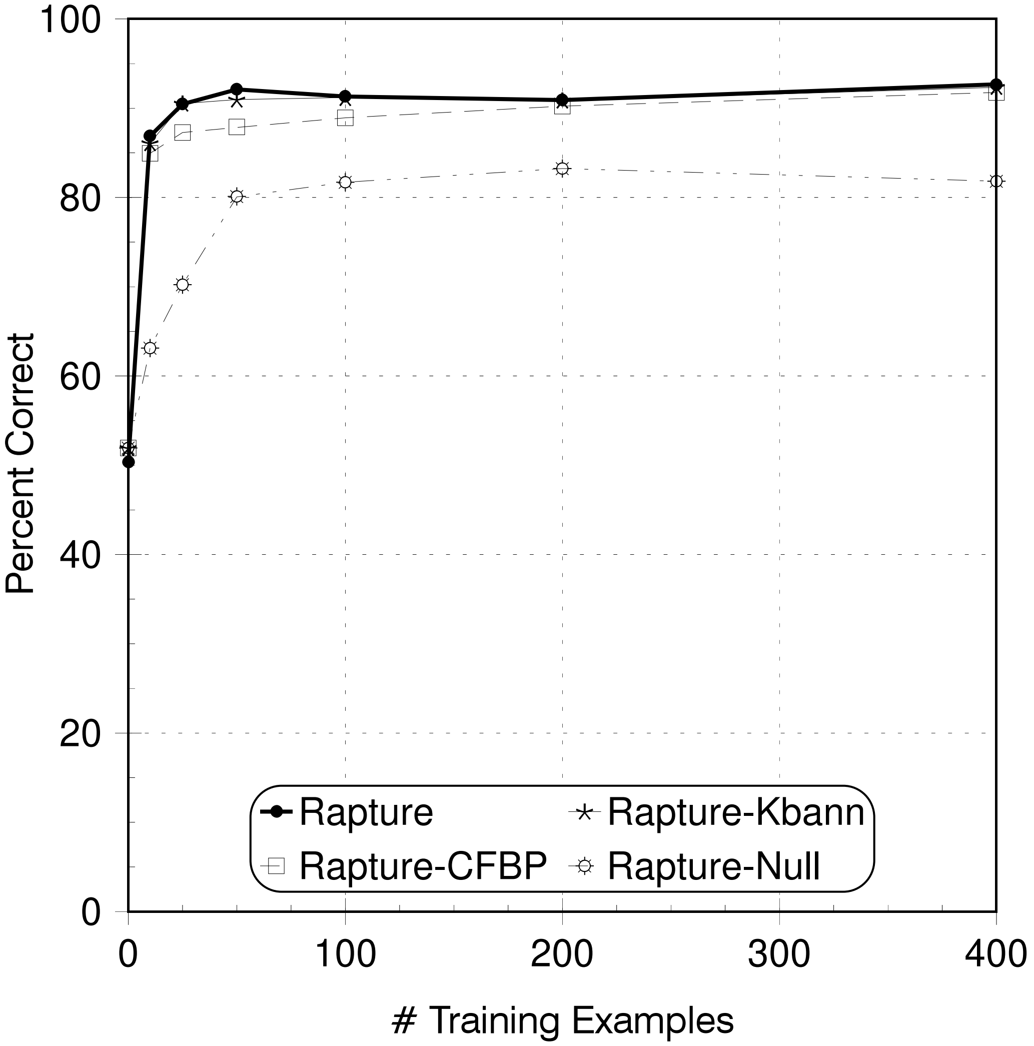

Figure 4.11: Promoter 468 Recognition Accuracy – Rapture Ablations

ability, and its performance suffers. Further, the systems with the advantage of domain knowledge have an edge in performance.

These observations are further demonstrated in the plot of Figure 4.11. The Rapture--variants with domain knowledge are all performing at nearly equivalent levels, though Rapture is performing significantly better ($\rho = 0.05$ level) than Rapture–CFBP with 20 and 50 training examples. Only Rapture–Null with its lack of domain knowledge suffers, as its performance is significantly worse than that of Rapture at the 0.001 level throughout. Its performance very similar to that of C4.5.

Unlike the 106-example data set, this time Rapture requires the use of its architecture-modification routines in order to fully train the examples. This is most noticeable in the increased training time with the larger-sized training sets. These effects can be clearly seen in Figure 4.12. For 400 training examples, Rapture requires a considerable number of new links in order to fully train the network. Due primarily to usage of the Upstart module, Rapture actually takes slightly

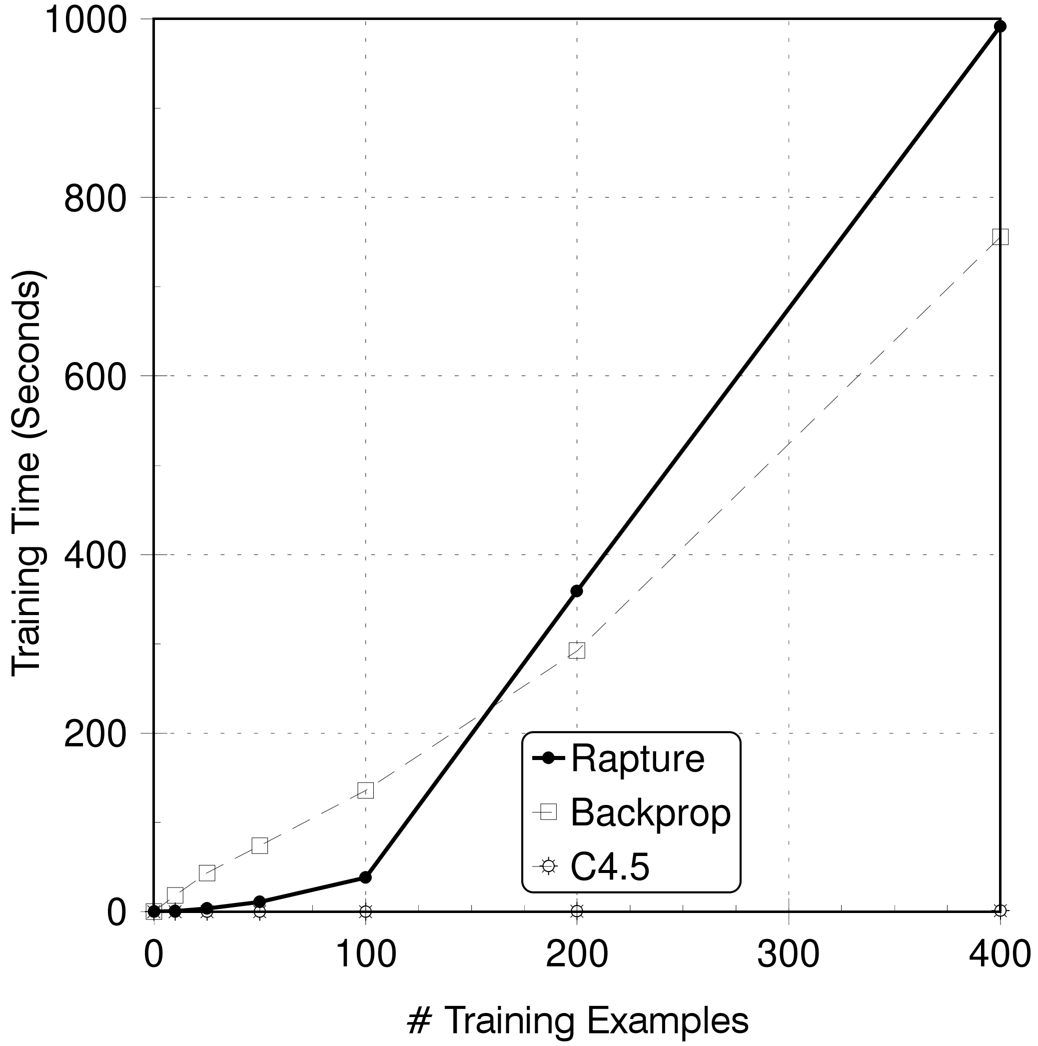

Figure 4.12: Promoter 468 Training Time – Standard Systems

longer than BackProp to learn 400 training examples.

The plot of Figure 4.13 clearly demonstrates the effects of the Upstart algorithm. Rapture–Null with no domain knowledge requires a great deal of time to train the network to learn 400 training examples. The majority of time is spent in the Upstart algorithm, where a new sub-network must be built each time it is called. This sub-network must learn to distinguish a handful of exception-examples from the rest of the entire training set, which may or may not be a simple task. When it is not, the considerable effort necessary is reflected in the training time, as well as the complexity of the resulting rule base. This can be seen in Figure 4.14. Whereas Rapture–Null is able to learn very simple rule bases in the smaller Promoter data set, it is forced to build considerably more complex ones as the number of training examples grows.

As can be seen in the complexity plot, the rule bases of Rapture–Kbann are significantly larger than the rest. This is due both to its requirement that all features from the domain must be present in the network with complete connectivity between layers. Since complexity is measured

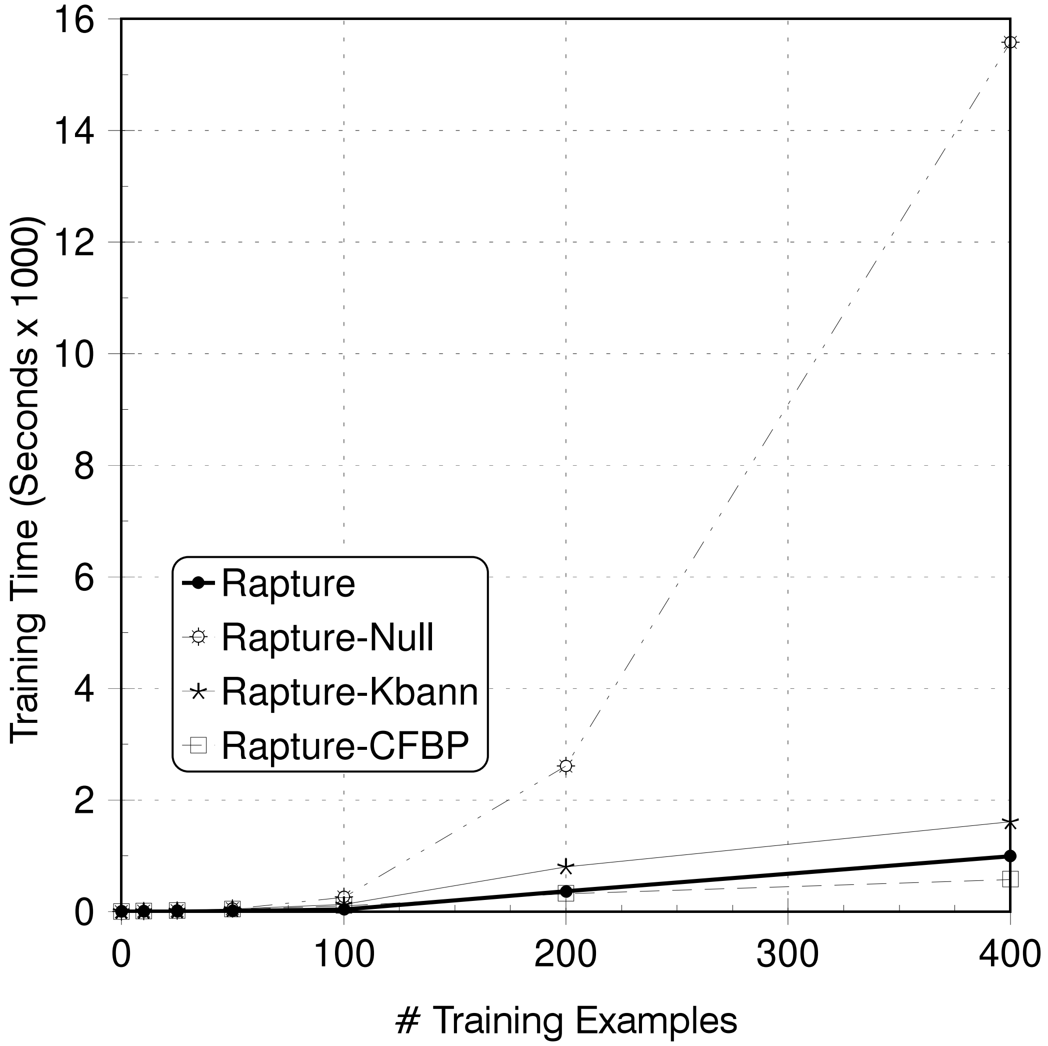

Figure 4.13: Promoter 468 Training Time – Rapture Ablations

here as the number of links in the network, this greatly effect RAPTURE–KBANN.

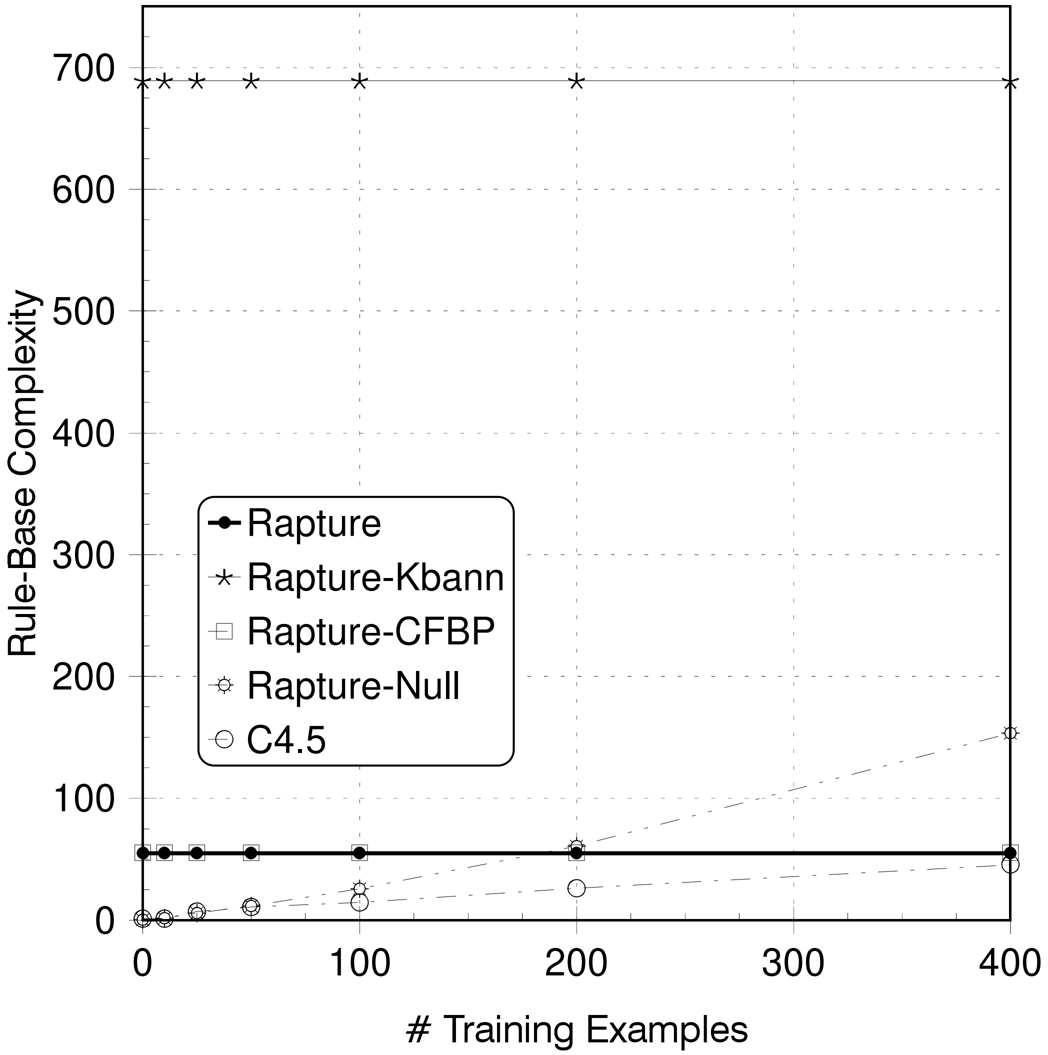

Figure 4.14: Promoter 468 Rule-Base Complexity

4.6 Splice-Junction Results

The other DNA-sequencing domain discussed in this work is the Splice-Junction Domain. Throughout the more than 3,000,000,000 nucleotides that exist in human DNA, as much as 90% of these do not code for anything, and are known as junk DNA. As protein is created in living organisms, the junk DNA gets *spliced* out of the sequence, and protein is built from the remaining coding-DNA. Learning to recognize the borders between coding and junk DNA is the Splice-Junction task.

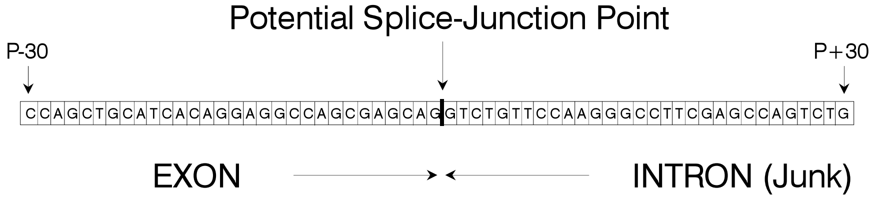

Figure 4.15: Exon-Intron Example 1 From the Rule Base

Areas of DNA that contain genetic information are called Exons, and the areas of interspersed junk DNA are known as Introns. These two different areas of a DNA strand give rise to two types of borders that must be identified. Intron-Exon (IE) borders (also known as acceptors) delimit the end of junk DNA, and the beginning of genetic information. Similarly, Exon-Intron (EI) borders (known as donors) delimit the end of a sequence of genetic information, and the beginning of junk DNA.

The rules and examples for this domain come from M. Noordewier (Noordewier et al., 1991), and consists of a database of 3190 examples of DNA strings. Of these examples, there are 768 (or 24%) examples of IE borders, 767 (24%) EI borders, and 1655 (52%) negative examples, which are randomly chosed DNA sequences containing neither border. Each example in this domain is represented by a string of 60 nucleotides. The first 30 features are labelled P-30 through P-1 (positions -30 to -1), and the second 30 are labelled P+1 through P+30 (positions 1 to 30). The learning task is to determine which of the above three border types (including negative) exists between nucleotides P-1 and P+1.

M. Noordewier, derived these rules from information found in Watson, Roberts, Steitz, and Weiner (1987). These rules specify common patterns that are present at border sites. IE borders are characterized by regions dense with C and T values (known as pyramidine-rich regions). The rules also contain information for recognizing Stop-Codons. The certainty-factor rules for this task are presented in Figure 4.16.

In order to train RAPTURE for this task, use of a threshold was again necessary due to the negative category. A 0.9 threshold was used throughout training, so that if both EI and IE output categories received certainty-factors less than 0.9, a negative classification would result. Once the network was trained, the threshold was again (as in the Promoter domain) adjusted to maximize the distance between positive (i.e., borders) and negative examples. In order to do this, an average was made between the certainty-factor of the positive example with the lowest value in its category, and the certainty-factor of the negative example that scored the highest in either border category. This domain is quite similar to the Promoter domain in that a set of rules capable of evidence summation should be at an advantage. Test results appear to confirm this hypothesis.

```
EI <-0.23-- (p-3 A)
EI <-0.23-- (p-3 C)
EI <-0.23-- (p-2 A)
EI <-0.23-- (p-1 G)
EI <-0.23-- (p1 G)
EI <-0.23-- (p2 T)
EI <-0.23-- (p3 A)
EI <-0.23-- (p3 G)
EI <-0.23-- (p4 A)
EI <-0.23-- (p5 G)
EI <-0.23-- (p6 T)
EI <-0.68-- (NOT EI-stop)

IE <-0.68-- (pyramidine-rich)
IE <-0.25-- (p-3 C)
IE <-0.25-- (p-3 T)
IE <-0.25-- (p-2 A)
IE <-0.25-- (p-1 G)
IE <-0.25-- (p1 G)
IE <-0.68-- (NOT IE-stop)

EI-stop <- 1.0 -- (p-3 T)(p-2 A)(p-1 A)
EI-stop <- 1.0 -- (p-3 T)(p-2 A)(p-1 G)
EI-stop <-1.0-- (p-3 T)(p-2 G)(p-1 A)
EI-stop <-1.0-- (p-4 T)(p-3 A)(p-2 A)
EI-stop <-1.0-- (p-4 T)(p-3 A)(p-2 G)
EI-stop <-1.0-- (p-4 T)(p-3 G)(p-2 A)
EI-stop <-1.0-- (p-5 T)(p-4 A)(p-3 A)
EI-stop <-1.0-- (p-5 T)(p-4 A)(p-3 G)
EI-stop <-1.0-- (p-5 T)(p-4 G)(p-3 A)
```

```
IE-stop <-1.0-- (p1 T)(p2 A)(p3 A)
IE-stop <-1.0-- (p1 T)(p2 A)(p3 G)
IE-stop <-1.0-- (p1 T)(p2 G)(p3 A)
IE-stop <-1.0-- (p2 T)(p3 A)(p4 A)
IE-stop <-1.0-- (p2 T)(p3 A)(p4 G)
IE-stop <-1.0-- (p2 T)(p3 G)(p4 A)
IE-stop <-1.0-- (p3 T)(p4 A)(p5 A)
IE-stop <-1.0-- (p3 T)(p4 A)(p5 G)
IE-stop <-1.0-- (p3 T)(p4 G)(p5 A)

pyramidine-rich <-0.32-- (p-15 C)
pyramidine-rich <-0.32-- (p-15 T)
pyramidine-rich <-0.32-- (p-14 C)
pyramidine-rich <-0.32-- (p-14 T)
pyramidine-rich <-0.32-- (p-13 C)
pyramidine-rich <-0.32-- (p-13 T)
pyramidine-rich <-0.32-- (p-12 C)
pyramidine-rich <-0.32-- (p-12 T)
pyramidine-rich <-0.32-- (p-11 C)
pyramidine-rich <-0.32-- (p-11 T)
pyramidine-rich <-0.32-- (p-10 C)
pyramidine-rich <-0.32-- (p-10 T)
pyramidine-rich <-0.32-- (p-9 C)
pyramidine-rich <-0.32-- (p-9 T)
pyramidine-rich <-0.32-- (p-8 C)
pyramidine-rich <-0.32-- (p-8 T)
pyramidine-rich <-0.32-- (p-7 C)
pyramidine-rich <-0.32-- (p-7 T)
pyramidine-rich <-0.32-- (p-6 C)
pyramidine-rich <-0.32-- (p-6 T)
```

Figure 4.16: The Splice-Junction Rule Base in Certainty-Factor Format

As can be seen in Figure 4.17, Rapture appears to be outperforming the other systems. Both BackProp and C4.5 are identifying splice-junction points significantly less accurately than Rapture ($\rho = 0.05$, $\rho = 0.001$). C4.5 is clearly hindered by its inability to model evidence-summing, and BackProp with no domain knowledge requires a considerable number of training examples in order to be competitive.

The two systems with both domain knowledge and evidence-summing ability are clearly performing the best. Despite not being able to run t-tests on Kbann, it does appear that Kbann is performing very near the BackProp level at the 400 training example point. T-tests confirm that Rapture is performing significantly better than BackProp throughout at the 0.1 level at 400 training examples, and at least the 0.05 levels earlier.

Figure 4.18 is a plot of the Rapture-variants, which shows the effect of each of the Rapture components. Beginning with Rapture–Null, which has the worst performance, the contribution of the initial theory can be seen. Its performance is very similar to that of C4.5.

Rapture–CFBP performs better than BackProp when there are fewer than 200 training examples, but does quite similarly after this point. This early edge is most likely due Rapture–CFBP-'s domain knowledge, which is of the greatest help with smaller numbers of examples. Note also than Rapture–CFBP and Kbann are performing nearly identically.

The performances of Rapture and Rapture–Add are also virtually the same, indicating

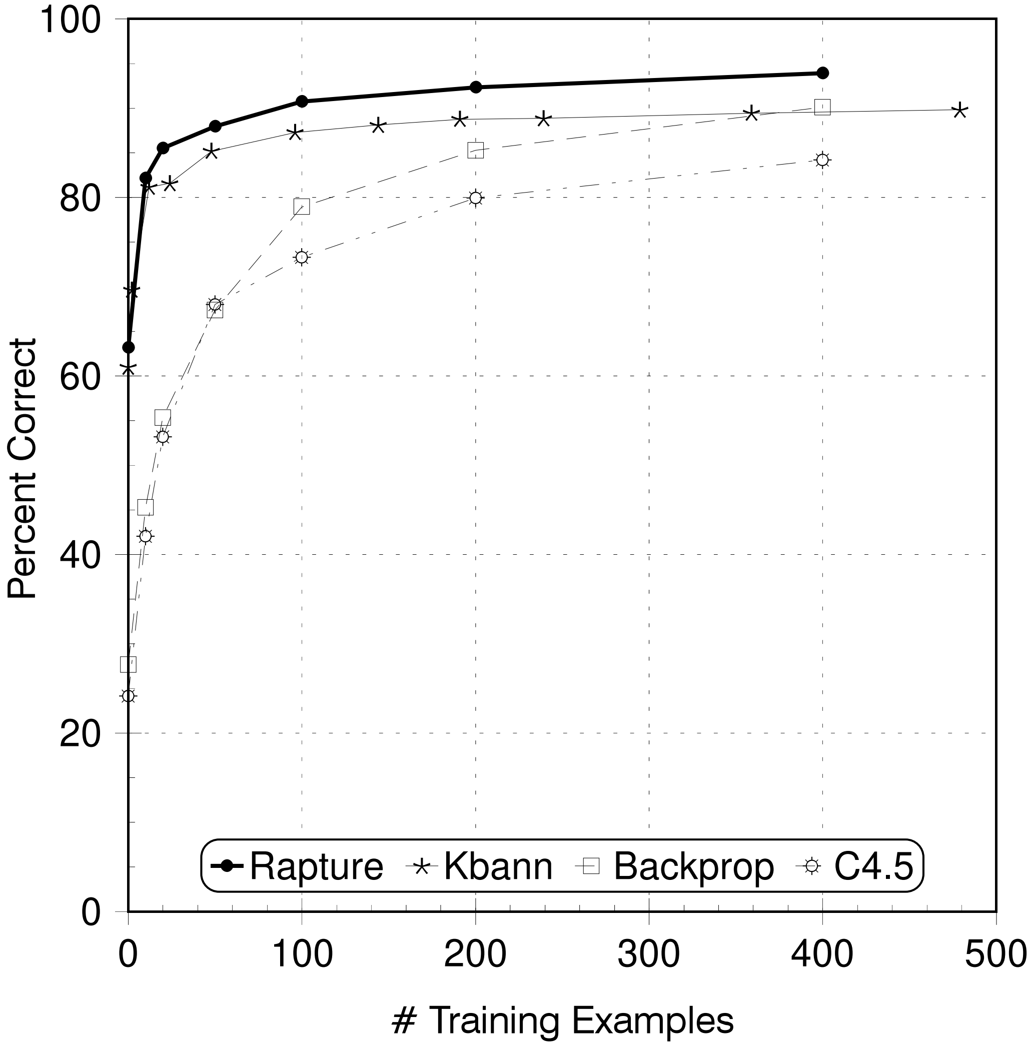

Figure 4.17: Splice-Junction Recognition – Standard Systems

that RAPTURE found some use in adding new rules into the network, but had little need for the UPSTART algorithm. Any usage of UPSTART had little effect upon the system performance.

Some of these same conclusions can be drawn by examining the training-time plots of Figure 4.19 and Figure 4.20. RAPTURE learns faster than BACKPROP, but is no match in speed for RAPTURE–CFBP. It is apparent that RAPTURE only takes slightly more time than RAPTURE–ADD-, again indicating only moderate UPSTART usage. RAPTURE–NULL learns correct target concepts rather quickly with small numbers of training examples, but as the numbers get larger, its lack of domain knowledge causes it to suffer. It is forced to make extensive use of the UPSTART module, which reflects is the increased training times.

In Figure 4.21, it is apparent that RAPTURE–KBANN requires produces larger rule bases than the other variants, due to its domain-complete network.

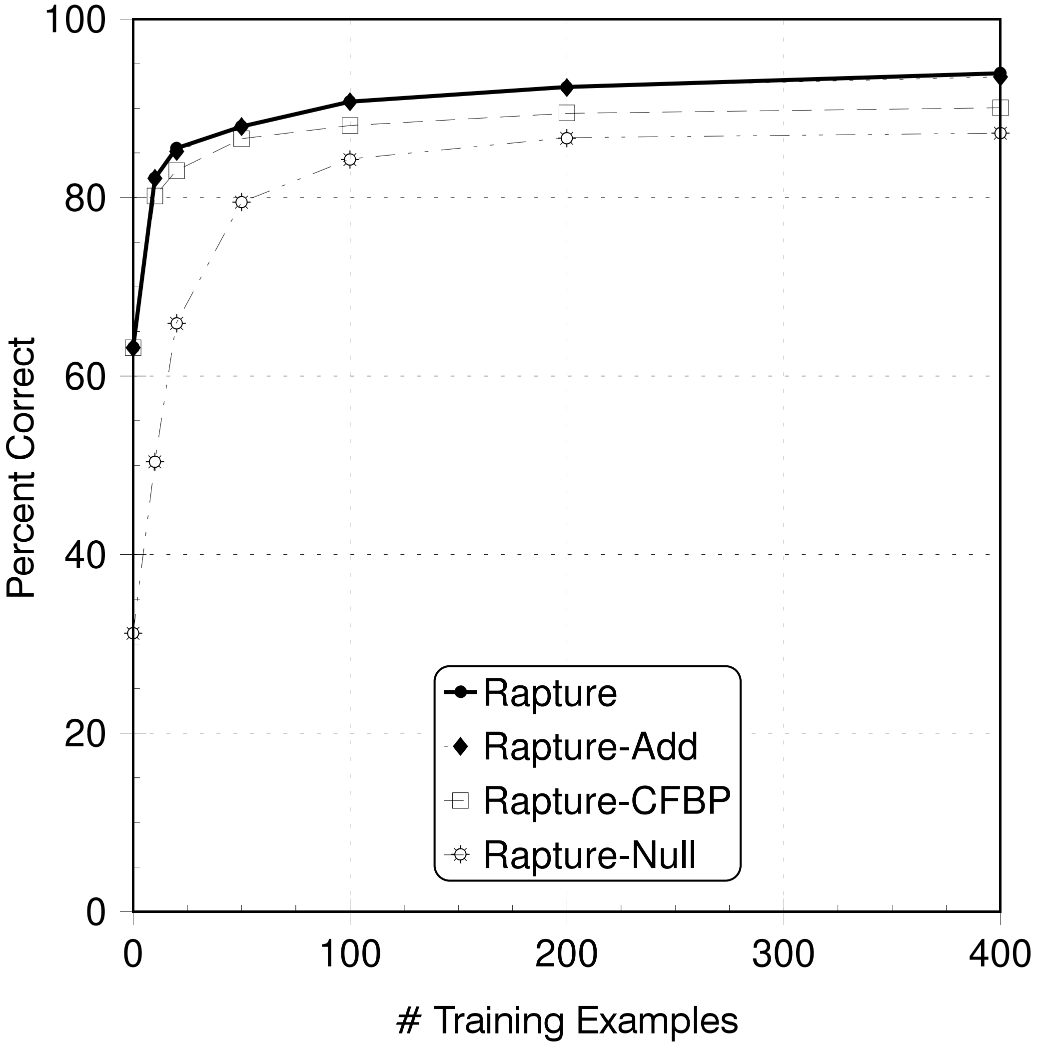

Figure 4.18: Splice-Junction Recognition – Rapture Ablations

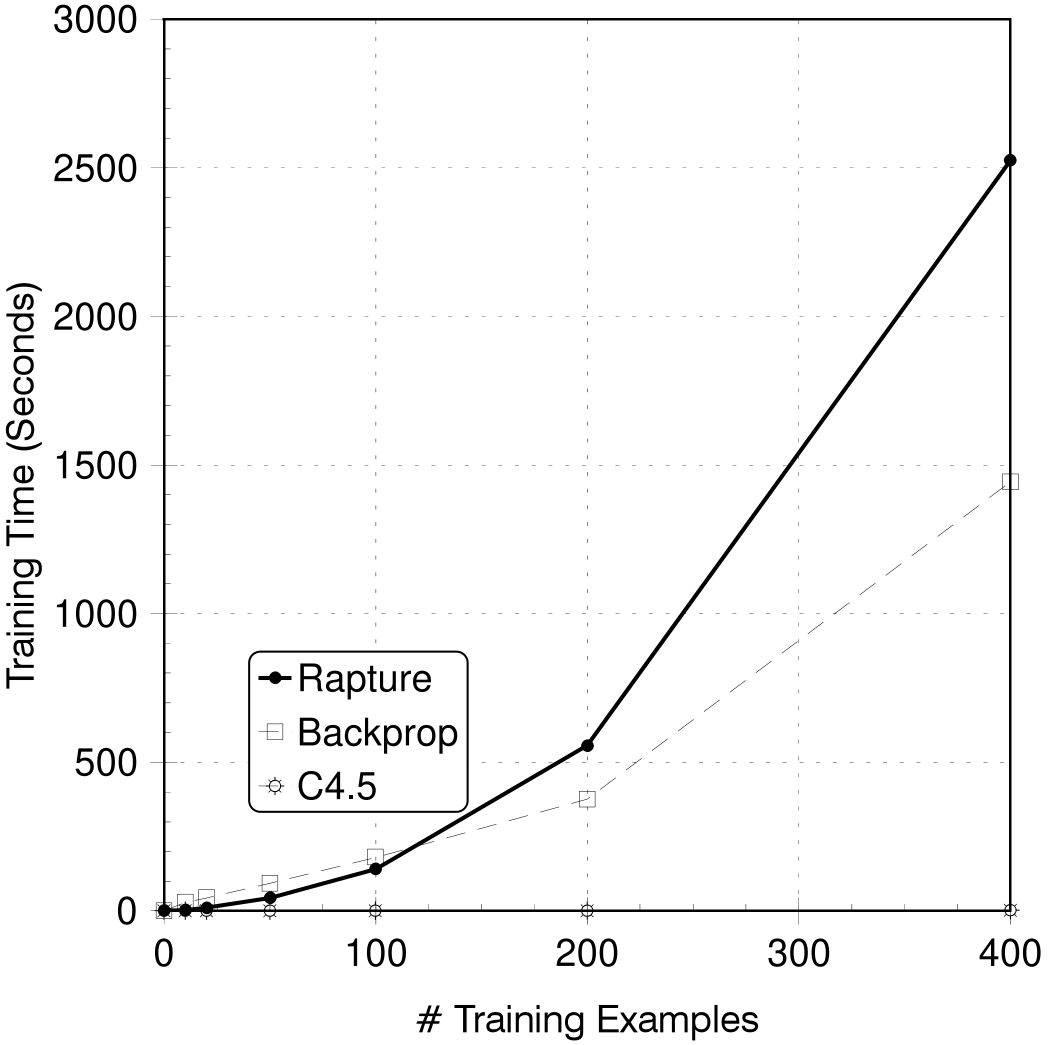

Figure 4.19: Splice-Junction Training Time – Standard Systems

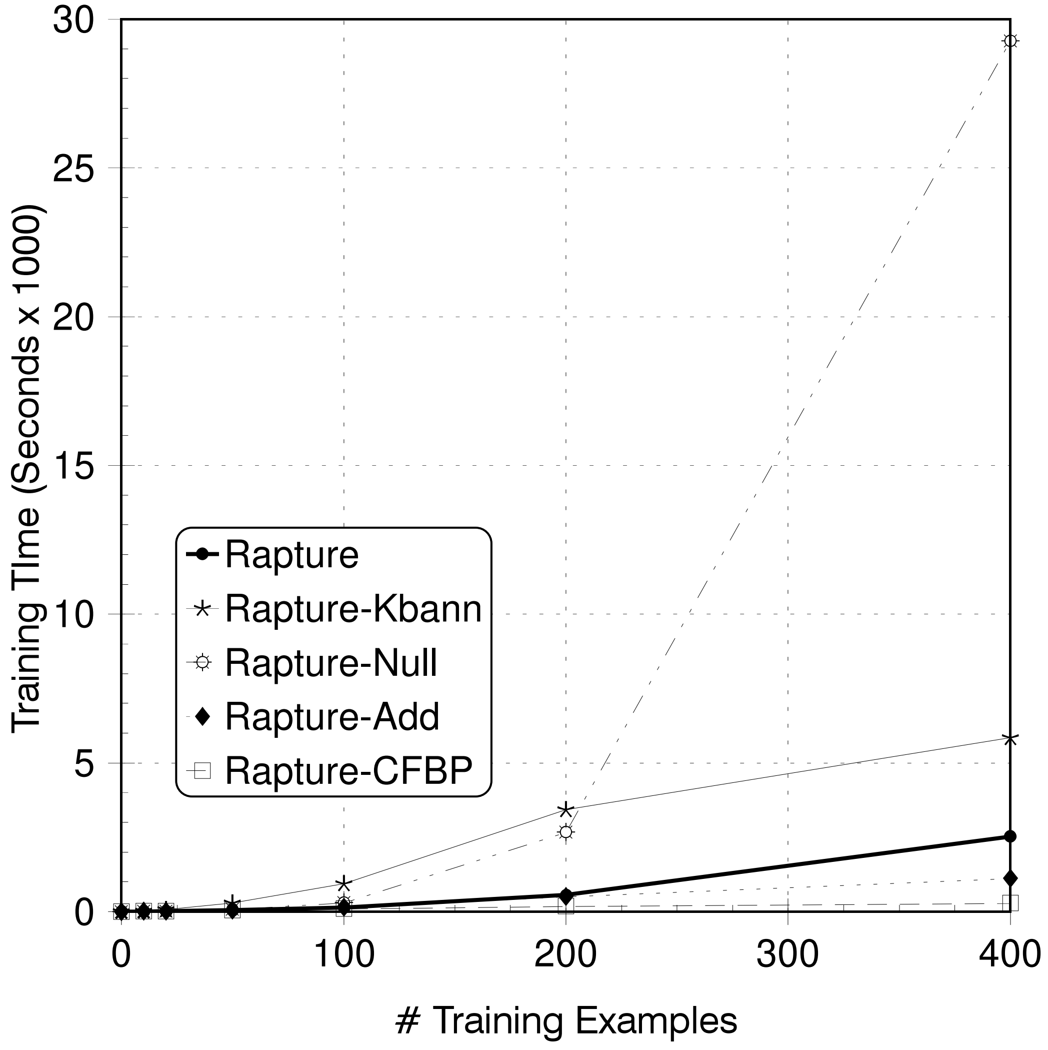

Figure 4.20: Splice-Junction Training Time – Rapture Ablations

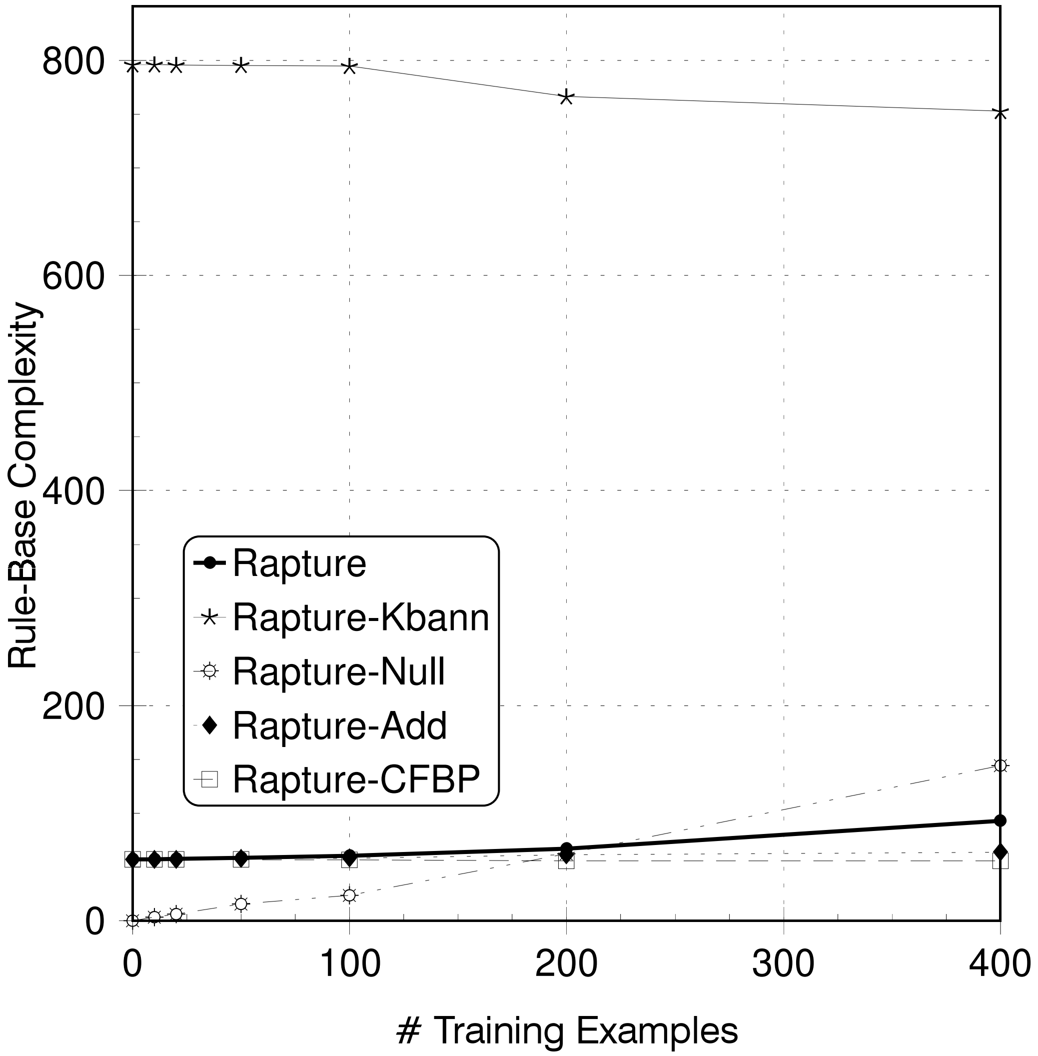

Figure 4.21: Splice-Junction Rule Base Complexity

4.7 Mycin Diagnosis Results

Whereas the previous sections dealt with problems of DNA sequencing, a domain commonly believed to possess the M-OF-N property, this section and the next both deal with problems of diagnosis.

The first of these is the MYCIN system (Shortliffe, 1976; Buchanan & Shortliffe, 1984), which was designed to provide consultative advice for diagnosing and treating infectious diseases. Developed as part of the Stanford Heuristic Programming project, it was heavily influenced by earlier expert systems, most notably DENDRAL (Feigenbaum, Buchanan, & Lederberg, 1971). DENDRAL was a system for determining molecular structures of complex organic chemicals from mass spectrograms. This was the first AI program that put more emphasis on domain-specific knowledge than on general problem-solving methods, which became a large part of MYCIN.

Domain knowledge in MYCIN is represented as rules using certainty factors. The use of these rules to represent uncertainty is one to the major contributions of the MYCIN system. Certainty-factors, and their properties are fully described in Chapter 3. RAPTURE borrows from this formalism for representing the rule bases that it revises.

A version of the MYCIN rule base was prepared for RAPTURE consisting of 99 examples of solved cases (patients) of infectious diseases drawn from the Stanford Medical Center (Ma & Wilkins, 1991). Each patient is described using a vector of 262 features ranging from sex and age to the duration of headaches, and each may have any one of nine infectious diseases, Many features are missing for any given patient, and there are a large number of continuously-valued features. The complete set of rules used in these experiments contains 137 certainty-factor rules, and is listed in Appendix B.

Figure 4.22 plots the learning performance of RAPTURE against both BACKPROP and C4.5. Neither of these competing systems benefit from domain knowledge, and this is reflected in their performances. RAPTURE's accuracy is clearly superior early on ($\rho = 0.001$), and even through 80 examples is doing significantly better ($\rho = 0.05$) than the other systems. While C4.5 is never competitive with RAPTURE, BACKPROP with its evidence-summing ability does come to within four percentage points.

The comparisons of the RAPTURE variants shown in Figure 4.23 shows that there is little advantage gained through architecture modification. RAPTURE, CFBP, and RAPTURE–ADD are all performing at levels that have no significant differences. The advantages of the domain knowledge are demonstrated by the relatively poor performance of RAPTURE–NULL, with performance that is very similar to that of C4.5. The performance of RAPTURE–KBANN is considerably better than that of RAPTURE–NULL, but is slightly, yet significantly less than that of RAPTURE. RAPTURE–KBANN is performing very nearly the same as BACKPROP from 60 examples on.

An examination of Figure 4.24 through Figure 4.26 leads to several observations. The most obvious is the fact that RAPTURE takes a big time hit once there are more than 20 training examples. This is primarily due to the use of the UPSTART algorithm, whose effects can also be seen by examining rule base complexities. A good indication of the accuracy of the original set of rules is that this architecture modification does little, if any good at improving its overall performance. The RAPTURE rule base is more than 20% larger than RAPTURE–ADD and CFBP. RAPTURE does, however, perform significantly better, both in terms of training time, and complexity of the resulting rule bases, than RAPTURE–KBANN. RAPTURE–KBANN's complexity is so large on this

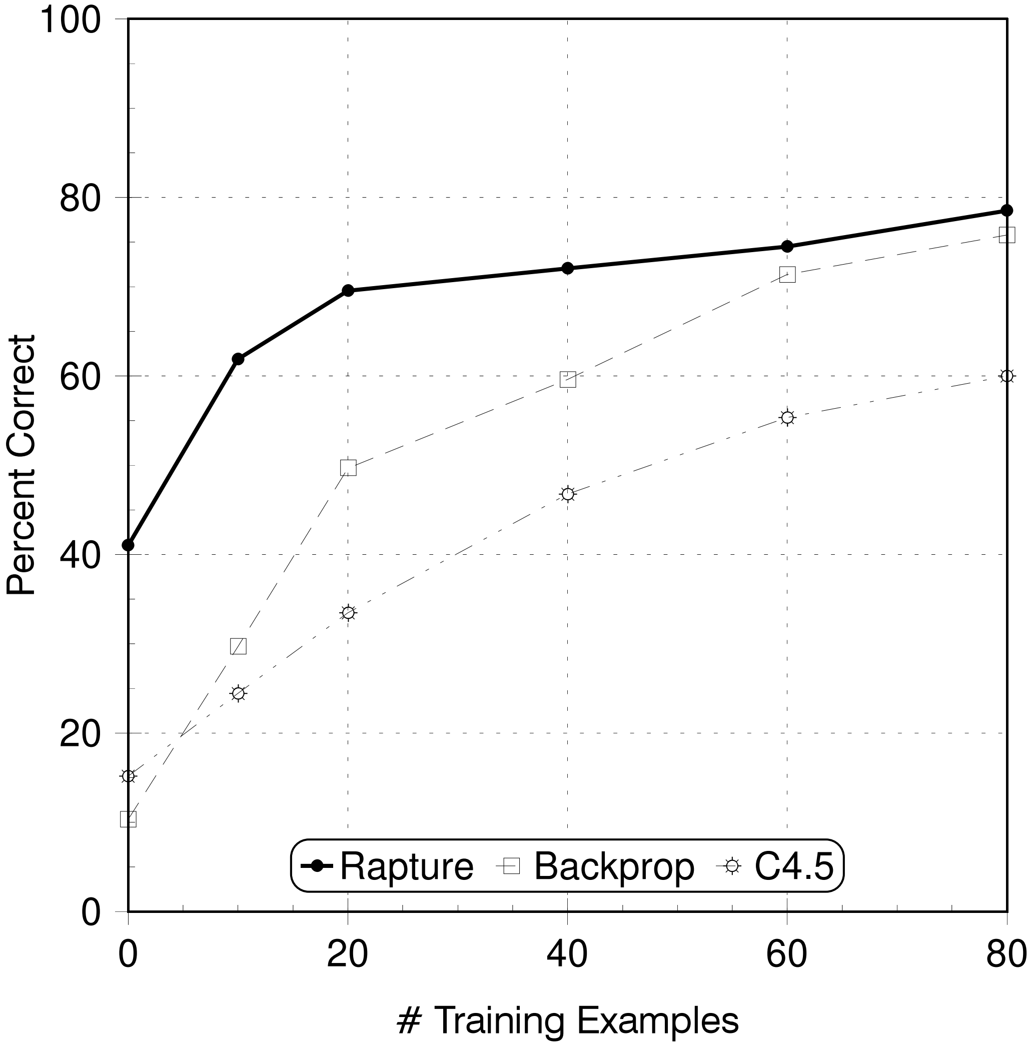

Figure 4.22: Mycin Disease Diagnosis Accuracy – Standard Systems

problem, that the plot represents the complexity divided by 10. Because of this, RAPTURE–KBANNS training time suffers greatly.

Similar to the DNA rule bases, RAPTURE–NULL is able to learn much simpler rule bases than most of the other systems. Again, however, this results is a significant drop in performance accuracy.

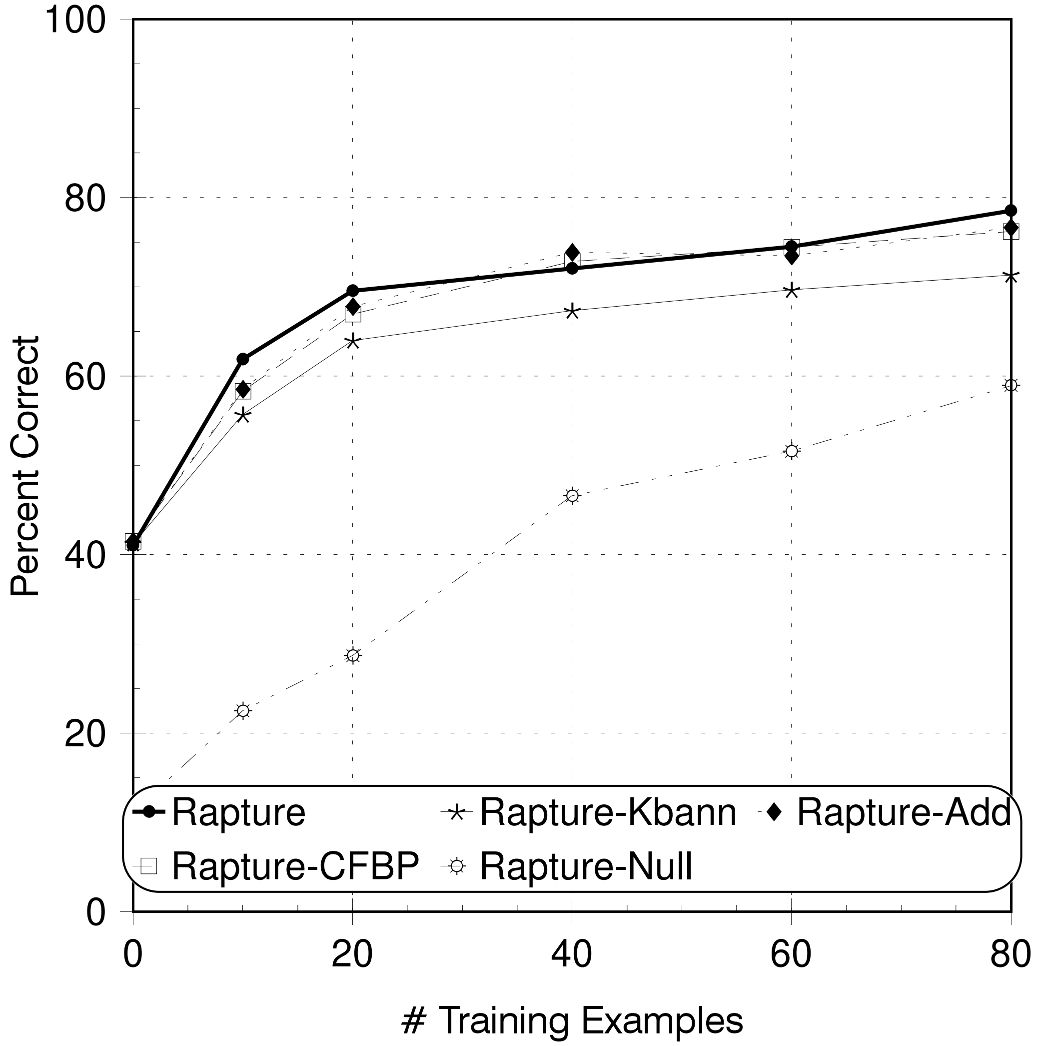

Figure 4.23: Mycin Disease Diagnosis Accuracy – Rapture Ablations

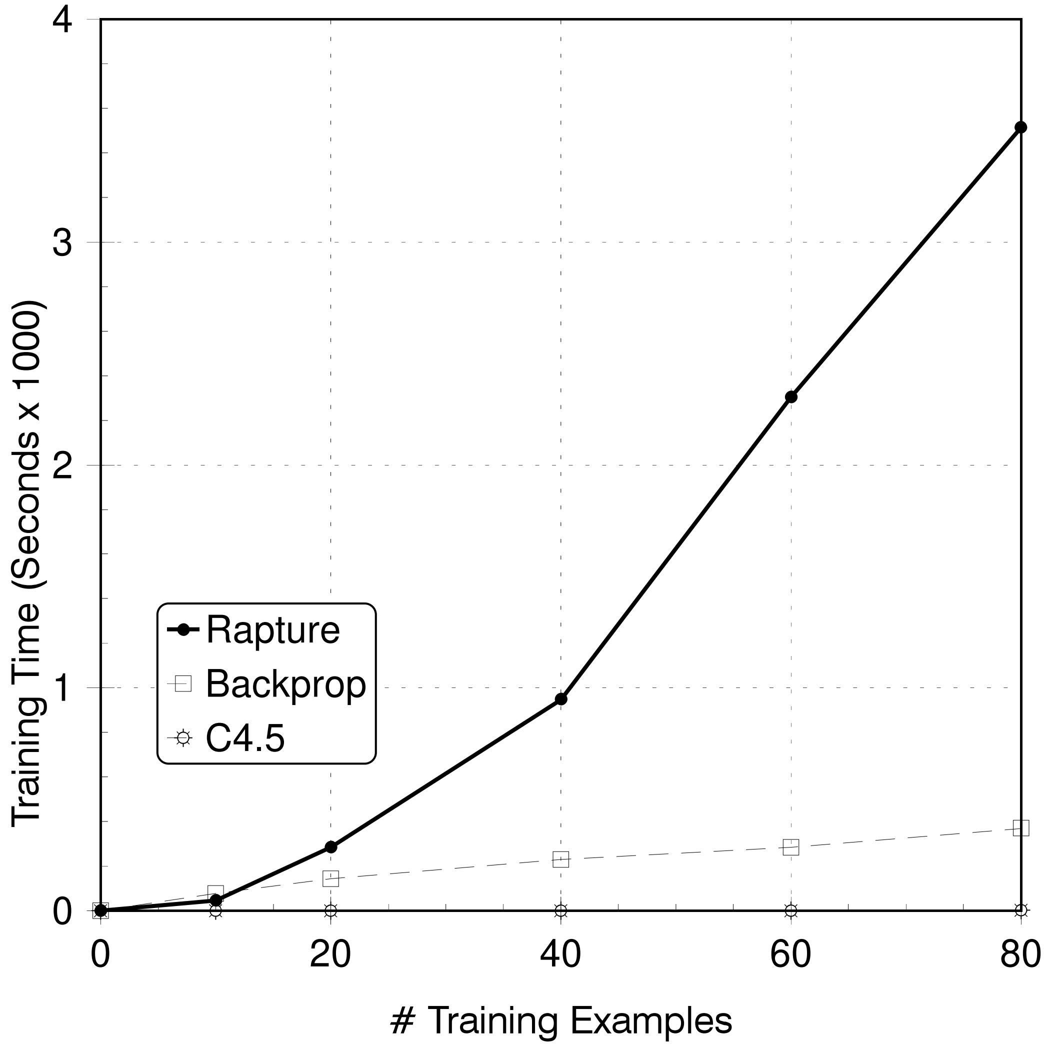

Figure 4.24: Mycin Training Time – Standard Systems

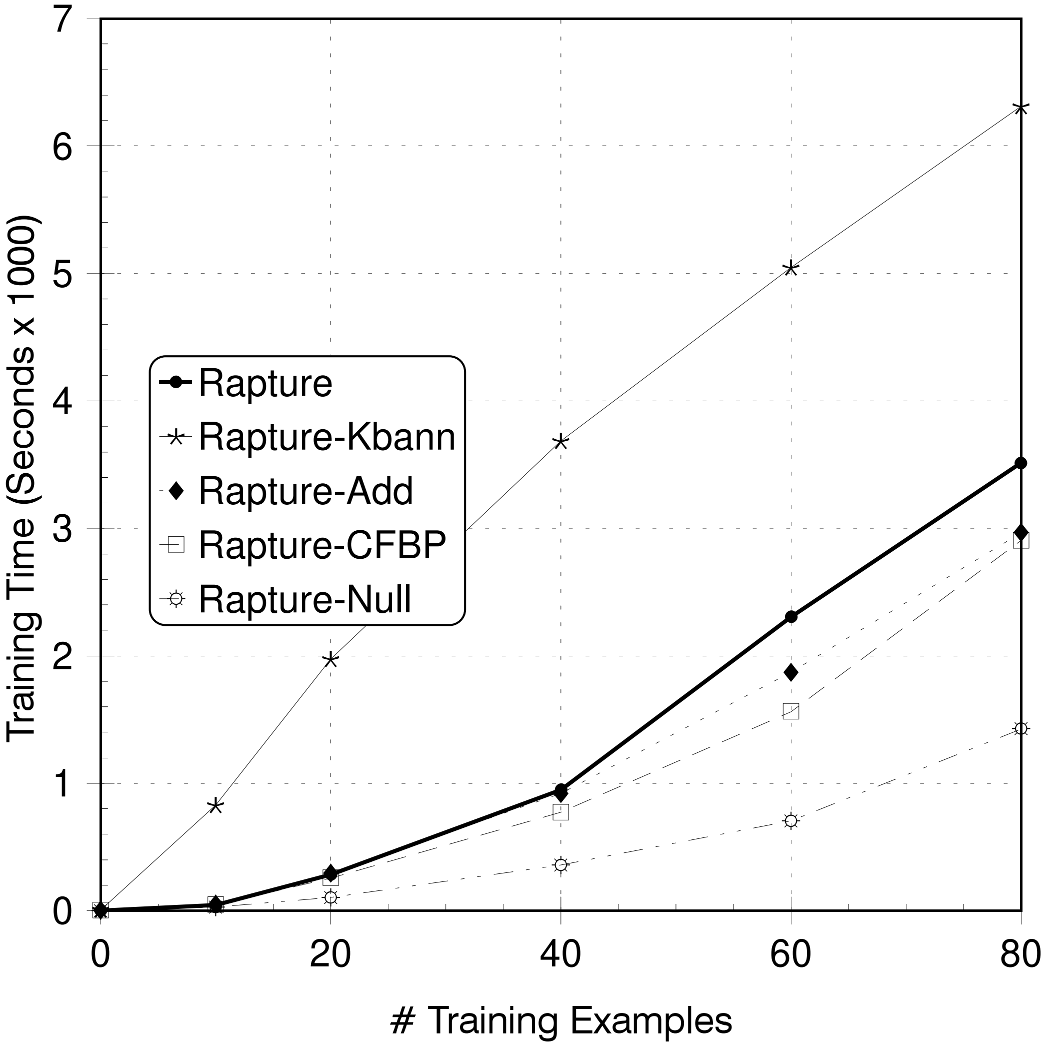

Figure 4.25: Mycin Training Time – Rapture Ablations

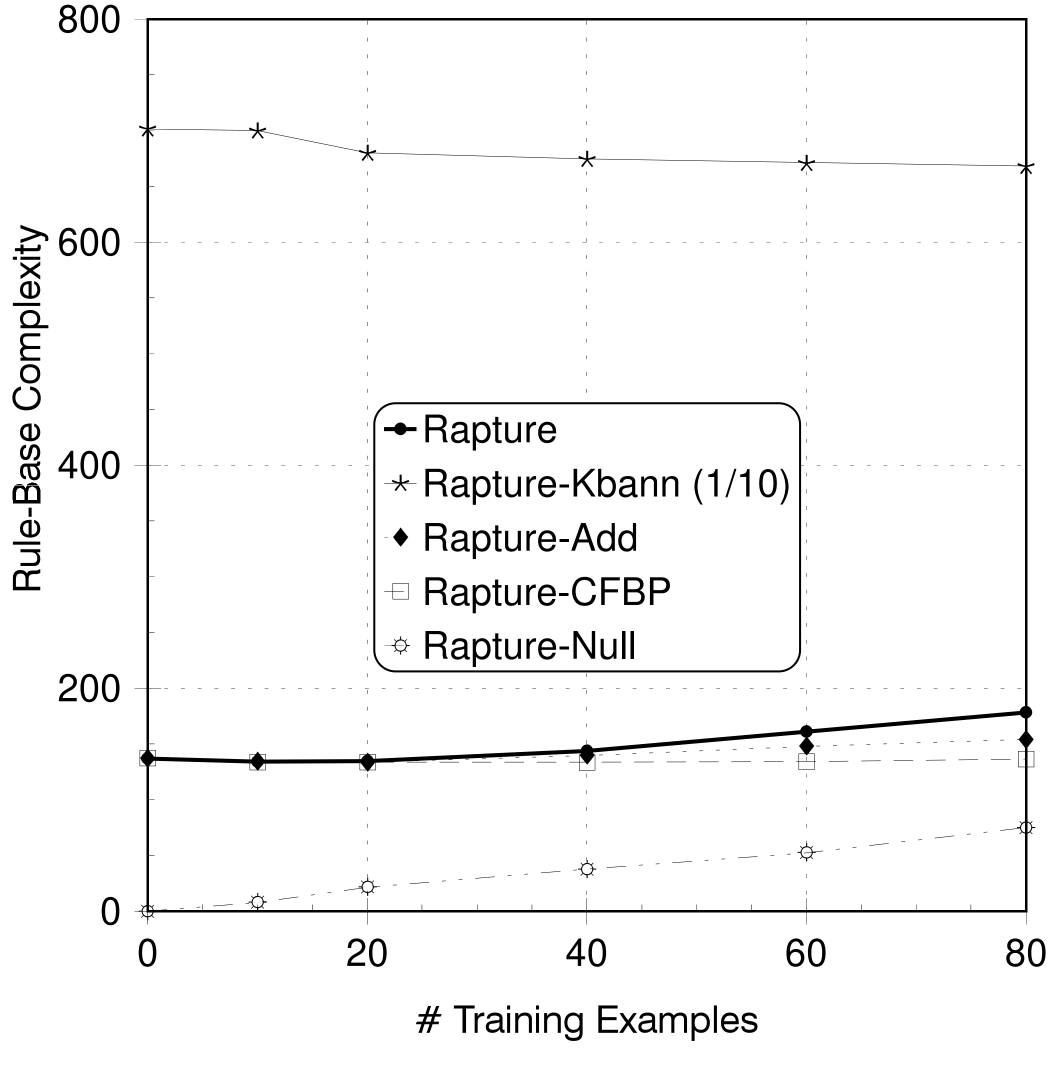

Figure 4.26: Mycin Concept Complexity

4.8 Soybean Diagnosis Results

The final data set tested in these experiments is one for diagnosing diseased soybean plants. This data comes from Michalski and Chilausky (1980), and was prepared in collaboration with experts in soybean pathology. The data set includes 562 examples of diseased soybean plants, and an initial rule base describing 15 soybean diseases.

While the original rule base for this domain was written as a set of Horn clauses, it contains characteristics similar to a probabilistic rule base. This is due to the different types of rules that are used to describe each disease. The expert pathologist suggested the use of rules with different strengths in order to properly identify each soybean disease. *Significant* rules were used to define the main characteristics of each disease, while *Confirmatory* rules were used to supply additional information that would confirm a diagnosis. While each rule is written as a Horn-clause, the authors suggest the use of a weighting scheme whereby significant rules provide 90% of the evidence, and confirmatory rules provide the remaining 10%. These values were used for the certainty-factor version of the rule base given to RAPTURE. The certainty-factor version contains 187 certainty-factor rules, and these rules are presented in Appendix B, along with the original set of rules.

Each soybean example is described with a vector of 35 features including the condition of the stem, roots, seeds, as well as climate information, time of year, and features of the soil. Each soybean has been classified by an expert into one of the 15 soybean diseases. In this data set there exists one pair of examples with identical feature vectors, yet differing classifications. This is handled as described in Chapter 3. There are also many examples with missing information. This data set has been used as a benchmark for a number of learning systems. The original certainty-factor rules for this domain, along with a revised rule base, rare presented in Appendix B.

Figure 4.27 is a plot of the learning curves for standard learning systems on this data. This plot has the unique characteristic that systems given no domain knowledge perform as well as RAPTURE. From 80 examples onward, C4.5 performs at a level for which there is no statistically significant difference from the performance of RAPTURE, and from 150 examples on, the same can be said for BACKPROP. RAPTURE does, however, significantly outperform the other systems ($\rho = 0.05$ level) when there are fewer than 80 training examples. This is an important characteristic, since for many domains, collecting large numbers of examples may be impossible.

Likely explanations for these performances come from the data. One characteristic of these 15 diseases is that they are for the most part, relatively easy to learn. Many of the diseases can be diagnosed by examining one or two key features from the plant. In these cases, the domain knowledge helps, but the inductive learners can catch up relatively quickly.

Secondly, three of the diseases are very closely related. These are Brown Spot, Alternaria Leaf Spot, and Frog Eye Leaf Spot. They are so closely related that, as mentioned previously, two identical examples were classified differently by different experts. One was classified as a Frog Eye Leaf Spot while the other was classified as a Alternaria Leaf Spot. This may perhaps be an indication that an insufficient set of features was used in describing these soybean plants. The domain knowledge seems to be of minimal help, and only with the easier to classify diseases. Because of this, RAPTURE is forced to make extensive use of architecture modification routines. This is reflected in the time necessary to train, as well as the complexity of the resulting rule base.

Unfortunately, the three difficult to distinguish *Spot* diseases dominate the examples in this data set. Out of 562 training examples, nearly half of them (274) belong to one of these three

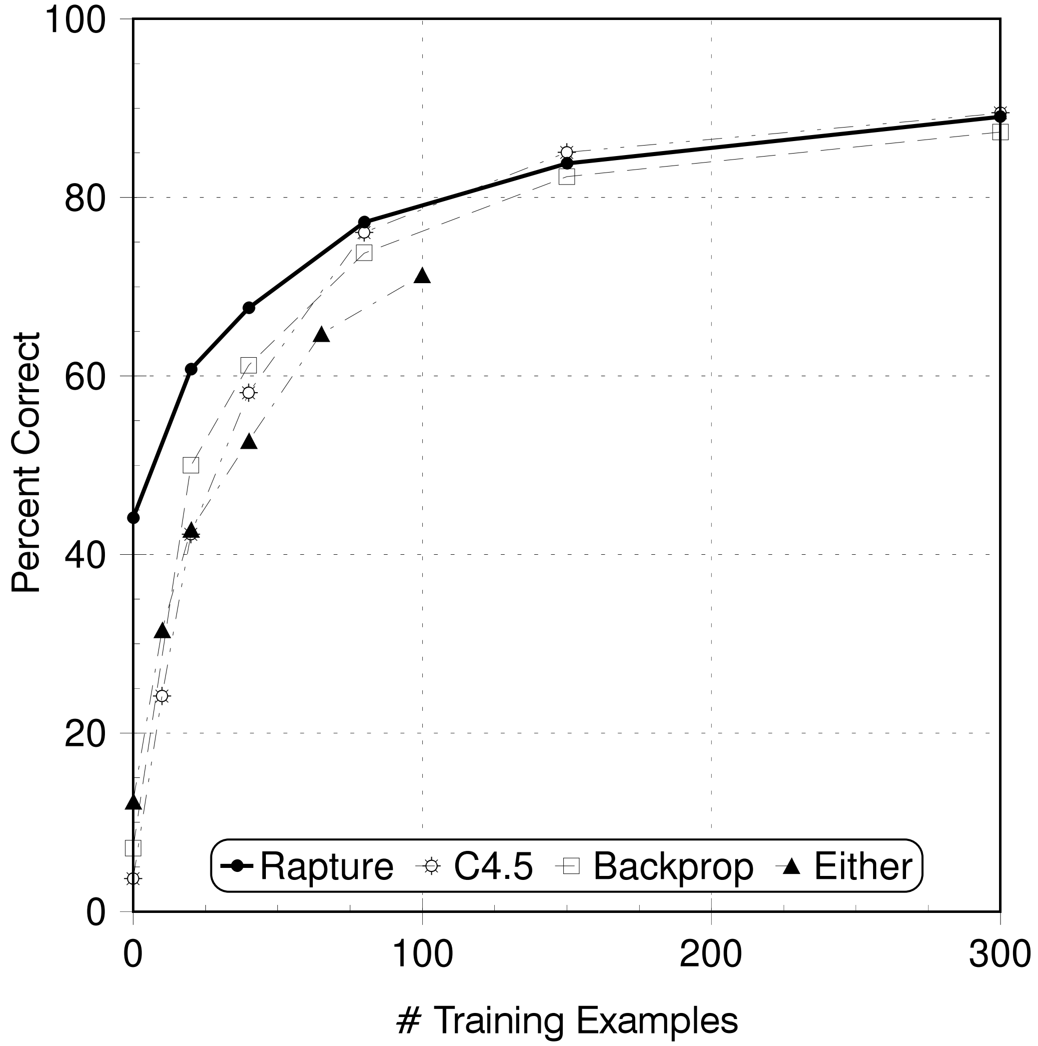

Figure 4.27: Soybean Diagnosis Accuracy – Standard Systems

categories. Because of this, systems with no domain knowledge are able to learn the easier diseases rather quickly, and the domain knowledge is of little help with the more difficult diseases. Thus all of the systems have equal difficulty with the these diseases.

EITHER to date has only been run up to 100 examples. It has, however, also been run using a partial-matching technique, where examples are classified into the category that was closest to firing, in which case its performance through 100 examples nearly matches that of RAPTURE (Ourston & Mooney, 1994).

Figure 4.28 effectively demonstrates the contributions of RAPTURE's component algorithms. From 80 examples onward, RAPTURE is performing at significantly better levels than the variants. This can only be attributed to the UPSTART algorithm. As the initial set of rules lacks essential information for distinguishing the *Spot* diseases, UPSTART is able to create useful *hidden* concepts. Both RAPTURE–ADD and RAPTURE–NULL come close to the levels of RAPTURE, while RAPTURE–CFBP and RAPTURE–KBANN are performing much worse. The poor perfor-

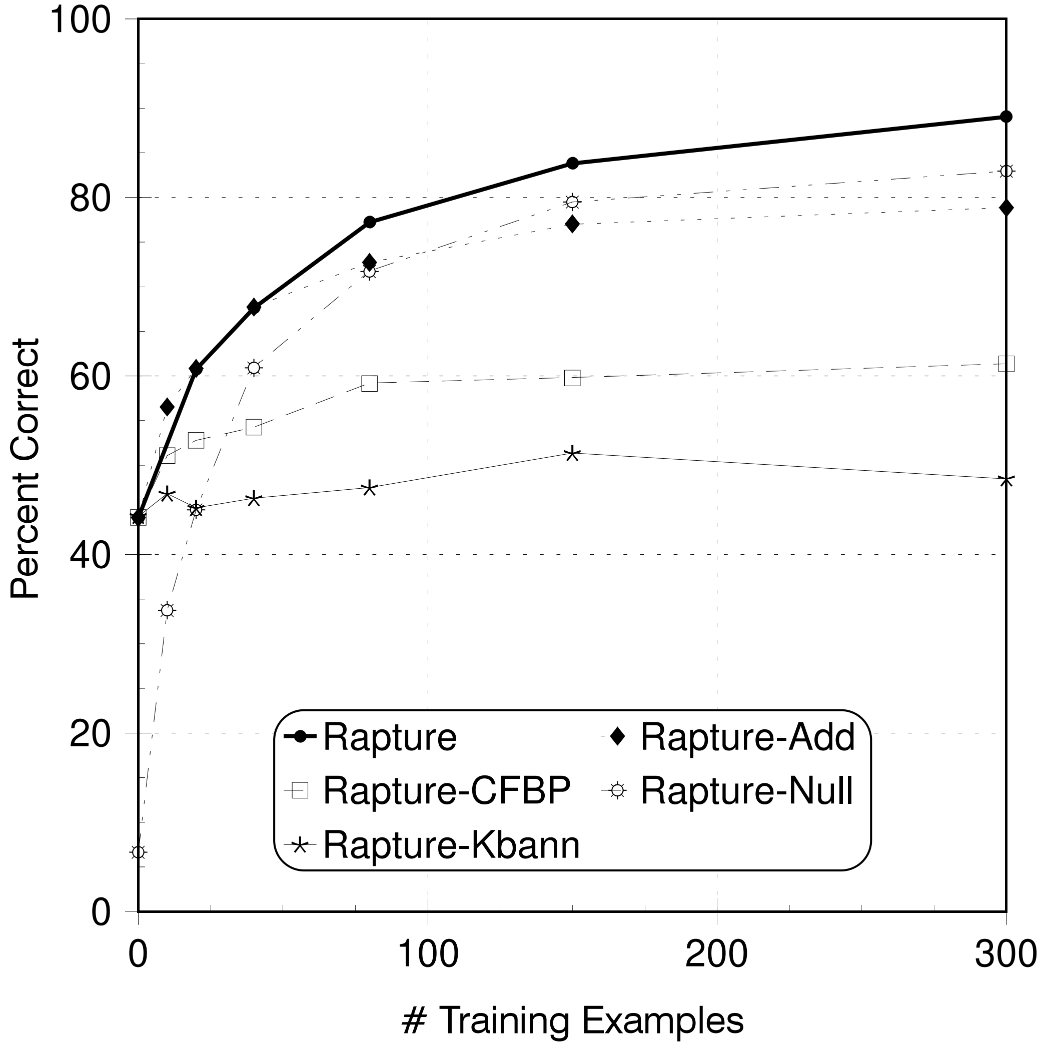

Figure 4.28: Soybean Diagnosis Accuracy – Rapture Ablations

mance of RAPTURE–CFBP is a clear indication that the original rules are lacking appropriate features to discriminate among many of the diseases. By allowing RAPTURE–CFBP to add new rules (RAPTURE–ADD), significant increases in performance are obtained. The performance of RAPTURE–KBANN is due to its inability to converge on the training data. This is very similar to the difficulties encountered by CF-NET which is unable to learn the training data for any of the data sets presented here, RAPTURE–KBANN is equally hindered by the local-minimum characteristic of gradient descent over certainty-factors. This is probably due to the less than perfect initial rule base, and the large number of examples with hard to discriminate diseases.

Examining the training time plots of Figure 4.29 and Figure 4.30, further effects of the algorithm can be seen. Due to the use of the UPSTART algorithm, RAPTURE's training time becomes quite slow. Only RAPTURE–NULL which makes even more extensive use of the UPSTART algorithm takes longer. RAPTURE–ADD and RAPTURE–CFBP are able to run in much less time, due to their simpler algorithms, yet perform at significantly poorer level.

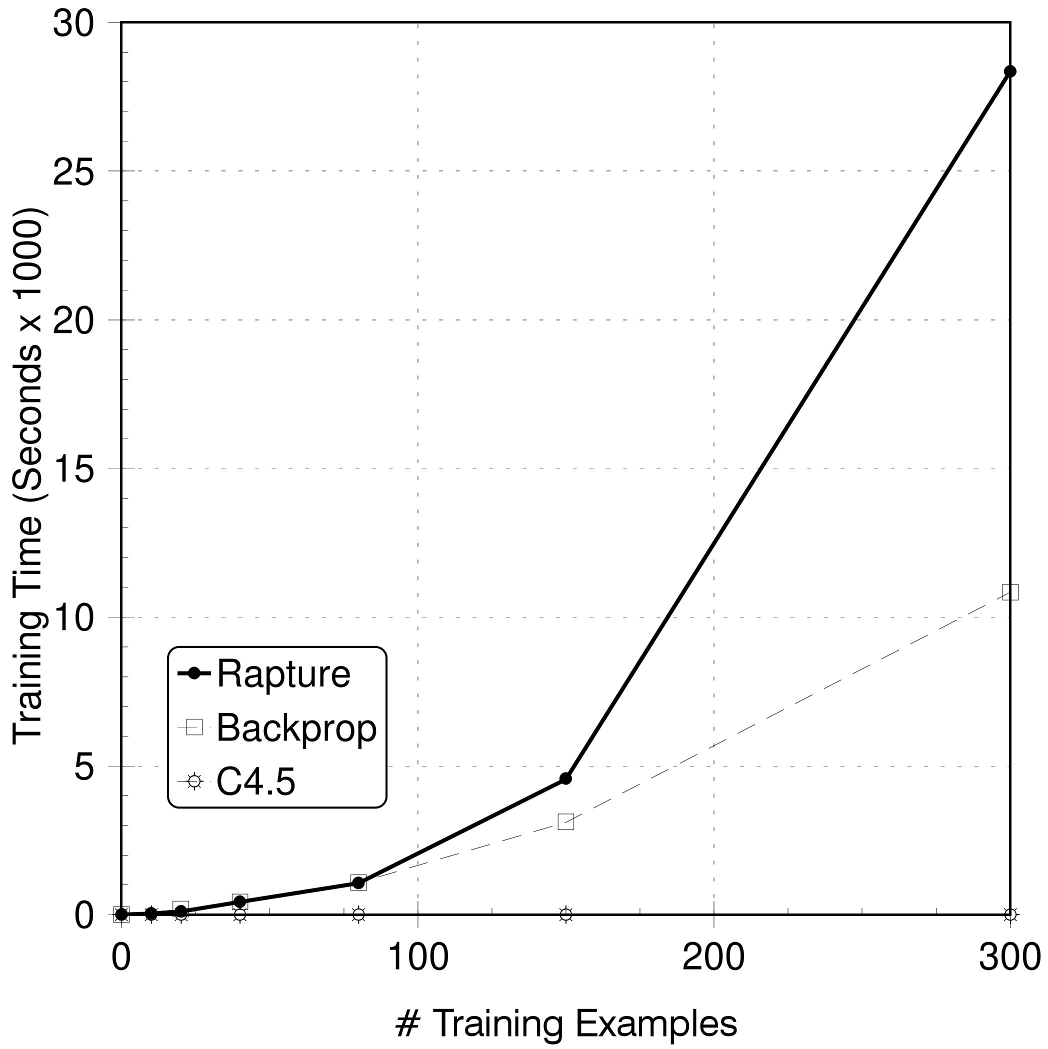

Figure 4.29: Soybean Training Time – Standard Systems

By examining the rule base complexities of Figure 4.31, the great complexity of RAPTURE–KBANN can be observed. There is also very little link deletion, which is further evidence of its inability to converge on the training data. RAPTURE–NULL again produces more complex rule bases than RAPTURE, another indication of the complexity of the problem.

4.8.1 Examination of a Revised Rule Base

Section 4 of Appendix B displays the original certainty-factor rules for this domain, followed by a rule base that was revised by training on 100 randomly selected examples. Contrasting the two rule bases reveals much information about the revision algorithm, as well as the quality of the original rules.

One of the first observations is that for many disease categories, very little change was made to the rules. A primary example is the rules for `Bacterial-Blight`. No rules are added or deleted, and the biggest change to a certainty factor is a change of 0.05. Similar to this are the rules for

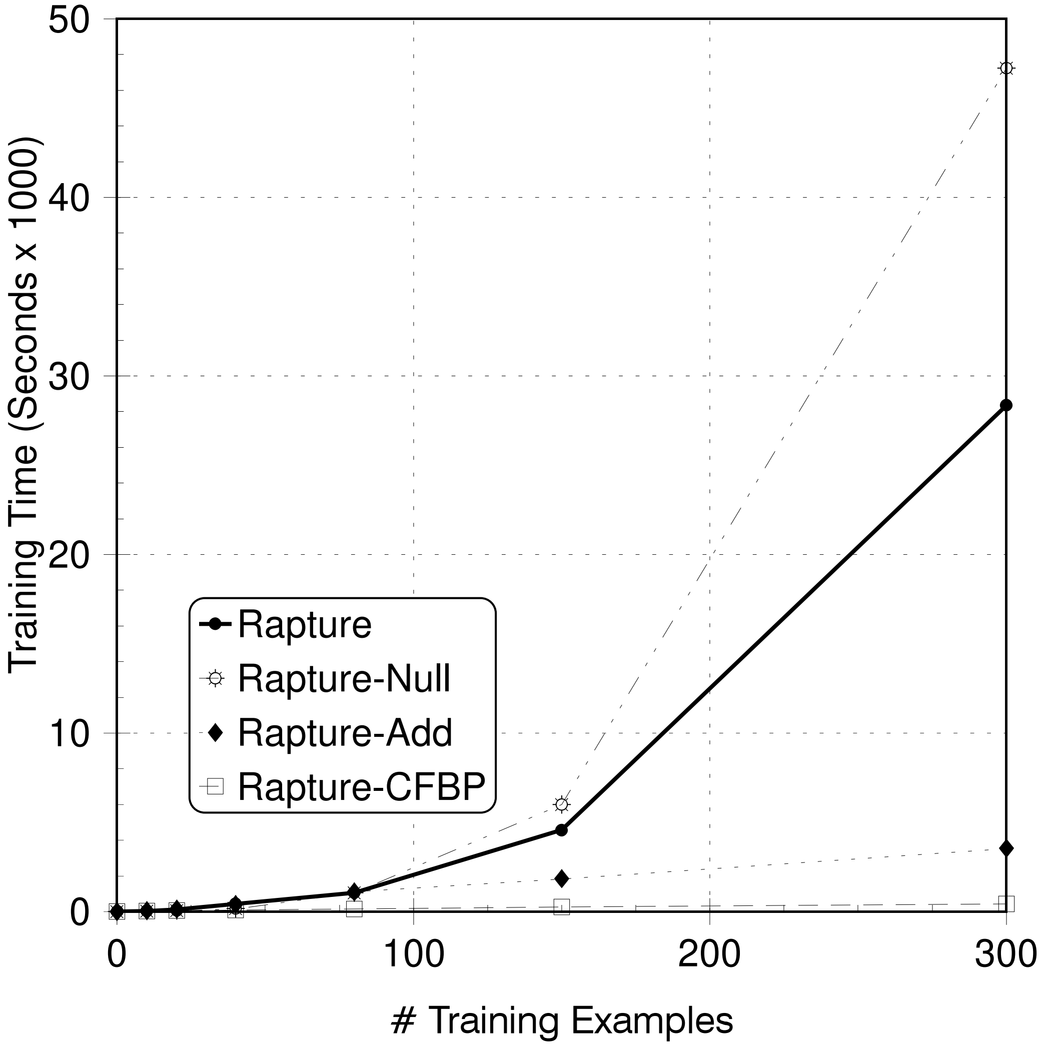

Figure 4.30: Soybean Training Time – Rapture Ablations

`Diaporthe-Stem-Canker`. While many of the rules change very little, one of the confirmatory conditions (crop history) is deleted, and a new condition (fruit spots) is added.

This is in contrast to the rules for `Brown-Spot`. As mentioned previously, the spot diseases are apparently very similar, and are difficult to distinguish via the given rules. Whereas the original rules only contain 6 conditions, the revised rules contain 24 conditions, and one of these is a new intermediate term which itself is made up of 12 conditions. There is very little in common among the two sets of rules. The original confirmatory condition for precipitation becomes a very strong negative condition in the revised rules. The new intermediate condition was built while training a new node on 5 false-negative examples for `Brown-Spot`. This new intermediate condition seems to be emphasizing examples in the sixth month (date 6), and a very strong condition is added for condition `area-damaged`.

Another leaf spot disease that caused the creation of a new intermediate term is `Phyllosticta-Leaf-Spot`. This set of rules went from 9 rules to 18 rules, including an intermediate term with 9 rules. This

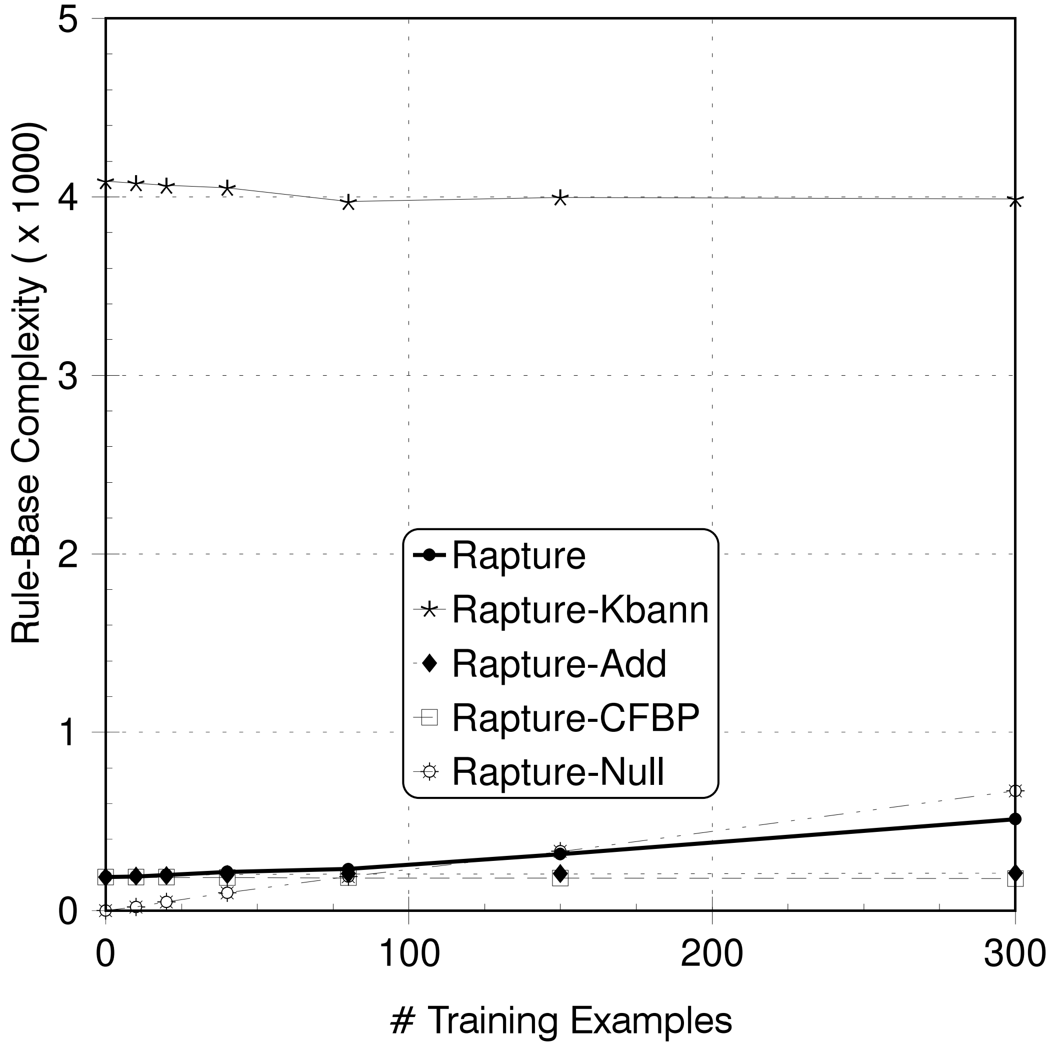

Figure 4.31: Soybean Rule-Base Complexity

new intermediate term seems to be focussed on low precipitation. It would be indeed interesting to see if a soybean expert would be able to identify these new conditions.

It appears that one of RAPTURE's keys to success is its ability to stay very close to the original theory whenever possible. This is apparent by the large numbers of rules that remain constant. RAPTURE is able, however, to make rather large changes to a few key features, and this allows examples to be classified correctly.

4.9 Summary of the Results

One of the overriding conclusions from the experimental data presented here is the benefit gained through the use of domain knowledge. Except for the final domain, systems given initial problem-solving knowledge performed uniformly better than those that were not given this knowledge. Even in the Soybean data, this holds true for smaller numbers of training examples. The ablation

RAPTURE–NULL clearly demonstrates this point. This system begins with no background knowledge, and has uniformly inferior performance. With larger numbers of training examples, it also takes much longer to train, and creates very complex rule bases.

Similar claims can be made for systems allowing evidence summation. While this is perhaps an obvious claim when dealing with DNA data, which is widely believed to have the M-OF-N property, results from the diagnosis domains show similar success. The systems C4.5 and EITHER are both unable to easily represent such concepts, and their performances suffer. While C4.5 is very fast, and produces relatively less complex rule bases, it has significantly inferior performance.

The contributions of the three RAPTURE modules are demonstrated in the ablation graphs. In the promoter experiments, where CFBP alone can train (most of)the data, there are very few differences. In the splice and soybean experiments, RAPTURE performs significantly better than CFBP, and the differences are dramatic in the soybean experiments. This is due to the inability of the initial set of soybean rules to effectively discriminate between certain diseases. This is also the one domain where RAPTURE outperforms RAPTURE–ADD.

One of the biggest surprises from the results is the performance of CF-NET. Because of the randomness of the initial network, along with the abundance of local minima in the space of certainty-factor weights, this system was generally unable to learn. The weight-adjustment of CFBP generally reaches such a minimum point after many fewer epochs than BACKPROP. This combined with the random weight settings caused CF-NET to be quite unsuccessful.

RAPTURE–KBANN performs at a slightly less accurate level than RAPTURE, but suffers from extremely large rule bases, and slow training time. This system has the same difficulty as CF-NET above, as is unable to train on the Soybean data due to the large size of the rule base.

4.9.1 Concept Drift

One of the claims of this research is that the revised rule bases will be easier to understand than the translated output of other learning systems. Since the learning remains in a certainty-factor framework, and the resulting sets of rules are reasonably small, it is hypothesized that an expert will have little difficulty in understanding the new rules.

Unfortunately, it is not known whether or not the CFBP stage of the algorithm will cause some of the concepts to "drift" in meaning. When certainty-factor values change dramatically, it is possible that intermediate terms will have new meaning in the revised rule bases. Fortunately, as can be witnessed in the revised rule bases, RAPTURE leaves many certainty factors virtually untouched, while changing only a few in a more dramatic fashion. Presenting revised rule bases to an expert will provide some answers to these questions.

Chapter 5

Related Work

This chapter reviews some of the related work in rule base revision, from both the connectionist and symbolic areas. Compared with much of the previous work, RAPTURE deals with a greater range of refinement problems and has been more thoroughly tested on actual expert knowledge bases.

5.1 Symbolic and Weight Adjustment Systems

Perhaps the earliest success with automatically performing weight adjustment in an Artificial Intelligence system comes from Samuel (1959). This was the landmark checker-playing program. Though this system did not actually revise rules in a rule base, it did automatically tune the coefficients in its polynomial evaluation functions to improve learning, with great success.

An early attempt at properly setting weights in a certainty-factor rule base was Rada (1985). This system attempted to combine credit assignment with weight adjustment in order to improve performance. Unfortunately, much of the weight adjustment was ad hoc, and their were no mechanisms for architecture modification.

Another early attempt at automating rule-base revision is the SEEK2 system of (Ginsberg, 1988). This system is able to specialize and generalize *choice-component* rules, which are simply M-OF-N rules. Guided by heuristics that estimate the net gain of various rule modifications, this system could adjust the threshold, N, of individual rules. Experiments show that this leads to a performance increase on both training and test data results. Training with SEEK2, however, does not guarantee 100% accuracy with the training data, and has no mechanisms for adding new rules, intermediate concepts, or new conjuncts. Unlike RAPTURE, SEEK2 is unable to allow the individual antecedents of a rule to carry differing amounts of weight.

Ma and Wilkins (1991) have developed methods for improving the accuracy of a certainty-factor rule base by deleting rules. They report only modest improvements in the accuracy of the same MYCIN rule base (used in section 4.7). Their experiments increase accuracy from 26.8% to 36.0%. RAPTURE has the advantage of being able to adjust certainty factors, add rules, and add new intermediate terms in addition to deleting rules.

The PTR system by Koppel, Feldman, and Segre (1994a) takes an initial rule base expressed as a collection of Horn clause rules, along with an expert's confidence values in the accuracy of each of these rules. Unlike in RAPTURE, these values do not represent the strength, or amount of evidence suggested by each of these rules, but rather one's confidence that the rule is correct. By using these values, along with a set of training examples, PTR is able to incrementally reformulate the rule base in such a way that is consistent with the training data, as well as maximizing one's confidence in the rules. They further demonstrate the significance of the rule base for identifying promoters. By simply noting how closely each example is satisfied by the rules, and categorizing the top 50% as promoters, results similar to those of RAPTURE are obtained (Koppel, Segre, & Feldman, 1994b). It is unclear how well this technique would do on further domains.

A similar result has been obtained by Ortega (1995). By modifying the promoter rules into a set of M-OF-N rules, accuracy of 93.4% is achieved.

EITHER (Ourston, 1991) was one of the inspirations for RAPTURE. This is a completely symbolic system that modifies initial rule-bases by adding and deleting antecedents and rules. By attempting to make as few changes to the initial rules as possible, while correctly processing all training examples, much of the original expertise remains intact, resulting in good performance. Unfortunately, the system is not capable of effectively representing uncertain rule-bases.

Valtorta and Ling (Valtorta, 1988, 1990; Ling & Valtorta, 1991) have examined the computational complexity of various refinement tasks for uncertain knowledge bases. They have considered networks using various combination functions, such as `MIN`, `MAX`, and probabilistic sum. For most combination functions and network architectures, they show that refining the weights to fit a set of training data is an NP-Hard problem. However, they present a simple linear-time algorithm for determining a correct setting of weights (or proving none exist) in the special case of a one-layer network with MAX as the combining function. They further make assumptions that each training example is paired with its correct certainty factor. However, even with these assumptions, they provide no mechanisms for adding new features or otherwise altering the network architecture. Their NP-completeness results motivated the use of a hill-climbing technique, such as gradient descent, in RAPTURE.

The work of Caruana (1989) uses truth maintenance, along with gradient descent in order to train rule bases that incorporate uncertainty. This system, however, has only been tested in toy domains, and no significant results are presented, though a claim is made that the truth maintenance played heavily in the resulting revisions.

5.2 Connectionist Systems

Gallant (1988) was among the first to design and implement a system that combines expert domain-knowledge with connectionist learning. Given a set of training examples and expert-supplied dependency information, his system builds a connectionist network that correctly classifies the training data. However, the training method is a variation of perceptron learning and is not suitable for multi-layer networks or for alternative combination functions like probabilistic sum.

Fu (1989) and Lacher, Hruska, and Kuncicky (1992) have also used backpropagation techniques to revise certainty factors on rules. Fu has apparently derived formulas for CFBP, although they are not given in the paper. Unlike RAPTURE, Fu's method does not implement complete CFBP, but rather uses it only on every other layer of the network, and uses a different hill-climbing method on the alternate layers. The claim is that this approach is required since the MIN and MAX functions are not differentiable. In RAPTURE, this does not cause a major problem since although these functions are not *everywhere* differentiable, they are trivially so *almost* everywhere. When working with real-valued weights, the problem of having two non-zero activation levels that are exactly the same, and both being the minimum value into another node is a rare occurrence. Lacher apparently concurs with this assessment, and has independently implemented a complete version of CFBP. However, recent publications on these two projects do not address the problem of altering the network architecture (i.e. adding new rules) and do not present results on revising actual expert knowledge bases. Fu does, however, present data on a rule base which is initially 100% accurate, and then corrupted by adding contradictory rules.

There have been a number of methods for building a connectionist networks from scratch that are able to correctly classify a set of training examples, e.g. CASCADE CORRELATION (Fahlman & Lebiere, 1989), the UPSTART algorithm (Frean, 1990), and the TILING algorithm (Mezard & Nadal, 1989). RAPTURE makes use of one of these (UPSTART) in creating new hidden units in existing networks, and also uses methods from decision-tree induction (Quinlan, 1986) to add observable units. Currently, however, RAPTURE is limited to adding new units that directly feed into the output layer, and there exists no way to add a new link into an existing non-output unit, other than a newly created UPSTART node. This inability does not appear to create major difficulties for RAPTURE, based on the empirical results.

The most closely related work to that described here has been done in conjunction with KBANN (Towell et al., 1990; Towell & Shavlik, 1992). This system uses standard backpropagation to refine a symbolic rule base. A propositional Horn-clause rule base is mapped into a standard neural network, the network is refined using normal backpropagation, and the result is mapped back into rules with real-valued antecedent weights. Unlike RAPTURE, the mapping between the symbolic rules and the network is only an approximation. Also, it is unclear how certainty-factor rules might be mapped into a KBANN network. KBANN allows the learning of new rules by including an underlying fully-connected network of low-weighted links. These links can be "recruited" by backpropagation and eventually mapped back into new rules. Weight decay (Hinton, 1986) is used to keep weights small and therefore help minimize the number of new rules that are eventually introduced. By contrast, RAPTURE uses symbolic methods to add a minimal number of new connections (rules) as needed. The results on promoter recognition in Section 4.5, and splice-junction recognition in Section 4.6 indicate that the RAPTURE approach produces a slightly more accurate revised knowledge base than KBANN, and a significantly less complex rule base than RAPTURE–KBANN. The original KBANN work contained a suggestion for adding new hidden units into a KBANN-net that would seem to work only in very specialized domains—such as promoter recognition. This was achieved through adding "cone" units that would connect contiguous features.

KBANN-DAID (Towell & Shavlik, 1992) is an attempt to improve the way that backpropagation adjusts network weights. In a standard neural network with many hidden layers, backpropagation tends to modify weights equally throughout the layers. By placing more emphasis on weight adjustment at the lowest layers, better performances can result.

TOPGEN (Opitz & Shavlik, 1993) is a method for adding new hidden units to a KBANN-network. By keeping track of of the false negative and false positives for which each node is responsible, the algorithm locates areas of the network requiring additional units. However, when TOPGEN adds a new hidden unit, it adopts the KBANN approach of fully connecting it to the input layer. By contrast, RAPTURE uses information gain to add new input features only as needed.

5.3 Bayesian Approaches

Schwalb (1993) has shown how the parameters of Bayesian networks can also be refined by mapping them into neural networks with SIGMA-PI nodes, and performing backpropagation. However, his method creates a neural network whose size is exponential in the fan-in of the Bayesian network. Because of this, it is unable to run on rule bases of even modest size such as promoter. This work also does not address the issue of adding new features or hidden units, and was not tested on revising actual knowledge bases.

Cooper and Herskovits (1992) have proposed a Bayesian method for the inductive learning of Bayesian networks. By hill-climbing through the space of Bayesian networks, and determining the probability of each network given the training data, a network with high conditional probability can be produced. This approach unfortunately requires extensive search.

The work of Musick (1994) attempts to inductively learn the parameters of a Bayesian network using statistical techniques. This approach assumes that the structure of the network is provided, and creates parameters that are actually distributions of values.

Finally, the work of Ramachandran (1995) was an outgrowth of RAPTURE. Her BANNER system attempts to refine rule bases specified in a Bayesian-network framework. Once a Bayesian rule base has been obtained, it is converted into an equivalent neural network. This network is then modified with standard backpropagation. In order to alleviate the exponential number of links that are required, noisy `AND` and `OR` nodes are created. Though this system has yet to implement and test architecture modification techniques, initial results suggest that this approach is very promising.

Chapter 6

Future Work

The development of RAPTURE has raised many questions, and left many problems for future research. One of the first tasks that should be addressed is running on more domains. In the experiments to date, RAPTURE has performed as well or better than other learning systems. Perhaps this is due to the fact that the tested domains all suggested the use of uncertain reasoning, which is one of RAPTURE's strongest points. Further, most of the tested domains had little use of new intermediate terms. Because of this, the UPSTART algorithm often failed to improve generalization over RAPTURE–ADD. It would indeed be interesting to perform tests in domains where intermediate terms were known to be required.

One of the claims of this work is that RAPTURE is able to produce less complex and hence hopefully more humanly-understandable sets of rules. It would be extremely beneficial to work closely with a domain expert, and examine the changes that RAPTURE is making to a particular rule base. Such a person would be able to indicate the validity of any new rules, new intermediate terms, or modified weights. By pointing out any perceived errors, or suggesting alternative modifications, perhaps even better performance can be realized. An interactive version of RAPTURE that suggests a number of revisions at each stage to a domain expert, who then selects the most appropriate revision, could prove quite powerful.

Further study is also in order for new methods of architecture modification. Currently, RAPTURE can only add links that directly connect into output nodes. By working recursively, this has the effect of creating hidden units, but there is no mechanism for directly creating them. As mentioned previously, the UPSTART algorithm only enhanced the performance of one of the test domains (soybean), while greatly increasing training times. Perhaps the use of other connectionist techniques would prove significantly better. By examining those nodes in the network that appear to be creating network error, revisions can perhaps be better focused. TOPGEN (Opitz & Shavlik, 1993) offers an approach similar to this.

One aspect of RAPTURE that has gone almost completely untested is the setting of parameters. One of the undesirable characteristics that RAPTURE inherited from backpropagation is the usage of many parameters. These include the learning rate, momentum, weight-decay rate, etc. Throughout most of the testing of this system, parameters were set once at obvious levels, and not adjusted further. Momentum has yet to be tested during training (has been continually set at 0). The favorable results are certainly an indication of the robustness of the system, but perhaps even better performance could be achieved through a fine-tuning of all parameters. The work of

Kohavi and John (1995) offers one approach to automatically tuning parameters through the use of cross-validation.

The problem of overfitting has only been partially addressed in the development of RAPTURE-. By attempting to limit the number of new links that are added directly into the output layer, some improvement in overfitting has resulted. Using UPSTART as early as possible in the architecture modication has produced better generalizations, but it is possible that this routine is causing some overfitting as well. Indications of this can be seen in the graph of Figure 4.11, where the performance of RAPTURE declines slightly after the first 50 training examples. More extensive overfitting-prevention techniques could result in better performance. This could perhaps be achieved with the use of tuning sets during training, or adapting the Minimum Description Length principle (Rissanen, 1978) to the RAPTURE system.

Also, RAPTURE is currently limited to propositional rule bases. It should be feasible to modify the system to handle first-order systems. I am currently unaware of the existence of first-order certainty-factor rule bases, and am not convinced that the Horn-clause to certainty-factor conversion technique described earlier will be effective for first-order rules. Once obtained, however, such a set of rules should be amenable to the techniques of RAPTURE. One problem may occur when trying to add new features. Currently, RAPTURE considers every possible combination of features and values as a new input feature. In first-order logic, this may require the consideration of every possible value for each first-order variable. Heuristic techniques will probably be required.

Although they have proven quite useful in practice, certainty factors have frequently been criticized as *ad hoc* and restrictive (Shafer & Pearl, 1990). Actually, certainty factors have been shown to have a clear probabilistic semantics, but only under very restrictive independence assumptions (Heckerman, 1986). Nevertheless, the basic revision framework in RAPTURE should be applicable to other uncertain reasoning formalisms such as Bayesian networks (Pearl, 1988), Dempster-Shafer theory (Shafer, 1976), or fuzzy logic (Zadeh, 1965). Although Schwalb's approach to revising Bayesian networks is intractable in the general case (Schwalb, 1993), it may be useful for networks with limited fan-in. The approach of Ramachandran (1995), utilizing noisy-or and noisy-and nodes seems to confirm that the Bayesian approach is worthy of further study. In addition, techniques for inducing Bayesian networks from data (Connolly, 1993; Cooper & Herskovits, 1992) could potentially be used to refine the underlying causal structure as well. Finally, there has also been some recent work on combining symbolic and neural-network methods to revise fuzzy-logic controllers (Berenji, 1990).

Chapter 7

Conclusions

Automatic refinement of uncertain rule-bases is an under-studied problem with important applications to the development of intelligent systems. This dissertation has described and evaluated an approach to refining certainty-factor rule bases that integrates connectionist and symbolic learning. The approach is implemented in a system called RAPTURE, which uses a revised backpropagation algorithm to modify certainty factors, ID3's information gain criteria to determine new rules to add, and the UPSTART algorithm to create new intermediate terms in the network. In other words, connectionist methods are used to adjust parameters and symbolic methods are used to make structural changes to the knowledge base.

In domains with limited training data or domains requiring meaningful explanations for conclusions, refining existing expert knowledge has clear advantages. Results on revising four real-world knowledge bases indicates that RAPTURE performs generally better than purely inductive systems (C4.5 and BACKPROP), a purely symbolic revision system (EITHER), and purely connectionist revision system (KBANN).

As mentioned in Section 2.3, certainty factors have received some criticism since the original MYCIN experiments. Much criticism was focused on the idea that that the performance of a certainty-factor system was relatively insensitive to the precision of the weights on the rules. The experiments discussed here do *not* support this notion. Since the original MYCIN experiments only involved adjusting the precision of the weight of a rule, and not on their relative ranking, the values of the weights were unnecessarily confined. One of the lessons learned from the experiments of RAPTURE is that it does not matter whether the certainty factor of the output category with the highest value for a given example is 0.5 or 0.99. It is only the relative value that determines classification. This was one reason why a threshold of 0.9 was kept constant throughout training in experiments involving single-category problems. This prevented a gradual drifting of certainty factors towards 0, and a threshold being set at 0.05. This seemed undesirable.

Further, upon examination of the revised rule bases, much of the characteristics of RAPTURE can be seen. Many of the original rules remain virtually intact throughout training. This is due to RAPTURE'S bias for making modifications only as dictated by training examples. While many rules remain constant, a few rules with key features are changed dramatically. This can be done by deleting the rule entirely, or giving it a very high or low certainty-factor value. This has the effect of changing many of the relative weights of the rules.

The certainty-factor networks used in RAPTURE blur the distinction between connectionist and symbolic representations. They can be viewed either as connectionist networks or symbolic rule bases. RAPTURE demonstrates the utility of applying connectionist learning methods to "symbolic" knowledge bases and employing symbolic methods to modify "connectionist" networks. Hopefully these results will encourage others to explore similar opportunities for cross-fertilization of ideas between connectionist and symbolic learning.

It was the hypothesis of this thesis that rule bases could be created that were more accurate, more understandable, less complex, and in less time with the use of RAPTURE than with other systems. Experimental results have demonstrated some of the trade-offs. C4.5 appears to create uniformly the fastest and smallest rule bases, but suffers from poor generalization accuracy. RAPTURE appears to be the optimal compromise as it consistently produces the best performance, with reasonably small rule bases, and in reasonable time. Its major advantages are its ability to incorporate expert background-knowledge into its learning, and to keep all of the rules in a symbolic form. This creates rules that should generally be easier to comprehend.

Appendix A

Derivation of the CFBP Formulae

Using the designed RAPTURE network, we wish to perform backpropagation on the certainty factors (which are the weights on the links between the nodes). Following the presentation given in (Rumelhart et al., 1986), we can define the network error due to input pattern p as

$$E_p = \frac{1}{2}\sum_j (t_{pj} - o_{pj})^2 \tag{A.1}$$

and total error is similarly measured as $E = \sum E_p$. For any p-sum unit j, the net input to this unit for input pattern p is defined as the certainty factor of all incoming activation values.

$$net_{pj} = \mathbf{CF}(w_{ji}o_{pi})_{\forall i} \tag{A.2}$$

Certainty factor (**CF**) is defined in (Shortliffe & Buchanan, 1975) as

$$\mathbf{CF} = \frac{\mathtt{MB} - \mathtt{MD}}{1 - \min(\mathtt{MB}, \mathtt{MD})} \tag{A.3}$$

where MB (Measure of Belief) is the positive or confirming evidence, and MD (Measure of Disbelief) is the negative or disconfirming evidence. For any p-sum unit j, these are defined as

$$\mathbf{MB_j} = \sum_{+i} w_{ji}o_{pi} \qquad \mathbf{MD_j} = \sum_{-i} w_{ji}o_{pi}$$

where the first term is summed over only belief links, and the second summed over disbelief links. Note that $\sum x_i$ is shorthand for $x_1 \oplus x_2 \oplus ... \oplus x_n$ (the probabilistic sum of the x_i, where $x \oplus y \equiv x + y - x * y$). Also $0 \leq \mathtt{MB} \leq 1$, $0 \leq \mathtt{MD} \leq 1$, and $-1 \leq \mathtt{CF} \leq 1$. Expanding equation A.2 gives

$$net_{pj} = \frac{\sum_{+i} w_{ji}o_{pi} - \sum_{-i} w_{ji}o_{pi}}{1 - \min(\mathtt{MB_j}, \mathtt{MD_j})} \tag{A.4}$$

Since in RAPTURE we define a units output as simply the certainty factor of its inputs, we get

$$o_{pj} = net_{pj} \tag{A.5}$$

In order to achieve gradient descent, we need to have

$$\Delta_p w_{ji} \propto -\frac{\partial E_p}{\partial w_{ji}} \tag{A.6}$$

The term on the right can be expressed as

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial net_{pj}} \frac{\partial net_{pj}}{\partial w_{ji}} \tag{A.7}$$

Looking at the rightmost term, and applying equation A.2 gives us

$$\frac{\partial net_{pj}}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \mathbf{CF}(w_{jk} o_{pk}) \forall k = \frac{\partial}{\partial w_{ji}} \frac{\sum_{+k} w_{jk} o_{pk} + \sum_{-k} w_{jk} o_{pk}}{1 - \min(\mathtt{MB_j}, \mathtt{MD_j})} \tag{A.8}$$

Further examination shows that $\frac{\partial net_{pj}}{\partial w_{ji}}$ breaks down into four cases depending on whether w_{ji} is contributing positive or negative evidence, and on which of (MB, MD) has the smaller value. As the two former cases are symmetrical, the following assumes w_{ji} is contributing positive evidence, leaving the following cases for $\frac{\partial net_{pj}}{\partial w_{ji}}$.

1. **MB > MD**: This is the more common case in the domains studied to date. Noting that $\frac{\partial}{\partial x_k} \sum_i x_i = 1 - \sum_{i \neq k} x_i$ gives us

$$\frac{\partial net_{pj}}{\partial w_{ji}} = \frac{o_{pi}(1 - \sum_{+k \neq i} w_{jk} o_{pk})}{1 - \mathtt{MD_j}} \tag{A.9}$$

2. **MB < MD**: This gives us

$$\frac{\partial net_{pj}}{\partial w_{ji}} = \frac{o_{pi}(1 - \sum_{+k \neq i} w_{jk} o_{pk})(1 - \mathtt{MD_j})}{(1 - \mathtt{MB_j})^2} \tag{A.10}$$

In the event that MB=MD, the certainty factor becomes zero, and a zero derivative is assumed as well. For simplicity, the following assumes the more frequent MB > MD. By defining

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} \tag{A.11}$$

and combining with equations A.7 and A.9 we get

$$-\frac{\partial E_p}{\partial w_{ji}} = \frac{\delta_{pj} o_{pi}(1 - \sum_{+k \neq i} w_{jk} o_{pk})}{1 - \mathtt{MD_j}} \tag{A.12}$$

Therefore, in order to implement gradient descent in total error E, we need to make our weight adjustments according to

$$\Delta_p w_{ji} = \frac{\eta \delta_{pj} o_{pi}(1 - \sum_{+k \neq i} w_{jk} o_{pk})}{1 - \mathtt{MD_j}} \tag{A.13}$$

where η is the learning rate. All that now remains is to calculate δ_{pj} for each unit u_j in the network. This can be done by applying the chain rule to our original definition (A.11), giving us

$$\delta_{pj} \equiv -\frac{\partial E_p}{\partial net_{pj}} = -\frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial net_{pj}} \tag{A.14}$$

The rightmost factor is trivial. Since $o_{pj} = net_{pj}$ (A.5), this results in

$$\frac{\partial o_{pj}}{\partial net_{pj}} = \frac{\partial x}{\partial x} = 1. \tag{A.15}$$

In order to calculate the left-hand factor, we need to consider two separate cases. Beginning at the top of the network and working our way down, we first consider that u_j is an output unit. From our definition of E_p (A.1) we see that

$$\frac{\partial E_p}{\partial o_{pj}} = -(t_{pj} - o_{pj}), \tag{A.16}$$

and by combining this with equations A.14 and A.15 we get

$$\delta_{pj} = (t_{pj} - o_{pj}) \tag{A.17}$$

for any output unit u_j. If u_j is *not* an output unit, then we must remember that the output produced may first pass through a min node before its value can be forward propagated. Therefore, the *effective* output for any such node j is either 0, or the net_{pj} value previously calculated. In other words, a probabilistic-sum unit j's value only propagates into certain k's, which I label as k_{min}. Clearly, if j's value does not pass successfully through the min node, then its value is not a factor in the network's output. This can be shown formally by again applying the chain rule.

$$\frac{\partial E_p}{\partial o_{pj}} = \sum_{k_{min}} \frac{\partial E_p}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial o_{pj}} = \sum_{k_{min}} \frac{\partial E_p}{\partial net_{pk}} \frac{\partial}{\partial o_{pj}} CF(w_{ki} o_{pi})_{\forall i} =$$

$$-\sum_{k_{min}} \frac{\delta_{pk} w_{kj} (1 - \sum_{+i \neq j} w_{ki} o_{pi})}{1 - \mathtt{MD_k}} \tag{A.18}$$

Therefore, in order to modify weights using certainty factor backpropagation, we need to utilize the following three equations:

$$\Delta_p w_{ji} = \frac{\eta \delta_{pj} o_{pi} (1 - \sum_{+k \neq i} w_{jk} o_{pk})}{1 - \mathtt{MD_j}} \tag{A.19}$$

If u_j an output unit

$$\delta_{pj} = (t_{pj} - o_{pj}) \tag{A.20}$$

If u_j is not an output unit

$$\delta_{pj} = \sum_{k_{min}} \frac{\delta_{pk} w_{kj} (1 - \sum_{+i \neq j} w_{ki} o_{pi})}{1 - \mathtt{MD_k}} \tag{A.21}$$

While backpropagating, the only difficulty occurs while passing through `MIN` nodes, and determining which of the connected certainty-factor nodes actually passes a value through. This is easily determined, by checking each certainty-factor node's value, and comparing with value at the `MIN`. The only possible problem arises when two or more nodes each have the same value, and they have the same value as the `MIN` node. In this case the `MIN` function is not differentiable. This problem turns out to be negligible in practice, as it occurs rarely, and usually this involves two or more 0 values. In these cases, RAPTURE simply picks one of these nodes at random and continues backpropagating.

Appendix B

The Initial Rule-Bases in Original and Certainty-Factor Format

B.1 The Promoter Recognition Rule-Base

B.1.1 The Original Rules

```
(promoter ?x) <- (contact ?x)(conformation ?x)

(contact ?x) <- (minus_35 ?x)(minus_10 ?x)

(minus_35 ?x) <- (p-37 c)(p-36 t)(p-35 t)(p-34 g)(p-33 a)(p-32 c)
(minus_35 ?x) <- (p-36 t)(p-35 t)(p-34 g)(p-32 c)(p-31 a)
(minus_35 ?x) <- (p-36 t)(p-35 t)(p-34 g)(p-33 a)(p-32 c)(p-31 a)
(minus_35 ?x) <- (p-36 t)(p-35 t)(p-34 g)(p-33 a)(p-32 c)

(minus_10 ?x) <- (p-14 t)(p-13 a)(p-12 t)(p-11 a)(p-10 a)(p-9 t)
(minus_10 ?x) <- (p-13 t)(p-12 a)(p-10 a)(p-8 t)
(minus_10 ?x) <- (p-13 t)(p-12 a)(p-11 t)(p-10 a)(p-9 a)(p-8 t)
(minus_10 ?x) <- (p-12 t)(p-11 a)(p-7 t)

(conformation ?x) <- (p-47 c)(p-46 a)(p-45 a)(p-43 t)(p-42 t)
                     (p-40 a)(p-39 c)(p-22 g)(p-18 t)(p-16 c)(p-8 g)
                     (p-7 c)(p-6 g)(p-5 c)
                     (p-4 c)(p-2 c)(p-1 c)
(conformation ?x) <- (p-45 a)(p-44 a)(p-41 a)
(conformation ?x) <- (p-49 a)(p-44 t)(p-27 t)(p-22 a)(p-18 t)(p-16 t)
                     (p-15 g)(p-1 a)
(conformation ?x) <- (p-45 a)(p-41 a)(p-28 t)(p-27 t)(p-23 t)(p-21 a)
                     (p-20 a)(p-17 t) (p-15 t) (p-4 t)
```

B.1.2 The Certainty-Factor Rules

```
promoter <-0.68-- (contact)
promoter <-0.68-- (conformation)


contact  <-0.68-- (minus_35)
contact  <-0.68-- (minus_10)


minus_35 <-0.28-- (p-37 c)
minus_35 <-0.28-- (p-36 t)
minus_35 <-0.28-- (p-35 t)
minus_35 <-0.28-- (p-34 g)
minus_35 <-0.28-- (p-33 a)
minus_35 <-0.28-- (p-32 c)
minus_35 <-0.28-- (p-31 a)


minus_10 <-0.25-- (p-14 t)
minus_10 <-0.25-- (p-13 t)
minus_10 <-0.25-- (p-13 a)
minus_10 <-0.25-- (p-12 t)
minus_10 <-0.25-- (p-12 a)
minus_10 <-0.25-- (p-11 t)
minus_10 <-0.25-- (p-11 a)
minus_10 <-0.25-- (p-10 a)
minus_10 <-0.25-- (p-9 t)
minus_10 <-0.25-- (p-9 a)
minus_10 <-0.25-- (p-8 t)
minus_10 <-0.25-- (p-7 t)
```

```
conformation <-0.08-- (p-47 c)
conformation <-0.08-- (p-46 a)
conformation <-0.08-- (p-45 a)
conformation <-0.08-- (p-44 t)
conformation <-0.08-- (p-44 a)
conformation <-0.08-- (p-43 t)
conformation <-0.08-- (p-42 t)
conformation <-0.08-- (p-41 a)
conformation <-0.08-- (p-40 a)
conformation <-0.08-- (p-39 c)
conformation <-0.08-- (p-28 t)
conformation <-0.08-- (p-27 t)
conformation <-0.08-- (p-23 t)
conformation <-0.08-- (p-22 a)
conformation <-0.08-- (p-22 g)
conformation <-0.08-- (p-21 a)
conformation <-0.08-- (p-20 a)
conformation <-0.08-- (p-18 t)
conformation <-0.08-- (p-17 t)
conformation <-0.08-- (p-16 t)
conformation <-0.08-- (p-16 c)
conformation <-0.08-- (p-15 t)
conformation <-0.08-- (p-15 g)
conformation <-0.08-- (p-8 g)
conformation <-0.08-- (p-7 c)
conformation <-0.08-- (p-6 g)
conformation <-0.08-- (p-5 c)
conformation <-0.08-- (p-4 t)
conformation <-0.08-- (p-4 c)
conformation <-0.08-- (p-2 c)
conformation <-0.08-- (p-1 a)
conformation <-0.08-- (p-1 c)
```

B.2 The Splice-Junction Rule-Base

B.2.1 The Original Rules

```
% The rules use a shorthand notation for expressing sequences.
% Namely, the rule:
%       EI-stop ::- @-3 ‘TAA’.
% which means that at position -3, there exists the sequence ’TAA’,
% and this could be expanded to:
%       EI-stop ::- @-3=T, @-2=A, @-1=A.
```

```
% In this shorthand, there is no position 0.

% An exon->intron boundary is defined by a short sequence arround
% the boundary and the absence of a "stop" codon on the exon side
% of the boundary.
        EI :- @-3 `MAGGTRAGT', not(EI-stop).

        EI-stop ::- @-3 `TAA'.
        EI-stop ::- @-3 `TAG'.
        EI-stop ::- @-3 `TGA'.
        EI-stop ::- @-4 `TAA'.
        EI-stop ::- @-4 `TAG'.
        EI-stop ::- @-4 `TGA'.
        EI-stop ::- @-5 `TAA'.
        EI-stop ::- @-5 `TAG'.
        EI-stop ::- @-5 `TGA'.

% An intro->exon boundary is defined by a short sequence arround the
% boundary, the absence of a "stop" codon immediately following the
% boundary and a "pryamidine rich" region preceeding the boundary.
        IE :- pyramidine-rich, @-3 `YAGG', not(IE-stop).

        pyramidine-rich :- 6 of (@-15 `YYYYYYYYYY').

        IE-stop1 ::- @1 `TAA'.
        IE-stop2 ::- @1 `TAG'.
        IE-stop3 ::- @1 `TGA'.
        IE-stop4 ::- @2 `TAA'.
        IE-stop5 ::- @2 `TAG'.
        IE-stop6 ::- @2 `TGA'.
        IE-stop7 ::- @3 `TAA'.
        IE-stop8 ::- @3 `TAG'.
        IE-stop9 ::- @3 `TGA'.

% In addition to the above rules, the following iterative constructs
% can be used as needed to define letters other than {A G C T}.
% These letters represent disjunctive combinations of the nucleotides.
% The codes are standard in the biological literature.
For i from ((-30 to -1) and (+1 to +30))
          {@<i>`Y' ::- @<i>`C'.
           @<i>`Y' ::- @<i>`T'.}

For i from ((-30 to -1) and (+1 to +30))
          {@<i>`M' ::- @<i>`C'.
           @<i>`M' ::- @<i>`A'.}

For i from ((-30 to -1) and (+1 to +30))
          {@<i>`R' ::- @<i>`A'.
           @<i>`R' ::- @<i>`G'.}
```

B.2.2 The Certainty-Factor Rules

```
EI <-0.23-- (p-3 A)
EI <-0.23-- (p-3 C)
EI <-0.23-- (p-2 A)
EI <-0.23-- (p-1 G)
EI <-0.23-- (p1 G)
EI <-0.23-- (p2 T)
EI <-0.23-- (p3 A)
EI <-0.23-- (p3 G)
EI <-0.23-- (p4 A)
EI <-0.23-- (p5 G)
EI <-0.23-- (p6 T)
EI <-0.68-- (NOT EI-stop)

IE <-0.68-- (pyramidine-rich)
IE <-0.25-- (p-3 C)
IE <-0.25-- (p-3 T)
IE <-0.25-- (p-2 A)
IE <-0.25-- (p-1 G)
IE <-0.25-- (p1 G)
IE <-0.68-- (NOT IE-stop)

EI-stop<-1.0--(p-3 T)(p-2 A)(p-1 A)
EI-stop<-1.0--(p-3 T)(p-2 A)(p-1 G)
EI-stop<-1.0--(p-3 T)(p-2 G)(p-1 A)
EI-stop<-1.0--(p-4 T)(p-3 A)(p-2 A)
EI-stop<-1.0--(p-4 T)(p-3 A)(p-2 G)
EI-stop<-1.0--(p-4 T)(p-3 G)(p-2 A)
EI-stop<-1.0--(p-5 T)(p-4 A)(p-3 A)
EI-stop<-1.0--(p-5 T)(p-4 A)(p-3 G)
EI-stop<-1.0--(p-5 T)(p-4 G)(p-3 A)
```

```
IE-stop <-1.0-- (p1 T)(p2 A)(p3 A)
IE-stop <-1.0-- (p1 T)(p2 A)(p3 G)
IE-stop <-1.0-- (p1 T)(p2 G)(p3 A)
IE-stop <-1.0-- (p2 T)(p3 A)(p4 A)
IE-stop <-1.0-- (p2 T)(p3 A)(p4 G)
IE-stop <-1.0-- (p2 T)(p3 G)(p4 A)
IE-stop <-1.0-- (p3 T)(p4 A)(p5 A)
IE-stop <-1.0-- (p3 T)(p4 A)(p5 G)
IE-stop <-1.0-- (p3 T)(p4 G)(p5 A)

pyramidine-rich <-0.32-- (p-15 C)
pyramidine-rich <-0.32-- (p-15 T)
pyramidine-rich <-0.32-- (p-14 C)
pyramidine-rich <-0.32-- (p-14 T)
pyramidine-rich <-0.32-- (p-13 C)
pyramidine-rich <-0.32-- (p-13 T)
pyramidine-rich <-0.32-- (p-12 C)
pyramidine-rich <-0.32-- (p-12 T)
pyramidine-rich <-0.32-- (p-11 C)
pyramidine-rich <-0.32-- (p-11 T)
pyramidine-rich <-0.32-- (p-10 C)
pyramidine-rich <-0.32-- (p-10 T)
pyramidine-rich <-0.32-- (p-9 C)
pyramidine-rich <-0.32-- (p-9 T)
pyramidine-rich <-0.32-- (p-8 C)
pyramidine-rich <-0.32-- (p-8 T)
pyramidine-rich <-0.32-- (p-7 C)
pyramidine-rich <-0.32-- (p-7 T)
pyramidine-rich <-0.32-- (p-6 C)
pyramidine-rich <-0.32-- (p-6 T)
```

B.3 The Mycin Certainty-Factor Rules

```
acute_bacterial_meningitis <- 0.30 -- finding(seizures,yes)
subarachnoid_hemorrhage <- 0.10 -- greater(wbc,Wbc,10.5)
subarachnoid_hemorrhage <- 0.10 -- greater(pmns,Pmns,78)
subarachnoid_hemorrhage <- 0.10 -- greater(bands,Bands,10)
acute_bacterial_meningitis <- 0.20 -- finding(seizures,yes)
acute_bacterial_meningitis <- -0.80 --
                                   lessthan(csfprotein, Csfprotein,45)
acute_bacterial_meningitis <- 0.20 --
                                between(csfprotein, Csfprotein,45,100)
```

```
acute_bacterial_meningitis <- 0.70 --
                                   greater(csfprotein, Csfprotein,100)
viral_meningitis <- 0.30 -- lessthan(csfprotein, Csfprotein,45)
viral_meningitis <- 0.20 -- between(csfprotein, Csfprotein,45,100)
viral_meningitis <- -0.50 -- greater(csfprotein, Csfprotein,500)
chronic_meningitis <- -0.50 -- lessthan(csfprotein, Csfprotein,45)
chronic_meningitis <- 0.0 -- between(csfprotein, Csfprotein,45,100)
chronic_meningitis <- 0.70 -- greater(csfprotein, Csfprotein,100)
acute_bacterial_meningitis <- 0.80 --
         lessthan(csfglucose, Csfglucose,25) finding(csfglucnormal,no)
acute_bacterial_meningitis <- 0.10) --
       between(csfglucose, Csfglucose,25,40) finding(csfglucnormal,no)
acute_bacterial_meningitis <- -0.80 --
          greater(csfglucose, Csfglucose,39) finding(csfglucnormal,no)
chronic_meningitis <- 0.40 -- lessthan(csfglucose, Csfglucose,25)
                              finding(csfglucnormal,no)
chronic_meningitis <- 0.60 -- between(csfglucose, Csfglucose,25,40)
                              finding(csfglucnormal,no)
chronic_meningitis <- 0.0 -- greater(csfglucose, Csfglucose,40)
                             finding(csfglucnormal,no)
viral_meningitis <- -0.80 -- lessthan(csfglucose, Csfglucose,25)
                             finding(csfglucnormal,no)
viral_meningitis <- 0.0 -- between(csfglucose, Csfglucose,25,40)
                           finding(csfglucnormal,no)
viral_meningitis <- 0.50 -- greater(csfglucose, Csfglucose,40)
                            finding(csfglucnormal,no)
acute_bacterial_meningitis <- -0.80 --
                                 greater(csfcellcount, Csfcellcount,4)
                                 lessthan(csfpoly, Csfpoly,35)
acute_bacterial_meningitis <- 0.0 --
                                greater(csfcellcount, Csfcellcount,4)
                                between(csfpoly, Csfpoly,35,65)
acute_bacterial_meningitis <- 0.80 --
                                 greater(csfcellcount, Csfcellcount,4)
                                 greater(csfpoly, Csfpoly,65)
viral_meningitis <- 0.80 -- greater(csfcellcount, Csfcellcount,4)
                            lessthan(csfpoly, Csfpoly,35)
viral_meningitis <- 0.0 -- greater(csfcellcount, Csfcellcount,4)
                           between(csfpoly, Csfpoly,35,65)
viral_meningitis <- -0.80 -- greater(csfcellcount, Csfcellcount,4)
                             greater(csfpoly, Csfpoly,65)
chronic_meningitis <- 0.50 -- greater(csfcellcount, Csfcellcount,4)
                              lessthan(csfpoly, Csfpoly,35)
chronic_meningitis <- 0.0 -- greater(csfcellcount, Csfcellcount,4)
                             between(csfpoly, Csfpoly,35,65)
chronic_meningitis <- -0.50 -- greater(csfcellcount, Csfcellcount,4)
                               greater(csfpoly, Csfpoly,65)
acute_bacterial_meningitis <- -0.90 -- greater(wbc,Wbc,1)
                               lessthan(csfcellcount, Csfcellcount,10)
acute_bacterial_meningitis <- 0.20 -- greater(wbc,Wbc,1)
                            between(csfcellcount, Csfcellcount,10,100)
acute_bacterial_meningitis <- 0.60 -- greater(wbc,Wbc,1)
                          between(csfcellcount, Csfcellcount,100,1500)
acute_bacterial_meningitis <- 0.80 -- greater(wbc,Wbc,1)
                              greater(csfcellcount, Csfcellcount,1500)
acute_bacterial_meningitis <- -0.90 -- finding(wbc,unknown)
                               lessthan(csfcellcount, Csfcellcount,10)
acute_bacterial_meningitis <- 0.20 -- finding(wbc,unknown)
                            between(csfcellcount, Csfcellcount,10,100)
acute_bacterial_meningitis <- 0.60 -- finding(wbc,unknown)
                          between(csfcellcount, Csfcellcount,100,1500)
acute_bacterial_meningitis <- 0.80 -- finding(wbc,unknown)
```

```
                             greater(csfcellcount, Csfcellcount,1500)
viral_meningitis <- -0.70 -- greater(wbc,Wbc,1)
                              lessthan(csfcellcount, Csfcellcount,10)
viral_meningitis <- 0.10 -- greater(wbc,Wbc,1)
                           between(csfcellcount, Csfcellcount,10,100)
viral_meningitis <- 0.60 -- greater(wbc,Wbc,1)
                         between(csfcellcount, Csfcellcount,100,1500)
viral_meningitis <- -0.50 -- greater(wbc,Wbc,1)
                             greater(csfcellcount, Csfcellcount,1500)
viral_meningitis <- -0.70 -- finding(wbc,unknown)
                              lessthan(csfcellcount, Csfcellcount,10)
viral_meningitis <- 0.10 -- finding(wbc,unknown)
                           between(csfcellcount, Csfcellcount,10,100)
viral_meningitis <- 0.60 -- finding(wbc,unknown)
                         between(csfcellcount, Csfcellcount,100,1500)
viral_meningitis <- -0.50 -- finding(wbc,unknown)
                             greater(csfcellcount, Csfcellcount,1500)
chronic_meningitis <- -0.70 -- greater(wbc,Wbc,1)
                              lessthan(csfcellcount, Csfcellcount,10)
chronic_meningitis <- 0.10 -- greater(wbc,Wbc,1)
                           between(csfcellcount, Csfcellcount,10,100)
chronic_meningitis <- 0.60 -- greater(wbc,Wbc,1)
                         between(csfcellcount, Csfcellcount,100,1500)
chronic_meningitis <- -0.50 -- greater(wbc,Wbc,1)
                             greater(csfcellcount, Csfcellcount,1500)
chronic_meningitis <- -0.70 -- finding(wbc,unknown)
                              lessthan(csfcellcount, Csfcellcount,10)
chronic_meningitis <- 0.10 -- finding(wbc,unknown)
                           between(csfcellcount, Csfcellcount,10,100)
chronic_meningitis <- 0.60 -- finding(wbc,unknown)
                         between(csfcellcount, Csfcellcount,100,1500)
chronic_meningitis <- -0.50 -- finding(wbc,unknown)
                            greater(csfcellcount, Csfcellcount,1500)
subarachnoid_hemorrhage <- 0.10 --
                               greater(csfcellcount.Csfcellcount,5)
                               greater(csfpoly, Csfpoly,0)
                               greater(csfprotein, Csfprotein,65)
subarachnoid_hemorrhage <- 0.50 -- finding(stiff_neck_on_flexion,yes)
hypertension <- 0.10 -- finding(saccular_aneurysm,yes)
pckd <- 0.20 -- finding(saccular_aneurysm,yes)
pckd <- 0.90 -- finding(hx_pckd,yes)
pckd <- 0.50 -- finding(family_hx_pckd,yes)
cluster_headache <- 0.30 -- finding(headache_location,unilateral)
diplococcus_pneumoniae <- 0.20 --
                               finding(pneumococcal_pneumonia_hx,yes)
subarachnoid_hemorrhage <- 60 --
                              finding(headache_physical_exertion,yes)
                              finding(headache_frequency,first_time)
migraine <- 0.40 -- finding(headache_location,unilateral)
subarachnoid_hemorrhage <- 0.80 -- finding(retinal_hemorrhage,yes)
diplococcus_pneumoniae <- 0.30 -- finding(cough,yes)
headache_episodic <- -1.00 -- finding(headache_frequency,first_time)
chronic_bacterial_sinusitis <- 0.60 -- finding(nasal_congestion,yes)
leptospirosis <- 0.10 -- finding(headache_location,generalized)
leptospirosis <- 0.10 -- finding(scleral_injection,yes)
cluster_headache <- 0.30 -- finding(scleral_injection,yes)
cluster_headache <- 0.30 -- finding(lacrimation,yes)
allergic_sinusitis <- 0.30 -- finding(scleral_injection,yes)
allergic_sinusitis <- 0.30 -- finding(itchy_eyes,yes)
allergic_sinusitis <- 0.30 -- finding(sneezing,yes)
allergic_sinusitis <- 0.30 -- finding(headache_location,frontal)
acute_meningitis <- 0.30 -- finding(photophobia,yes)
```

```
allergic_sinusitis <- 0.30 -- finding(nasal_congestion,yes)
headtrauma <- 0.20 -- finding(headache,yes)
necktrauma <- 1.00 -- finding(necktraumasigns,yes)
necktrauma <- 0.20 -- finding(stiff_neck_signs,yes)
meningitis <- 0.70 -- finding(redflag_cns_finding,yes)
psychogenic <- 0.30 -- finding(sedatives,yes)
increased_intracranial_pressure <- 0.90 -- finding(tense_fontanel,yes)
high_grade_fever <- 1.00 -- greater(temperature,Temperature,102)
low_grade_fever <- -1.00 -- greater(temperature,Temperature,102)
low_grade_fever <- 1.00 -- between(temperature,Temperature,99,102)
high_grade_fever <- -1.00 -- between(temperature,Temperature,99,102)
acute_bacterial_meningitis <- 0.50 -- finding(high_grade_fever,yes)
increased_intracranial_pressure <- 0.95 --
                                  finding(ctscan_ventricular_size,yes)
otitis_media <- 0.90 -- finding(otitis_media_signs,yes)
brain_abscess <- 1.00 -- finding(intracerebral_pus,yes)
av_malformation <- 0.20 -- finding(subarachnoid_hemorrhage,yes)
neoplastic <- 0.30 -- finding(weight_loss,yes)
brain_aneurysm <- 0.30 -- finding(intracerebral_hemorrhage,yes)
hypertension <- 0.30 -- finding(intracerebral_hemorrhage,yes)
traumatic_process <- 0.20 -- finding(alcoholic,yes)
epi_subdural_hemorrhage <- 1.00 -- finding(epi_subdural_hematoma,yes)
headtrauma <- 0.80 -- finding(epi_subdural_hemorrhage,yes)
chronic_bacterial_sinusitis <- 0.30 --
                                     finding(epi_subdural_empyema,yes)
subarachnoid_hemorrhage <- 0.20 --
                          finding(increased_intracranial_pressure,yes)
other_ic_pressure_causes <- 0.30 --
                          finding(increased_intracranial_pressure,yes)
chronic_lung_infection <- 0.30 -- finding(brain_abscess,yes)
chronic_bacterial_sinusitis <- 0.30 -- finding(brain_abscess,yes)
chronic_ear_infection <- 0.30 -- finding(brain_abscess,yes)
hemorrhage <- 0.30 -- finding(bleeding_disorder,yes)
csfglucnormal <- -1.00 -- lessthan(csfglucose, Csfglucose,40)
csfglucnormal <- -1.00 --
                 lessthan(quotient,quotient(csfglucose,bloodgluc),0.4)
increased_intracranial_pressure <- 0.95 --
                                   finding(increased_csf_pressure,yes)
neonate <- -1.00 -- greater(age,Age,0.4167)
meningitis <- 0.50 -- finding(stiff_neck_on_flexion,yes)
                      finding(neonate,yes)
febrile <- 0.70 -- finding(shaking_chills,yes)
neisseria_meningitidis <- 0.20 -- finding(crowded_environment,yes)
mycobacterium_tb_meningitis <- 0.80 -- finding(cxrsug_active_tb,yes)
mycobacterium_tb_meningitis <- 0.20 -- finding(steroids,yes)
mycobacterium_tb_meningitis <- 0.20 -- finding(cytotoxic,yes)
migraine <- 0.20 -- finding(nausea,yes)
mycobacterium_tb_meningitis <- 0.20 -- finding(granuloma_hx,yes)
chronic_ear_infection <- 0.30 -- finding(epi_subdural_empyema,yes)
csfglucnormal <- 1.00 -- greater(csfglucose, Csfglucose,40)
                         finding(bloodgluc,unknown)
csfglucnormal <- 1.00 -- greater(csfglucose, Csfglucose,40)
                  greater(quotient,quotient(csfglucose,bloodgluc),0.4)
cryptococcus <- 0.70 -- finding(crypto_serology,yes)
coccidioides <- 0.70 -- finding(cocci_serology,yes)
leukopenia <- 1.00 -- lessthan(wbc,Wbc,2.5)
headtrauma <- 0.90 -- finding(headtraumasigns,yes)
migraine <- 0.30 -- finding(headache_episodic,yes)
temporal_arteritis <- 40 -- finding(temporal_tenderness,yes)
e_coli <- 0.525 -- finding(neurosurgery,no) greater(age,Age,10)
                   finding(nosocomial,yes)
pseudomonas_aeruginosa <- 0.21 -- finding(neurosurgery,no)
```

```
                            greater(age,Age,10) finding(nosocomial,yes)
klebsiella_pneumoniae <- 0.35 -- finding(neurosurgery,no)
                            greater(age,Age,10) finding(nosocomial,yes)
staphylococcus_coag_pos <- 0.21 -- finding(neurosurgery,no)
                                    greater(age,Age,10)
                                    finding(nosocomial,yes)
viral_meningitis <- 0.30 -- finding(csfglucnormal,yes)
bacterial_endocarditis <- 0.90 -- finding(mycotic_aneurysm,yes)
intracranial_mass_lesion <- 0.90 -- finding(ctscan_mass_lesion,yes)
brain_abscess <- 0.80 -- finding(ctscan_abscess,yes)
intracranial_hematoma <- 0.80 -- finding(ctscan_hematoma,yes)
intracranial_tumor <- 0.80 -- finding(ctscan_tumor,yes)
cryptococcus <- 0.20 -- finding(low_grade_fever,yes)
intracranial_mass_lesion <- 0.50 -- finding(focalsigns,yes)
cryptococcus <- 0.90 -- finding(india_ink,yes)
mycobacterium_tb_meningitis <- 0.60 -- finding(pos_ppd,yes)
mycobacterium_tb_meningitis <- -0.30 -- finding(pos_ppd,no)
migraine <- 0.30 -- finding(headache_episodic,yes)
migraine <- 0.70 -- finding(headache_visual_prodrome,yes)
glaucoma <- 0.30 -- finding(headache,yes) finding(red_painful_eye,yes)
increased_intracranial_pressure <- 0.90 -- finding(papilledema,yes)
temporal_arteritis <- 0.50 -- finding(headache_location,temporal)
                        greater(headache_severity,Headache_severity,2)
                        greater(age,Age,60)
temporal_arteritis <- 0.40 -- finding(temporal_tenderness,yes)
increased_intracranial_pressure <- 0.90 -- finding(enlarged_head,yes)
increased_intracranial_pressure <- 0.70 --
                                 finding(headache_chronicity, chronic)
                                 finding(vomiting,yes)
intracranial_mass_lesion <- 0.75 --
                           finding(increased_intracranial_pressure,yes)
brain_abscess <- 0.40 -- finding(immunosuppressed,yes)
                          finding(redflag_cns_finding,yes)
increased_intracranial_pressure <- 0.20 -- finding(seizures,yes)
intracranial_mass_lesion <- 0.40 --
                      finding(headache_progression,steadily_worsening)
                      finding(headache_chronicity, chronic)
subarachnoid_hemorrhage <- 0.30 -- finding(syncope,yes)
increased_csf_pressure <- 1.00 --
                                greater(csf_pressure, Csf_pressure,20)
tension_headache <- 0.30 -- finding(headache_chronicity, chronic)
tension_headache <- 0.815 -- finding(headache_quality,pressure)
tension_headache <- 0.70 -- finding(headache_emotional,yes)
headache_chronicity(acute) <- 0.80 --
                       lessthan(headache_duration,Headache_duration,3)
headache_chronicity(acute) <- 0.20 --
                     between(headache_duration,Headache_duration,3,9)
headache_chronicity(acute) <- 0.0 --
                    between(headache_duration,Headache_duration,9,183)
headache_chronicity(acute) <- 0.0 --
                      greater(headache_duration,Headache_duration,183)
headache_chronicity(subacute) <- 0.20 --
                       lessthan(headache_duration,Headache_duration,3)
headache_chronicity(subacute) <- 0.80 --
                        between(headache_duration,Headache_duration,3)
headache_chronicity(subacute) <- 0.30 --
                    between(headache_duration,Headache_duration,9,183)
headache_chronicity(subacute) <- 0.0 --
                      greater(headache_duration,Headache_duration,183)
headache_chronicity(chronic) <- 0.0 --
                       lessthan(headache_duration,Headache_duration,3)
headache_chronicity(chronic) <- 0.0 --
```

```
                      between(headache_duration,Headache_duration,3,9)
headache_chronicity(chronic) <- 0.80 --
                    between(headache_duration,Headache_duration,9,183)
headache_chronicity(chronic) <- 1.00 --
                      greater(headache_duration,Headache_duration,183)
meningitis <- 0.80 -- finding(seizures,yes)
                      finding(tense_fontanel,yes)
neonate <- 1.00 -- lessthan(age,Age,0.4167)
acute_bacterial_meningitis <- 0.60 --
                lessthan(cns_finding_duration, Cns_finding_duration,2)
acute_bacterial_meningitis <- 0.30 --
               between(cns_finding_duration, Cns_finding_duration,2,5)
acute_bacterial_meningitis <- 0.0 --
               between(cns_finding_duration, Cns_finding_duration,5,8)
acute_bacterial_meningitis <- 0.0 --
              between(cns_finding_duration, Cns_finding_duration,8,12)
acute_bacterial_meningitis <- -0.60 --
             between(cns_finding_duration, Cns_finding_duration,12,60)
acute_bacterial_meningitis <- -0.80 --
                greater(cns_finding_duration, Cns_finding_duration,60)
viral_meningitis <- -0.20 --
                lessthan(cns_finding_duration, Cns_finding_duration,2)
viral_meningitis <- 0.30 --
               between(cns_finding_duration, Cns_finding_duration,2,5)
viral_meningitis <- 0.60 --
               between(cns_finding_duration, Cns_finding_duration,5,8)
viral_meningitis <- 0.20 --
              between(cns_finding_duration, Cns_finding_duration,8,12)
viral_meningitis <- -0.20 --
             between(cns_finding_duration, Cns_finding_duration,12,60)
viral_meningitis <- -0.60 --
                greater(cns_finding_duration, Cns_finding_duration,60)
chronic_meningitis <- -0.80 --
                lessthan(cns_finding_duration, Cns_finding_duration,2)
chronic_meningitis <- -0.20 --
               between(cns_finding_duration, Cns_finding_duration,2,5)
chronic_meningitis <- 0.0 --
               between(cns_finding_duration, Cns_finding_duration,5,8)
chronic_meningitis <- 0.20 --
              between(cns_finding_duration, Cns_finding_duration,8,12)
chronic_meningitis <- 0.60 --
             between(cns_finding_duration, Cns_finding_duration,12,60)
chronic_meningitis <- 0.80 --
                greater(cns_finding_duration, Cns_finding_duration,60)
acute_bacterial_meningitis <- 0.60 --
                        finding(headache_chronicity,acute)
                        finding(headache_onset,abrupt)
                        greater(headache_severity,Headache_severity,3)
viral_meningitis <- 0.40 -- finding(headache_chronicity,acute)
                        finding(headache_onset,abrupt)
                        greater(headache_severity,Headache_severity,3)
subarachnoid_hemorrhage <-0.60 -- finding(headache_chronicity,acute)
                         finding(headache_onset,abrupt)
                        greater(headache_severity,Headache_severity,3)
mycobacterium_tb_meningitis <- 0.80 -- finding(cxrsug_active_tb)
meningitis <- -0.80 -- lessthan(csfcellcount, Csfcellcount,10)
                       lessthan(csfprotein, Csfprotein,40)
meningitis <- -0.70 -- lessthan(csfcellcount, Csfcellcount,5)
                       greater(wbc,Wbc,1)
coccidioides <- 0.50 -- finding(cocci_endemic,yes)
                        finding(race,asian) finding(compromised,yes)
coccidioides <- 0.50 -- finding(cocci_endemic,yes)
```

```
                        finding(race,black) finding(compromised,yes)
coccidioides <- 0.50 -- finding(cocci_endemic,yes)
                        finding(race,indian) finding(compromised,yes)
viral_meningitis <- 0.40 -- finding(exanthems,yes)
viral_meningitis <- 0.10 -- finding(exp_exanthems,yes)
                            finding(exanthems,no)
staphylococcus_coag_pos <- 0.525 -- finding(skininfect,yes)
streptococcus_group_a <- 0.35 -- finding(skininfect,yes)
mycobacterium_tb_meningitis <- 0.20 -- finding(steroids,yes)
mycobacterium_tb_meningitis <- 0.20 -- finding(cytotoxic,yes)
infectious_process <- 0.50 -- greater(wbc,Wbc,10.5)
infectious_process <- 0.50 -- greater(pmns,Pmns,78)
infectious_process <- 0.50 -- greater(bands,Bands,10)
mycobacterium_tb_meningitis <- 0.20 -- finding(ocnerve,yes)
increased_intracranial_pressure <- 0.80 -- finding(diplopia,yes)
viral_meningitis <- 0.20 -- finding(vesicerupt,yes)
e_coli <- 0.525 -- finding(systemic_compromised,yes)
klebsiella_pneumoniae <- 0.35 -- finding(systemic_compromised,yes)
pseudomonas_aeruginosa <- 0.28 -- finding(systemic_compromised,yes)
infectious_process <- 0.70 -- finding(febrile,yes)
meningitis <- 0.50 -- finding(stiff_neck_on_flexion,yes)
                      finding(headache,yes)
partially_treated_bacterial_meningitis <- -1.00 --
                                       finding(antimicrobialrx,no)
intracerebral_hemorrhage <- 1.00 --
                                   finding(intracerebral_hematoma,yes)
headtrauma <- 0.30 -- finding(intracerebral_hemorrhage,yes)
brain_aneurysm <- 0.60 -- finding(subarachnoid_hemorrhage,yes)
staphylococcus_coag_pos <- 0.525 -- finding(cnsradiate,yes)
staphylococcus_coag_pos <- 0.525 -- lessthan(age,Age,9)
                                    finding(cnsmalform,yes)
low_platelets <- 0.70 -- finding(leukemia,yes)
low_platelets <- 0.70 -- finding(lymphoma,yes)
low_platelets <- 0.70 -- finding(cytotoxic,yes)
e_coli <- 0.20 -- finding(cns_compromised,no)
                  lessthan(age,Age,0.166667)
e_coli <- 0.0 -- finding(cns_compromised,no)
                 between(age,Age,0.166667,1)
e_coli <- 0.0 -- finding(cns_compromised,no) between(age,Age,1,5)
e_coli <- 0.0 -- finding(cns_compromised,no) between(age,Age,5,15)
e_coli <- 0.0 -- finding(cns_compromised,no) between(age,Age,15,55)
e_coli <- 0.0 -- finding(cns_compromised,no) greater(age,Age,55)
klebsiella_pneumoniae <- 0.14 -- finding(cns_compromised,no)
                                 lessthan(age,Age,0.166667)
klebsiella_pneumoniae <- 0.0 -- finding(cns_compromised,no)
                                between(age,Age,0.166667,1)
klebsiella_pneumoniae <- 0.0 -- finding(cns_compromised,no)
                                between(age,Age,1,5)
klebsiella_pneumoniae <- 0.0 -- finding(cns_compromised,no)
                                between(age,Age,5,15)
klebsiella_pneumoniae <- 0.0 -- finding(cns_compromised,no)
                                between(age,Age,15,55)
klebsiella_pneumoniae <- 0.0 -- finding(cns_compromised,no)
                                greater(age,Age,55)
listeria <- 0.112 -- finding(cns_compromised,no)
                     lessthan(age,Age,0.166667)
listeria <- 0.0 -- finding(cns_compromised,no)
                   between(age,Age,0.166667,1)
listeria <- 0.0 -- finding(cns_compromised,no) between(age,Age,1,5)
listeria <- 0.0 -- finding(cns_compromised,no) between(age,Age,5,15)
listeria <- 0.0 -- finding(cns_compromised,no) between(age,Age,15,55)
listeria <- 0.0 -- finding(cns_compromised,no) greater(age,Age,55)
```

```
streptococcus_group_b <- 0.20 -- finding(cns_compromised,no)
                                 lessthan(age,Age,0.166667)
streptococcus_group_b <- 0.0 -- finding(cns_compromised,no)
                                between(age,Age,0.166667,1)
streptococcus_group_b <- 0.0 -- finding(cns_compromised,no)
                                between(age,Age,1,5)
streptococcus_group_b <- 0.0 -- finding(cns_compromised,no)
                                between(age,Age,5,15)
streptococcus_group_b <- 0.0 -- finding(cns_compromised,no)
                                between(age,Age,15,55)
streptococcus_group_b <- 0.0 -- finding(cns_compromised,no)
                                greater(age,Age,55)
hemophilus_influenzae <- 0.0 -- finding(cns_compromised,no)
                                lessthan(age,Age,0.166667)
hemophilus_influenzae <- 0.20 -- finding(cns_compromised,no)
                                 between(age,Age,0.166667,1)
hemophilus_influenzae <- 0.20 -- finding(cns_compromised,no)
                                 between(age,Age,1,5)
hemophilus_influenzae <- 0.20 -- finding(cns_compromised,no)
                                 between(age,Age,5,15)
hemophilus_influenzae <- 0.0 -- finding(cns_compromised,no)
                                between(age,Age,15,55)
hemophilus_influenzae <- 0.0 -- finding(cns_compromised,no)
                                greater(age,Age,55)
diplococcus_pneumoniae <- 0.0 -- finding(cns_compromised,no)
                                 lessthan(age,Age,0.166667)
diplococcus_pneumoniae <- 0.112 -- finding(cns_compromised,no)
                                   between(age,Age,0.166667,1)
diplococcus_pneumoniae <- 0.112 -- finding(cns_compromised,no)
                                   between(age,Age,1,5)
diplococcus_pneumoniae <- 0.20 -- finding(cns_compromised,no)
                                  between(age,Age,5,15)
diplococcus_pneumoniae <- 0.20 -- finding(cns_compromised,no)
                                  between(age,Age,15,55)
diplococcus_pneumoniae <- 0.20 -- finding(cns_compromised,no)
                                  greater(age,Age,55)
neisseria_meningitidis <- 0.0 -- finding(cns_compromised,no)
                                 lessthan(age,Age,0.166667)
neisseria_meningitidis <- 0.112 -- finding(cns_compromised,no)
                                   between(age,Age,0.166667,1)
neisseria_meningitidis <- 0.112 -- finding(cns_compromised,no)
                                   between(age,Age,1,5)
neisseria_meningitidis <- 0.20 -- finding(cns_compromised,no)
                                  between(age,Age,5,15)
neisseria_meningitidis <- 0.20 -- finding(cns_compromised,no)
                                  between(age,Age,15,55)
neisseria_meningitidis <- 0.0 -- finding(cns_compromised,no)
                                 greater(age,Age,55)
staphylococcus_coag_pos <- 0.0 -- finding(cns_compromised,no)
                                  lessthan(age,Age,0.166667)
staphylococcus_coag_pos <- 0.0 -- finding(cns_compromised,no)
                                  between(age,Age,0.166667,1)
staphylococcus_coag_pos <- 0.0 -- finding(cns_compromised,no)
                                  between(age,Age,1,5)
staphylococcus_coag_pos <- 0.0 -- finding(cns_compromised,no)
                                  between(age,Age,5,15)
staphylococcus_coag_pos <- 0.0 -- finding(cns_compromised,no)
                                  between(age,Age,15,55)
staphylococcus_coag_pos <- 0.112 -- finding(cns_compromised,no)
                                    greater(age,Age,55)
streptococcus_species <- 0.0 -- finding(cns_compromised,no)
                                lessthan(age,Age,0.166667)
```

```
streptococcus_species <- 0.0 -- finding(cns_compromised,no)
                                between(age,Age,0.166667,1)
streptococcus_species <- 0.0 -- finding(cns_compromised,no)
                                between(age,Age,1,5)
streptococcus_species <- 0.0 -- finding(cns_compromised,no)
                                between(age,Age,5,15)
streptococcus_species <- 0.0 -- finding(cns_compromised,no)
                                between(age,Age,15,55)
streptococcus_species <- 0.112 -- finding(cns_compromised,no)
                                  greater(age,Age,55)
recentcnsdefect <- 1.00 -- finding(cnsdef,yes)
                           lessthan(cnsdeftime, Cnsdeftime,2)
staphylococcus_coag_pos <- 0.525 -- finding(neurosurgery,yes)
                        lessthan(neurotime,Neurotime,2)
                        finding(neurosurgtype,other_ventricular_shunt)
staphylococcus_coag_pos <- 0.525 -- finding(neurosurgery,yes)
                                    lessthan(neurotime,Neurotime,2)
                                    finding(neurosurgtype,no_shunt)
staphylococcus_coag_neg <- 0.35 -- finding(neurosurgery,yes)
                                   lessthan(neurotime,Neurotime,2)
                        finding(neurosurgtype,other_ventricular_shunt)
staphylococcus_coag_neg <- 0.35 -- finding(neurosurgery,yes)
                                   lessthan(neurotime,Neurotime,2)
                                   finding(neurosurgtype,no_shunt)
staphylococcus_group_a <- 0.21 -- finding(neurosurgery,yes)
                                  lessthan(neurotime,Neurotime,2)
                       finding(neurosurgtype,other_ventricular_shunt)
staphylococcus_group_a <- 0.21 -- finding(neurosurgery,yes)
                                  lessthan(neurotime,Neurotime,2)
                                  finding(neurosurgtype,no_shunt)
e_coli <- 0.28 -- finding(neurosurgery,yes)
                  lessthan(neurotime,Neurotime,2)
                  finding(neurosurgtype,other_ventricular_shunt)
e_coli <- 0.28 -- finding(neurosurgery,yes)
                  lessthan(neurotime,Neurotime,2)
                  finding(neurosurgtype,no_shunt)
klebsiella_pneumoniae <- 0.21 -- finding(neurosurgery,yes)
                                 lessthan(neurotime,Neurotime,2)
                        finding(neurosurgtype,other_ventricular_shunt)
klebsiella_pneumoniae <- 0.21 -- finding(neurosurgery,yes)
                                 lessthan(neurotime,Neurotime,2)
                                 finding(neurosurgtype,no_shunt)
pseudomonas_aeruginosa <- 0.20 -- finding(neurosurgery,yes)
                                  lessthan(neurotime,Neurotime,2)
                        finding(neurosurgtype,other_ventricular_shunt)
pseudomonas_aeruginosa <- 0.20 -- finding(neurosurgery,yes)
                                   lessthan(neurotime,Neurotime,2)
                                  finding(neurosurgtype,no_shunt)
e_coli <- 0.56 -- finding(neurosurgtype,ventricular_ureteral_shunt)
klebsiella_pneumoniae <- 0.525 --
                     finding(neurosurgtype,ventricular_ureteral_shunt)
proteus_non_mirabilis <- 0.35 --
                     finding(neurosurgtype,ventricular_ureteral_shunt)
proteus_mirabilis <- 0.35 --
                     finding(neurosurgtype,ventricular_ureteral_shunt)
pseudomonas_aeruginosa <- 0.28 --
                     finding(neurosurgtype,ventricular_ureteral_shunt)
staphylococcus_coag_pos <- 0.42 --
                     finding(neurosurgtype,ventricular_ureteral_shunt)
staphylococcus_coag_neg <- 0.28 --
                     finding(neurosurgtype,ventricular_ureteral_shunt)
diplococcus_pneumoniae <- 0.35 -- finding(lymphoma,yes)
```

```
diplococcus_pneumoniae <- 0.35 -- finding(leukemia,yes)
listeria <- 0.35 -- finding(lymphoma,yes)
listeria <- 0.35 -- finding(leukemia,yes)
cryptococcus <- 0.70 -- finding(lymphoma,yes)
cryptococcus <- 0.70 -- finding(leukemia,yes)
mycobacterium_tb_meningitis <- 0.10 -- finding(exp_tb,yes)
diplococcus_pneumoniae <- 0.35 -- finding(cxrsug_lobar_pneumonia,yes)
diplococcus_pneumoniae <- 0.525 -- finding(sicklecell,yes)
e_coli <- 0.20 -- finding(alcoholic,yes)
diplococcus_pneumoniae <- 0.21 -- finding(alcoholic,yes)
neisseria_meningitidis <- 0.56 --
                               finding(epid_meningococcal_disease,yes)
neisseria_meningitidis <- 0.525 -- finding(purpuric_rash,yes)
neisseria_meningitidis <- 0.21 -- finding(petechial_rash,yes)
                                  finding(purpuric_rash,no)
                                  finding(low_platelets,no)
listeria <- 0.20 -- finding(steroids,yes) finding(lymphoma,no)
                    finding(leukemia,no)
listeria <- 0.20 -- finding(cytotoxic,yes) finding(lymphoma,no)
                    finding(leukemia,no)
hemophilus_influenzae <- 0.35 -- finding(epiglottitis,yes)
diplococcus_pneumoniae <- 0.28 -- finding(epiglottitis,yes)
hemophilus_influenzae <- 0.525 -- finding(otitis_media_signs,yes)
diplococcus_pneumoniae <- 0.49 -- finding(otitis_media_signs,yes)
staphylococcus_coag_pos <- 0.525 -- finding(pent_headtrauma,yes)
                                    finding(traumadate,unknown)
staphylococcus_coag_pos <- 0.525 -- finding(pent_headtrauma,yes)
                                    lessthan(traumadate,Traumadate,60)
staphylococcus_coag_pos <- 0.21 -- finding(pent_headtrauma,yes)
                                   greater(traumadate,Traumadate,59)
streptococcus_group_a <- 0.28 -- finding(pent_headtrauma,yes)
                                 finding(traumadate,unknown)
streptococcus_group_a <- 0.28 -- finding(pent_headtrauma,yes)
                                 lessthan(traumadate,Traumadate,60)
streptococcus_group_a <- 0.0 -- finding(pent_headtrauma,yes)
                                greater(traumadate,Traumadate,59)
diplococcus_pneumoniae <- 0.21 -- finding(pent_headtrauma,yes)
                                  finding(traumadate,unknown)
diplococcus_pneumoniae <- 0.21 -- finding(pent_headtrauma,yes)
                                  lessthan(traumadate,Traumadate,60)
diplococcus_pneumoniae <- 0.35 -- finding(pent_headtrauma,yes)
                                  greater(traumadate,Traumadate,59)
diplococcus_pneumoniae <- 0.63 -- finding(headtraumasigns,yes)
                                  finding(pent_headtrauma,no)
                                  finding(traumadate,unknown)
diplococcus_pneumoniae <- 0.63 -- finding(headtraumasigns,yes)
                                  finding(pent_headtrauma,no)
                                  lessthan(traumadate,Traumadate,60)
diplococcus_pneumoniae <- 0.525 -- finding(headtraumasigns,yes)
                                   finding(pent_headtrauma,no)
                                   greater(traumadate,Traumadate,59)
diplococcus_pneumomiae <- 0.21 -- finding(splenectomy,yes)
coccidioides <- 0.30 -- finding(cocci_endemic,yes)
pseudomonas_aeruginosa <- 0.35 -- finding(burned,yes)
viral_meningitis <- 0.20 -- finding(epiglottitis,yes)
bacterial_meningitis <- 0.20 -- finding(pneumococcal_pneumonia_hx,yes)
bacterial_meningitis <- 0.30 -- finding(cough,yes)
bacterial_meningitis <- 0.20 -- finding(crowded_environment,yes)
fungal_meningitis <- 0.70 -- finding(crypto_serology,yes)
bacterial_meningitis <- 0.525 -- finding(neurosurgery,no)
                                 greater(age,A,10)
                                 finding(nosocomial,yes)
```

```
fungal_meningitis <- 0.20 -- finding(low_grade_fever,yes)
fungal_meningitis <- 0.90 -- finding(india_ink,yes)
fungal_meningitis <- 0.50 -- finding(cocci_endemic,yes)
                             finding(race,asian)
                             finding(compromised,yes)
fungal_meningitis <- 0.50 -- finding(cocci_endemic,yes)
                             finding(race,black)
                             finding(compromised,yes)
fungal_meningitis <- 0.50 -- finding(cocci_endemic,yes)
                             finding(race,indian)
                             finding(compromised,yes)
bacterial_meningitis <- 0.525 -- finding(skininfect,yes)
bacterial_meningitis <- 0.525 -- finding(systemic_compromised,yes)
bacterial_meningitis <- 0.525 -- finding(cnsradiate,yes)
bacterial_meningitis <- 0.525 -- lessthan(age,A,9)
                                 finding(cnsmalform,yes)
bacterial_meningitis <- 0.20 -- finding(cns_compromised,no)
                                lessthan(age,A,0.166667)
bacterial_meningitis <- 0.20 -- finding(cns_compromised,no)
                                between(age,A,0.166667,1)
bacterial_meningitis <- 0.20 -- finding(cns_compromised,no)
                                between(age,A,1,5)
bacterial_meningitis <- 0.20 -- finding(cns_compromised,no)
                                between(age,A,5,15)
bacterial_meningitis <- 0.20 -- finding(cns_compromised,no)
                                between(age,A,15,55)
bacterial_meningitis <- 0.20 -- finding(cns_compromised,no)
                                greater(age,A,55)
bacterial_meningitis <- 0.525 -- finding(neurosurgery,yes)
                                 lessthan(neurotime,A,2)
                         finding(neurosurgtype,other_ventricular_shunt)
bacterial_meningitis <- 0.525 -- finding(neurosurgery,yes)
                                 lessthan(neurotime,A,2)
                                 finding(neurosurgtype,no_shunt)
bacterial_meningitis <- 0.56 --
                      finding(neurosurgtype,ventricular_ureteral_shunt)
fungal_meningitis <- 0.70 -- finding(lymphoma,yes)
fungal_meningitis <- 0.70 -- finding(leukemia,yes)
bacterial_meningitis <- 0.35 -- finding(cxrsug_lobar_pneumonia,yes)
bacterial_meningitis <- 0.525 -- finding(sicklecell,yes)
bacterial_meningitis <- 0.21 -- finding(alcoholic,yes)
bacterial_meningitis <- 0.56 --
                               finding(epid_meningococcal_disease,yes)
bacterial_meningitis <- 0.525 -- finding(purpuric_rash,yes)
bacterial_meningitis <- 0.21 -- finding(petechial_rash,yes)
                                finding(purpuric_rash,no)
                                finding(low_platelets,no)
bacterial_meningitis <- 0.20 -- finding(steroids,yes)
                                finding(lymphoma,no)
                                finding(leukemia,no)
bacterial_meningitis <- 0.20 -- finding(cytotoxic,yes)
                                finding(lymphoma,no)
                                finding(leukemia,no)
bacterial_meningitis <- 0.35 -- finding(epiglottitis,yes)
bacterial_meningitis <- 0.525 -- finding(otitis_media_signs,yes)
bacterial_meningitis <- 0.525 -- finding(pent_headtrauma,yes)
                                 finding(traumadate,unknown)
bacterial_meningitis <- 0.525 -- finding(pent_headtrauma,yes)
                                 lessthan(traumadate,A,60)
bacterial_meningitis <- 0.35 -- finding(pent_headtrauma,yes)
                                greater(traumadate,A,59)
bacterial_meningitis <- 0.63 -- finding(headtraumasigns,yes)
```

```
                               finding(pent_headtrauma,no)
                               finding(traumadate,unknown)
bacterial_meningitis <- 0.63 -- finding(headtraumasigns,yes)
                               finding(pent_headtrauma,no)
                               lessthan(traumadate,A,60)
bacterial_meningitis <-0.525 -- finding(headtraumasigns,yes)
                               finding(pent_headtrauma,no)
                               greater(traumadate,A,59)
fungal_meningitis <- 0.30 -- finding(cocci_endemic,yes)
bacterial_meningitis <- 0.35 -- finding(burned,yes)
```

B.4 The Soybean Diagnosis Rule-Base

B.4.1 The Original Rules

```
The Significant conditions

diaporthe_stem_canker <-  (date ?d)(>= ?d 8)(<= ?d 9)(high_precip)
                          (stem_cankers above_sec_nde)
                          (fruiting_bodies present)(fruit_pods norm)

charcoal_rot <- (date ?d)(>= ?d 7)(<= ?d 8)(low_precip)(high_temp)
                (plant_growth abnorm)(leaves abnorm)(stem abnorm)
                (sclerotia present)(roots rotted)(int_discolor black)

rhizoctonia_root_rot <- (date ?d)(>= ?d 5)(<= ?d 6)
                        (plant_stand <_normal)(temp <_norm)
                        (precip <_norm)(leaves abnorm)(stem abnorm)
                        (canker_lesion brown)
                        (roots rotted)(hail_canker_relation)

phytophthora_root_rot <- (date ?d)(>= ?d 4)(<= ?d 8)
                         (plant_stand <_normal)(date_precip_relation1)
                         (date_temp_relation1)
                         (area_damaged low_areas)
                         (plant_growth abnorm)
                         (leaves abnorm)(stem abnorm)
                         (stem_cankers above_soil)
                         (date_canker_relation)
                         (roots rotted)

brown_stem_rot <- (date ?d)(>= ?d 7)(<= ?d 9)(precip >_norm)
                  (low_temp)(leaves abnorm)(stem abnorm)
                  (int_discolor brown)(lodging yes)

powdery_mildew <- (leaves abnorm)(leaf_mild upper_surf)

downy_mildew <- (date ?d)(>= ?d 6)(<= ?d 8)(high_precip)
                (area_damaged whole_field)
                (leaves abnorm) (leafspots-halo no_yellow_halos)
                (leaf_mild lower_surf)
                (date_seed_condition)(mold_growth present)

brown_spot <- (leaves abnorm)(leafspots-halos)
              (leafspots-marg no_w-s_marg)(leafspot_size >_1/8)

bacterial_blight <- (date_condition)(date_precip_relation2)
                   (date_temp_relation2)(leaves abnorm)
```

```
                   (leafspots-halo yellow_halos)
                   (leafspots-marg w-s_marg)(leafspot_size <_1/8)
                   (leaf_shread present)

bacterial_pustule <- (date ?d)(>= ?d 6)(<= ?d 8)(high_precip)
                     (leaves abnorm)
                     (leafspots-halo no_yellow_halos)
                     (leafspots-marg no_w-s_marg)(leafspot_size <_1/8)
                     (leaf_shread present)

purple_seed_stain <- (date ?d)(>= ?d 9)(<= ?d 10)(seed abnorm)
                     (seed_discolor present)(seed_size <_norm)

anthracnose <- (date ?d)(>= ?d 8)(<= ?d 10)(high_precip)(stem abnorm)
               (canker_lesion brown)(fruiting_bodies present)
               (date_seed_condition)(fruit_spot_condition)

phyllosticta_leaf_spot <- (date ?d)(>= ?d 4)(<= ?d 7)(high_precip)
                          (leaves abnorm)
                          (leafspots-halo no_yellow_halos)
                         (leafspots-marg no_w-s_marg)
                          (leafspot_size >_1/8)(leaf_shread present)

alternaria_leaf_spot <- (date ?d)(>= ?d 7)(<= ?d 10)(leaves abnorm)
                        (leafspots-halo no_yellow_halos)
                        (leafspots-marg no_w-s_marg)
                        (leafspot_size >_1/8)(leaf_shread absent)

frog_eye_leaf_spot <- (date ?d)(>= ?d 7)(<= ?d 9)(high_precip)
                      (leaves abnorm)(leafspots-halo no_yellow_halos)
                      (leafspots-marg no_w-s_marg)
                      (leafspot_size >_1/8)


The Confirmatory Conditions

diaporthe_stem_canker <- (>= temp norm)(canker_lesion brown)
                         (crop_hist_er1)

charcoal_rot <- (area_damaged upper_areas)(severity severe)
                (seed_size <_norm)(crop_hist_er2)

rhizoctonia_root_rot <- (fruiting_bodies absent)
                        (external_decay firm_and_dry)(mycelium absent)

phytophthora_root_rot <- (>= crop_hist same_lst_two_yrs)

brown_stem_rot <- (seed_size <_norm)(crop_hist_er3)

powdery_mildew <- (>= date 8)(<= date 9)

brown_spot <- (dates1)(>= precip norm)

bacterial_pustule <- (>= crop_hist same_lst_yr)

purple_seed_stain <- (>= date 8)(<= date 9)(>= precip norm)
                     (leaves abnorm)

anthracnose <- (area_damaged whole_field))

phyllosticta_leaf_spot <- (damage_date_vs_temp)(date_vs_temp3)
```

```
alternaria_leaf_spot <- (date_vs_pods)(pods_vs_spots)(seed_vs_color)

frog_eye_leaf_spot) <- (date_vs_spots)(stem_cankers above_sec_nde)
                       (canker_lesion tan)(fruiting_bodies absent)


Definitional Conditions

high_precip <- (precip norm)
high_precip <- (precip >_norm)
low_precip <- (precip norm)
low_precip <- (precip <_norm)
high_temp <- (temp norm)
high_temp <- (temp >_norm)
low_temp <- (temp norm)
low_temp <- (temp <_norm)
hail_canker_relation <- (hail no) (stem_cankers below_soil)
hail_canker_relation <- (hail no) (stem_cankers above_soil)
hail_canker_relation <- (hail yes) (stem_cankers above_sec_nde)
date_precip_relation1 <- (date ?d) (>= ?d 4) (<= ?d 6) (precip norm)
date_precip_relation1 <- (date ?d) (>= ?d 7) (<= ?d 8) (precip >_norm)
date_precip_relation1 <- (date ?d) (> ?d 8)
date_temp_relation1 <- (date 4) (temp >_norm)
date_temp_relation1 <- (date ?d) (>= ?d 5) (<= ?d 8) (temp norm)
date_temp_relation1 <- (date ?d) (> ?d 8)
date_canker_relation <- (date ?d) (< ?d 5)
date_canker_relation <- (date ?d) (> ?d 8)
date_canker_relation <- (canker_lesion dk_brown-blk)
date_seed_condition <- (date ?d) (< ?d 9)
date_seed_condition <- (seed abnorm)
leafspots-halos <- (leafspots-halos no_yellow_halos)
leafspots-halos <- (leafspots-halos yellow_halos)
date_condition <- (date ?d) (>= ?d 4) (<= ?d 6)
date_condition <- (date ?d) (>= ?d 8) (<= ?d 9)
date_precip_relation2 <- (date ?d) (>= ?d 4) (<= ?d 6) (high_precip)
date_precip_relation2 <- (date ?d) (>= ?d 8) (<= ?d 9) (precip >_norm)
date_precip_relation2 <- (date 7)
date_precip_relation2 <- (date 10)
date_temp_relation2 <- (date ?d) (< ?d 8) (temp norm)
date_temp_relation2 <- (date ?d) (> ?d 8) (temp norm)
date_temp_relation2 <- (date 8) (temp <_norm)
fruit_spot_condition <- (fruit_spots absent)
fruit_spot_condition <- (fruit_spots brown_w/blk_specks)
crop_hist_relation1 <- (crop_hist same_lst_sev_yrs)
crop_hist_relation1 <- (crop_hist same_lst_two_yrs)
crop_hist_relation2 <- (crop_hist_relation1)
crop_hist_relation2 <- (crop_hist same_lst_yr)
damage_date_condition <- (area_damaged scattered)
damage_date_condition <- (area_damaged low_areas)
damage_date_condition <- (area_damaged upper_areas)
damage_date_condition <- (date 6)
damage_date_condition <- (temp norm)
date_temp_relation3 <- (date ?d) (< ?d 6)
date_temp_relation3 <- (date ?d) (> ?d 6)
date_temp_relation3 <- (temp <_norm)
date_pods_condition <- (date ?d) (< ?d 9)
date_pods_condition <- (fruit_pods diseased)
pods_spots_condition <- (fruit_pods norm)
pods_spots_condition <- (fruit_pods few_present)
pods_spots_condition <- (fruit_pods dna)
pods_spots_condition <- (fruit_spots colored)
seed_color_condition <- (seed norm)
```

```
seed_color_condition <- (seed_discolor present)
date_spots_condition <- (date ?d) (< ?d 9)
date_spots_condition <- (date ?d) (> ?d 9)
date_spots_condition <- (fruit_spots colored)
```

B.4.2 The Initial Certainty-Factor Rule-Base

```
diaporthe_stem_canker <- 0.37-- (>= date 8)(<= date 9)
diaporthe_stem_canker <- 0.37-- (precip_ep)
diaporthe_stem_canker <- 0.37-- (stem_cankers above_sec_nde)
diaporthe_stem_canker <- 0.37-- (fruiting_bodies present)
diaporthe_stem_canker <- 0.37-- (fruit_pods norm)
diaporthe_stem_canker <- 0.37-- (>= temp norm)
diaporthe_stem_canker <- 0.37-- (canker_lesion brown)
diaporthe_stem_canker <- 0.035-- (crop_hist_er1)

charcoal_rot <- 0.23-- (>= date 7)(<= date 8)
charcoal_rot <- 0.23-- (<= precip norm)
charcoal_rot <- 0.23-- (>= temp norm)
charcoal_rot <- 0.23-- (plant_growth abnorm)
charcoal_rot <- 0.23-- (leaves abnorm)
charcoal_rot <- 0.23-- (stem abnorm)
charcoal_rot <- 0.23-- (sclerotia present)
charcoal_rot <- 0.23-- (roots rotted)
charcoal_rot <- 0.23-- (int_discolor black)
charcoal_rot <- 0.026-- (area_damaged upper_areas)
charcoal_rot <- 0.026-- (severity severe)
charcoal_rot <- 0.026-- (seed_size <_norm)
charcoal_rot <- 0.026-- (crop_hist_er2)

rhizoctonia_root_rot <- 0.23-- (>= date 5)(<= date 6))
rhizoctonia_root_rot <- 0.23-- (plant_stand <_norm)
rhizoctonia_root_rot <- 0.23-- (temp <_norm)
rhizoctonia_root_rot <- 0.23-- (precip <_norm)
rhizoctonia_root_rot <- 0.23-- (leaves abnorm)
rhizoctonia_root_rot <- 0.23-- (stem abnorm)
rhizoctonia_root_rot <- 0.23-- (canker_lesion brown)
rhizoctonia_root_rot <- 0.23-- (roots rotted)
rhizoctonia_root_rot <- 0.23-- (hail_vs_cankers)
rhizoctonia_root_rot <- 0.035-- (fruiting_bodies absent)
rhizoctonia_root_rot <- 0.035-- (external_decay firm_and_dry)
rhizoctonia_root_rot <- 0.035-- (mycelium absent)

phytophthora_root_rot <- 0.19-- (date_et)
phytophthora_root_rot <- 0.19-- (plant_stand <_norm)
phytophthora_root_rot <- 0.19-- (date_vs_precip1)
phytophthora_root_rot <- 0.19-- (date_vs_temp1)
phytophthora_root_rot <- 0.19-- (area_damaged low_areas)
phytophthora_root_rot <- 0.19-- (plant_growth abnorm)
phytophthora_root_rot <- 0.19-- (leaves abnorm)
phytophthora_root_rot <- 0.19-- (stem abnorm)
phytophthora_root_rot <- 0.19-- (stem_cankers above_soil)
phytophthora_root_rot <- 0.19-- (date_vs_cankers)
phytophthora_root_rot <- 0.19-- (roots rotted)
phytophthora_root_rot <- 0.10-- (>= crop_hist same_lst_two_yrs)

brown_stem_rot <- 0.28-- (>= date 7)(<= date 9)
brown_stem_rot <- 0.28-- (precip >_norm)
brown_stem_rot <- 0.28-- (<= temp norm)
```

```
brown_stem_rot <- 0.28-- (leaves abnorm)
brown_stem_rot <- 0.28-- (stem abnorm)
brown_stem_rot <- 0.28-- (int_discolor brown)
brown_stem_rot <- 0.28-- (lodging yes)
brown_stem_rot <- 0.05-- (seed_size <_norm)
brown_stem_rot <- 0.05-- (crop_hist_er3)

powdery_mildew <- 0.68-- (leaves abnorm)
powdery_mildew <- 0.68-- (leaf_mild upper_surf)
powdery_mildew <- 0.10-- (>= date 8)(<= date 9)

downy_mildew <-0.25-- (>= date 6)(<= date 8)
downy_mildew <-0.25-- (>= precip norm)
downy_mildew <-0.25-- (area_damaged whole_field)
downy_mildew <-0.25-- (leaves abnorm)
downy_mildew <-0.25-- (leafspots-halo no_yellow_halos)
downy_mildew <-0.25-- (leaf_mild lower_surf)
downy_mildew <-0.25-- (date_vs_seed)
downy_mildew <-0.25-- (mold_growth present)

brown_spot <- 0.44-- (leaves abnorm)
brown_spot <- 0.44-- (leafspots-halos)
brown_spot <- 0.44-- (leafspots-marg no_w-s_marg)
brown_spot <- 0.44-- (leafspot_size >_1/8)
brown_spot <- 0.05-- (dates1)
brown_spot <- 0.05-- (>= precip norm)

bacterial_blight <- 0.25-- (dates2)
bacterial_blight <- 0.25-- (date_vs_precip2)
bacterial_blight <- 0.25-- (date_vs_temp2)
bacterial_blight <- 0.25-- (leaves abnorm)
bacterial_blight <- 0.25-- (leafspots-halo yellow-halos)
bacterial_blight <- 0.25-- (leafspots-marg w-s_marg)
bacterial_blight <- 0.25-- (leafspot_size <_1/8)
bacterial_blight <- 0.25-- (leaf_shred present)

bacterial_pustule <- 0.28-- (>= date 6)(<= date 8)
bacterial_pustule <- 0.28-- (>= precip norm)
bacterial_pustule <- 0.28-- (leaves abnorm)
bacterial_pustule <- 0.28-- (leafspots-halo no_yellow_halos)
bacterial_pustule <- 0.28-- (leafspots-marg no_w-s_marg)
bacterial_pustule <- 0.28-- (leafspot_size <_1/8)
bacterial_pustule <- 0.28-- (leaf_shread present)
bacterial_pustule <- 0.10-- (>= crop_hist same_lst_yr)

purple_seed_stain <- 0.44-- (>= date 9)(<= date 10)
purple_seed_stain <- 0.44-- (seed abnorm)
purple_seed_stain <- 0.44-- (seed_discolor present)
purple_seed_stain <- 0.44-- (seed_size <_norm)
purple_seed_stain <- 0.035-- (>= date 8)(<= date 9)
purple_seed_stain <- 0.035-- (>= precip norm)
purple_seed_stain <- 0.035-- (leaves abnorm)

anthracnose <- 0.28-- (>= date 8)(<= date 10)
anthracnose <- 0.28-- (>= precip norm)
anthracnose <- 0.28-- (stem abnorm)
anthracnose <- 0.28-- (canker_lesion brown)
anthracnose <- 0.28-- (fruiting_bodies present)
anthracnose <- 0.28-- (date_vs_seed)
anthracnose <- 0.28-- (fruit_spots absent)
anthracnose <- 0.28-- (fruit_spots brn_w/blk_specks)
anthracnose <- 0.10-- (area_damaged whole_field)
```

```
phyllosticta_leaf_spot <- 0.28-- (>= date 4)(<= date 7)
phyllosticta_leaf_spot <- 0.28-- (>= precip norm)
phyllosticta_leaf_spot <- 0.28-- (leaves abnorm)
phyllosticta_leaf_spot <- 0.28-- (leafspots-halo no_yellow_halos)
phyllosticta_leaf_spot <- 0.28-- (leafspots-marg no_w-s_marg)
phyllosticta_leaf_spot <- 0.28-- (leafspot_size >_1/8)
phyllosticta_leaf_spot <- 0.28-- (leaf_shread present)
phyllosticta_leaf_spot <- 0.10-- (damage_date_vs_temp)
phyllosticta_leaf_spot <- 0.10-- (date_vs_temp3)

alternaria_leaf_spot <- 0.32-- (>= date 7)(<= date 10)
alternaria_leaf_spot <- 0.32-- (leaves abnorm)
alternaria_leaf_spot <- 0.32-- (leafspots-halo no_yellow_halos)
alternaria_leaf_spot <- 0.32-- (leafspots-marg no_w-s_marg)
alternaria_leaf_spot <- 0.32-- (leafspot_size >_1/8)
alternaria_leaf_spot <- 0.32-- (leaf_shread absent)
alternaria_leaf_spot <- 0.035-- (date_vs_pods)
alternaria_leaf_spot <- 0.035-- (pods_vs_spots)
alternaria_leaf_spot <- 0.035-- (seed_vs_color)

frog_eye_leaf_spot <- 0.32-- (>= date 7)(<= date 9)
frog_eye_leaf_spot <- 0.32-- (>= precip norm)
frog_eye_leaf_spot <- 0.32-- (leaves abnorm)
frog_eye_leaf_spot <- 0.32-- (leafspots-halo no_yellow_halos)
frog_eye_leaf_spot <- 0.32-- (leafspots-marg no_w-s_marg)
frog_eye_leaf_spot <- 0.32-- (leafspot_size >_1/8)
frog_eye_leaf_spot <- 0.026-- (date_vs_spots)
frog_eye_leaf_spot <- 0.026-- (stem_cankers above_sec_nde)
frog_eye_leaf_spot <- 0.026-- (canker_lesion tan)
frog_eye_leaf_spot <- 0.026-- (fruiting_bodies absent)

precip_ep <- 1.0-- (precip >_norm)
precip_ep <- 0.7-- (precip norm)

crop_hist_er1 <- 1.0-- (crop_hist same_lst_sev_yrs)
crop_hist_er1 <- 0.8-- (crop_hist same_lst_two_yrs)
crop_hist_er1 <- 0.7-- (crop_hist same_lst_yr)
crop_hist_er1 <- 0.2-- (crop_hist diff_lst_yr)

crop_hist_er2 <- 1.0-- (crop_hist same_lst_sev_yrs)
crop_hist_er2 <- 1.0-- (crop_hist same_lst_two_yrs)
crop_hist_er2 <- 0.6-- (crop_hist same_lst_yr)
crop_hist_er2 <- 0.2-- (crop_hist diff_lst_yr)

hail_vs_cankers <- 1.0-- (hail no)(stem_cankers below_soil)
hail_vs_cankers <- 1.0-- (hail no)(stem_cankers above_soil)
hail_vs_cankers <- 1.0-- (hail no)(stem_cankers above_sec_nde)

date_et <- 1.0-- (>= date 5)(<= date 7)
date_et <- 0.7-- (= date 4)
date_et <- 0.7-- (= date 8)

date_vs_precip1 <- 1.0-- (>= date 4)(<= date 6)(precip norm)
date_vs_precip1 <- 1.0-- (>= date 7)(<= date 8)(precip >_norm)
date_vs_precip1 <- 1.0-- (> date 8)

date_vs_temp1 <- 1.0-- (= date 4)(temp >_norm)
date_vs_temp1 <- 1.0-- (>= date 5)(<= date 8)(temp norm)
date_vs_temp1 <- 1.0-- (> date 4)

date_vs_cankers <- 1.0-- (>= date 5)(<= date 8)
```

```
                         (canker_lesion dk_brn-blk)
date_vs_cankers <- 1.0-- (< date 5)
date_vs_cankers <- 1.0-- (> date 8)

crop_hist_er3 <- 1.0-- (crop_hist same_lst_sev_yrs)
crop_hist_er3 <- 1.0-- (crop_hist same_lst_two_yrs)
crop_hist_er3 <- 0.5-- (crop_hist same_lst_yr)
crop_hist_er3 <- 0.1-- (crop_hist diff_lst_yr)

date_vs_seed <- 1.0-- (>= date 9)(<= date 10)(seed abnorm)
date_vs_seed <- 1.0-- (< date 9)

leafspots-halos <- 1.0-- (leafspots-halo yellow_halos)
leafspots-halos <- 1.0-- (leafspots-halo no_yellow_halos)

dates1 <- 1.0-- (= date 5)
dates1 <- 1.0-- (>= date 8)(<= date 9)
dates2 <- 1.0-- (>= date 4)(<= date 6)
dates2 <- 1.0-- (>= date 8)(<= date 9)

date_vs_precip2 <- 1.0-- (>= date 4)(<= date 6)(>= precip norm)
date_vs_precip2 <- 1.0-- (>= date 8)(<= date 9)(precip >_norm)
date_vs_precip2 <- 1.0-- (= date 7)
date_vs_precip2 <- 1.0-- (> date 9)

date_vs_temp2 <- 1.0-- (/= date 8)(temp norm)
date_vs_temp2 <- 1.0-- (= date 8)(temp <_norm)

damage_date_vs_temp <- 1.0-- (area_damaged whole_field)
                             (/= date 6)(temp norm)
damage_date_vs_temp <- 1.0-- (/= area_damaged whole_field)
damage_date_vs_temp <- 1.0-- (= date 6)

date_vs_temp3 <- 1.0-- (= date 6)(temp <_norm)
date_vs_temp3 <- 1.0-- (/= date 6)

date_vs_pods <- 1.0-- (>= date 9)(<= date 10)(fruit_pods diseased)
date_vs_pods <- 1.0-- (< date 9)

pods_vs_spots <- 1.0-- (fruit_pods diseased)(fruit_spots colored)
pods_vs_spots <- 1.0-- (/= fruit_pods diseased)

seed_vs_color <- 1.0-- (seed abnorm)(seed_discolor present)
seed_vs_color <- 1.0-- (seed norm)

date_vs_spots <- 1.0-- (date 9)(fruit_spots colored)
date_vs_spots <- 1.0-- (/= date 9)
```

B.4.3 A Revised Certainty-Factor Rule-Base

```
diaporthe_stem_canker <- 0.20-- (= fruit_spots dna)
diaporthe_stem_canker <- 0.17-- (>= date 8)(<= date 9)
diaporthe_stem_canker <- 0.24-- (precip_ep)
diaporthe_stem_canker <- 0.25-- (stem_cankers above_sec_nde)
diaporthe_stem_canker <- 0.24-- (fruiting_bodies present)
diaporthe_stem_canker <- 0.24-- (fruit_pods norm)
diaporthe_stem_canker <- 0.04-- (>= temp norm)
diaporthe_stem_canker <- -0.03-- (canker_lesion brown)

charcoal_rot <- 0.22-- (>= date 7)(<= date 8)
```

```
charcoal_rot <- 0.17-- (<= precip norm)
charcoal_rot <- 0.17-- (>= temp norm)
charcoal_rot <- 0.17-- (plant_growth abnorm)
charcoal_rot <- 0.23-- (leaves abnorm)
charcoal_rot <- 0.17-- (stem abnorm)
charcoal_rot <- 0.23-- (sclerotia present)
charcoal_rot <- 0.23-- (roots rotted)
charcoal_rot <- 0.23-- (int_discolor black)
charcoal_rot <- 0.026-- (area_damaged upper_areas)
charcoal_rot <- 0.026-- (severity severe)
charcoal_rot <- 0.026-- (seed_size <_norm)
charcoal_rot <- -0.02-- (crop_hist_er2)

rhizoctonia_root_rot <- 0.25-- (>= date 5)(<= date 6))
rhizoctonia_root_rot <- 0.24-- (plant_stand <_norm)
rhizoctonia_root_rot <- 0.29-- (temp <_norm)
rhizoctonia_root_rot <- 0.20-- (precip <_norm)
rhizoctonia_root_rot <- 0.16-- (leaves abnorm)
rhizoctonia_root_rot <- 0.24-- (stem abnorm)
rhizoctonia_root_rot <- 0.26-- (canker_lesion brown)
rhizoctonia_root_rot <- 0.23-- (roots rotted)
rhizoctonia_root_rot <- 0.18-- (hail_vs_cankers)
rhizoctonia_root_rot <- 0.084-- (fruiting_bodies absent)
rhizoctonia_root_rot <- 0.104-- (external_decay firm_and_dry)
rhizoctonia_root_rot <- -0.02-- (mycelium absent)

phytophthora_root_rot <- 0.19-- (date_et)
phytophthora_root_rot <- 0.20-- (plant_stand <_norm)
phytophthora_root_rot <- 0.14-- (date_vs_precip1)
phytophthora_root_rot <- 0.14-- (date_vs_temp1)
phytophthora_root_rot <- 0.17-- (area_damaged low_areas)
phytophthora_root_rot <- 0.17-- (plant_growth abnorm)
phytophthora_root_rot <- 0.17-- (leaves abnorm)
phytophthora_root_rot <- 0.17-- (stem abnorm)
phytophthora_root_rot <- 0.19-- (stem_cankers above_soil)
phytophthora_root_rot <- 0.14-- (date_vs_cankers)
phytophthora_root_rot <- 0.19-- (roots rotted)
phytophthora_root_rot <- 0.05-- (>= crop_hist same_lst_two_yrs)

brown_stem_rot <- 0.28-- (>= date 7)(<= date 9)
brown_stem_rot <- 0.06-- (precip >_norm)
brown_stem_rot <- 0.23-- (<= temp norm)
brown_stem_rot <- 0.24-- (leaves abnorm)
brown_stem_rot <- 0.38-- (stem abnorm)
brown_stem_rot <- 0.88-- (int_discolor brown)
brown_stem_rot <- 0.09-- (lodging yes)
brown_stem_rot <- 0.05-- (seed_size <_norm)
brown_stem_rot <- -0.069-- (crop_hist_er3)

powdery_mildew <- 0.10-- (<= precip norm)
powdery_mildew <- -0.10-- (precip >_norm)
powdery_mildew <- -0.10-- (precip_ep)
powdery_mildew <- 0.51-- (leaves abnorm)
powdery_mildew <- 0.70-- (leaf_mild upper_surf)
powdery_mildew <- 0.09-- (>= date 8)(<= date 9)

downy_mildew <- -0.10-- (seed_vs_color)
downy_mildew <- 0.10-- (seed abnorm)
downy_mildew <- 0.15-- (>= date 6)(<= date 8)
downy_mildew <- 0.23-- (>= precip norm)
downy_mildew <- 0.05-- (area_damaged whole_field)
downy_mildew <- 0.17-- (leaves abnorm)
```

```
downy_mildew <- 0.03-- (leafspots-halo no_yellow_halos)
downy_mildew <- 0.49-- (leaf_mild lower_surf)
downy_mildew <- 0.18-- (date_vs_seed)
downy_mildew <- 0.49-- (mold_growth present)

brown_spot <- 0.42-- (new-intermediate-term-for-br_s)
brown_spot <- 0.10-- (canker_lesion brown)
brown_spot <- -0.079-- (fruiting_bodies absent)
brown_spot <- -0.070-- (leaf_shread absent)
brown_spot <- -0.059-- (hail no)
brown_spot <- -0.026-- (<= precip norm)
brown_spot <- 0.057-- (seed_vs_color)
brown_spot <- -0.026-- (precip <_norm)
brown_spot <- 0.07-- (date_vs_temp2)
brown_spot <- -0.10-- (precip norm)
brown_spot <- 0.14-- (precip >_norm)
brown_spot <- 0.14-- (precip_ep)
brown_spot <- 0.13-- (leaf_shread present)
brown_spot <- -0.022-- (= severity minor)
brown_spot <- 0.057-- (fruit_spots absent)
brown_spot <- -0.022-- (< severity pot_severe)
brown_spot <- 0.40-- (fruiting_bodies present)
brown_spot <- 0.193-- (fruit_pods norm)
brown_spot <- 0.46-- (leaves abnorm)
brown_spot <- 0.46-- (leafspots-halos)
brown_spot <- 0.29-- (leafspots-marg no_w-s_marg)
brown_spot <- 0.59-- (leafspot_size >_1/8)
brown_spot <- 0.53-- (dates1)
brown_spot <- -0.89-- (>= precip norm)

new-intermediate-term-for-br_s <- 0.10-- (canker_lesion brown)
new-intermediate-term-for-br_s <- -0.10-- (stem_cankers above_sec_nde)
new-intermediate-term-for-br_s <- 0.10-- (date 9)
new-intermediate-term-for-br_s <- -0.10-- (fruiting_bodies absent)
new-intermediate-term-for-br_s <- -0.10-- (/= date 6)
new-intermediate-term-for-br_s <- 0.10-- (stem_cankers absent)
new-intermediate-term-for-br_s <- 0.10-- (fruiting_bodies present)
new-intermediate-term-for-br_s <- 0.10-- (area_damaged whole_filed)
                                         (/= date 6)(temp norm)
new-intermediate-term-for-br_s <- -0.28-- (date_vs_temp3)
new-intermediate-term-for-br_s <- 0.38-- (date 6)
new-intermediate-term-for-br_s <- -0.10--
                                         (/= area_damaged whole_field)
new-intermediate-term-for-br_s <- 0.91-- (area_damaged whole_field)

bacterial_blight <- 0.22-- (dates2)
bacterial_blight <- 0.26-- (date_vs_precip2)
bacterial_blight <- 0.21-- (date_vs_temp2)
bacterial_blight <- 0.21-- (leaves abnorm)
bacterial_blight <- 0.20-- (leafspots-halo yellow-halos)
bacterial_blight <- 0.25-- (leafspots-marg w-s_marg)
bacterial_blight <- 0.28-- (leafspot_size <_1/8)
bacterial_blight <- 0.23-- (leaf_shred present)

bacterial_pustule <- 0.10-- (leafspots-halo yellow_halos)
bacterial_pustule <- 0.25-- (>= date 6)(<= date 8)
bacterial_pustule <- 0.24-- (>= precip norm)
bacterial_pustule <- 0.24-- (leaves abnorm)
bacterial_pustule <- 0.15-- (leafspots-halo no_yellow_halos)
bacterial_pustule <- 0.38-- (leafspots-marg no_w-s_marg)
bacterial_pustule <- 0.32-- (leafspot_size <_1/8)
bacterial_pustule <- 0.21-- (leaf_shread present)
```

```
bacterial_pustule <- 0.045-- (>= crop_hist same_lst_yr)

purple_seed_stain <- -0.10-- (shriveling present)
purple_seed_stain <- 0.10-- (canker_lesion tan)
purple_seed_stain <- 0.10-- (stem_cankers absent)
purple_seed_stain <- -0.10-- (leafspot_size >_1/8)
purple_seed_stain <- 0.52-- (>= date 9)(<= date 10)
purple_seed_stain <- 0.48-- (seed abnorm)
purple_seed_stain <- 0.48-- (seed_discolor present)
purple_seed_stain <- 0.01-- (seed_size <_norm)
purple_seed_stain <- 0.214-- (>= date 8)(<= date 9)
purple_seed_stain <- 0.065-- (>= precip norm)
purple_seed_stain <- 0.065-- (leaves abnorm)

anthracnose <- -0.10-- (canker_lesion brown)
anthracnose <- 0.10-- (fruiting_bodies absent)
anthracnose <- 0.10-- (leaf_shread absent)
anthracnose <- 0.10-- (shriveling present)
anthracnose <- -0.10-- (canker_lesion tan)
anthracnose <- -0.10-- (leaf_shread present)
anthracnose <- -0.10-- (stem_cankers absent)
anthracnose <- -0.10-- (fruiting_bodies present)
anthracnose <- -0.10-- (fruit_spots dna)
anthracnose <- -0.10-- (fruit_pods norm)
anthracnose <- 0.28-- (>= date 8)(<= date 10)
anthracnose <- 0.28-- (>= precip norm)
anthracnose <- 0.28-- (stem abnorm)
anthracnose <- 0.28-- (canker_lesion brown)
anthracnose <- 0.28-- (fruiting_bodies present)
anthracnose <- 0.28-- (date_vs_seed)
anthracnose <- 0.28-- (fruit_spots absent)
anthracnose <- 0.28-- (fruit_spots brn_w/blk_specks)
anthracnose <- 0.10-- (area_damaged whole_field)

phyllosticta_leaf_spot <- 0.56 -- (new-intermediate-term-for-pls)
phyllosticta_leaf_spot <- 0.10-- (precip <_norm)
phyllosticta_leaf_spot <- -0.10-- (precip >_norm)
phyllosticta_leaf_spot <- -0.10-- (leafspots-halo yellow_halos)
phyllosticta_leaf_spot <- -0.10-- (precip_ep)
phyllosticta_leaf_spot <-  0.10-- (severity minor)
phyllosticta_leaf_spot <- -0.10-- (fruit_spots absent)
phyllosticta_leaf_spot <- 0.10-- (< severity pot_severe)
phyllosticta_leaf_spot <- 0.17-- (<= precip_norm)
phyllosticta_leaf_spot <- 0.28-- (>= date 4)(<= date 7)
phyllosticta_leaf_spot <- -0.10-- (>= precip norm)
phyllosticta_leaf_spot <- 0.25-- (leaves abnorm)
phyllosticta_leaf_spot <- 0.25-- (leafspots-halo no_yellow_halos)
phyllosticta_leaf_spot <- 0.28-- (leafspots-marg no_w-s_marg)
phyllosticta_leaf_spot <- 0.25-- (leafspot_size >_1/8)
phyllosticta_leaf_spot <- 0.27-- (leaf_shread present)
phyllosticta_leaf_spot <- 0.04-- (damage_date_vs_temp)
phyllosticta_leaf_spot <- 0.077-- (date_vs_temp3)

new_intermediate-term-for-pls <- 0.100-- (crop_hist same_lst_two_yrs)
new_intermediate-term-for-pls <- 0.100-- (<= precip norm)
new_intermediate-term-for-pls <- -0.100-- (precip >_norm)
new_intermediate-term-for-pls <- 0.100-- (date 7)
new_intermediate-term-for-pls <- -0.100-- (precip_ep)
new_intermediate-term-for-pls <- 0.100-- (area_damaged scattered))
new_intermediate-term-for-pls <- -0.100-- (>= precip norm)
new_intermediate-term-for-pls <- 0.100-- (precip <_norm)
new_intermediate-term-for-pls <- 0.100-- (hail no)
```

```
new_intermediate-term-for-pls <- -0.100-- (hail yes)

alternaria_leaf_spot <- -0.10-- (date_vs_temp2)
alternaria_leaf_spot <- -0.10-- (germination <_80%)
alternaria_leaf_spot <- -0.10-- (leaf_shread present)
alternaria_leaf_spot <- -0.10-- (crop_hist same_lst_sev_yrs)
alternaria_leaf_spot <- 0.10-- (fruit_pods norm)
alternaria_leaf_spot <- -0.10-- (fruiting_bodies present)
alternaria_leaf_spot <- 0.084-- (stem_cankers absent)
alternaria_leaf_spot <- 0.25-- (precip >_norm)
alternaria_leaf_spot <- -0.10-- (stem_cankers above_sec_nde)
alternaria_leaf_spot <- 0.37-- (>= date 7)(<= date 10)
alternaria_leaf_spot <- 0.34-- (leaves abnorm)
alternaria_leaf_spot <- 0.34-- (leafspots-halo no_yellow_halos)
alternaria_leaf_spot <- 0.32-- (leafspots-marg no_w-s_marg)
alternaria_leaf_spot <- 0.34-- (leafspot_size >_1/8)
alternaria_leaf_spot <- -0.011-- (leaf_shread absent)
alternaria_leaf_spot <- -0.011-- (pods_vs_spots)

frog_eye_leaf_spot <- 0.10-- (precip_norm)
frog_eye_leaf_spot <- 0.10-- (germination <_80%)
frog_eye_leaf_spot <- 0.10-- (crop_hist same_lst_sev_yrs)
frog_eye_leaf_spot <- -0.10-- (fruit_spots absent)
frog_eye_leaf_spot <- -0.10-- (fruit_pods norm)
frog_eye_leaf_spot <- -0.10-- (fruiting_bodies present)
frog_eye_leaf_spot <- -0.10-- (<= precip norm)
frog_eye_leaf_spot <- -0.10-- (stem_cankers absent)
frog_eye_leaf_spot <- 0.10-- (leafspot_size >_1/8)
frog_eye_leaf_spot <- 0.32-- (>= date 7)(<= date 9)
frog_eye_leaf_spot <- 0.32-- (>= precip norm)
frog_eye_leaf_spot <- 0.32-- (leaves abnorm)
frog_eye_leaf_spot <- 0.32-- (leafspots-halo no_yellow_halos)
frog_eye_leaf_spot <- 0.32-- (leafspots-marg no_w-s_marg)
frog_eye_leaf_spot <- 0.32-- (leafspot_size >_1/8)
frog_eye_leaf_spot <- 0.026-- (date_vs_spots)
frog_eye_leaf_spot <- 0.13-- (stem_cankers above_sec_nde)
frog_eye_leaf_spot <- 0.026-- (canker_lesion tan)
frog_eye_leaf_spot <- 0.026-- (fruiting_bodies absent)

precip_ep <- 1.0-- (precip >_norm)
precip_ep <- 0.7-- (precip norm)

crop_hist_er2 <- 1.0-- (crop_hist same_lst_sev_yrs)
crop_hist_er2 <- 1.0-- (crop_hist same_lst_two_yrs)
crop_hist_er2 <- 0.6-- (crop_hist same_lst_yr)
crop_hist_er2 <- 0.2-- (crop_hist diff_lst_yr)

hail_vs_cankers <- 1.0-- (hail no)(stem_cankers below_soil)
hail_vs_cankers <- 1.0-- (hail no)(stem_cankers above_soil)
hail_vs_cankers <- 1.0-- (hail no)(stem_cankers above_sec_nde)

date_et <- 1.0-- (>= date 5)(<= date 7)
date_et <- 0.7-- (= date 4)
date_et <- 0.7-- (= date 8)

date_vs_precip1 <- 1.0-- (>= date 4)(<= date 6)(precip norm)
date_vs_precip1 <- 1.0-- (>= date 7)(<= date 8)(precip >_norm)
date_vs_precip1 <- 1.0-- (> date 8)

date_vs_temp1 <- 1.0-- (= date 4)(temp >_norm)
date_vs_temp1 <- 1.0-- (>= date 5)(<= date 8)(temp norm)
date_vs_temp1 <- 1.0-- (> date 4)
```

```
date_vs_cankers <- 1.0-- (>= date 5)(<= date 8)
                         (canker_lesion dk_brn-blk)
date_vs_cankers <- 1.0-- (< date 5)
date_vs_cankers <- 1.0-- (> date 8)

crop_hist_er3 <- 1.0-- (crop_hist same_lst_sev_yrs)
crop_hist_er3 <- 1.0-- (crop_hist same_lst_two_yrs)
crop_hist_er3 <- 0.5-- (crop_hist same_lst_yr)
crop_hist_er3 <- 0.1-- (crop_hist diff_lst_yr)

date_vs_seed <- 1.0-- (>= date 9)(<= date 10)(seed abnorm)
date_vs_seed <- 1.0-- (< date 9)

leafspots-halos <- 1.0-- (leafspots-halo yellow_halos)
leafspots-halos <- 1.0-- (leafspots-halo no_yellow_halos)

dates1 <- 1.0-- (= date 5)
dates1 <- 1.0-- (>= date 8)(<= date 9)

dates2 <- 1.0-- (>= date 4)(<= date 6)
dates2 <- 1.0-- (>= date 8)(<= date 9)

date_vs_precip2 <- 1.0-- (>= date 4)(<= date 6)(>= precip norm)
date_vs_precip2 <- 1.0-- (>= date 8)(<= date 9)(precip >_norm)
date_vs_precip2 <- 1.0-- (= date 7)
date_vs_precip2 <- 1.0-- (> date 9)

date_vs_temp2 <- 1.0-- (/= date 8)(temp norm)
date_vs_temp2 <- 1.0-- (= date 8)(temp <_norm)

damage_date_vs_temp <- 1.0-- (area_damaged whole_field)(/= date 6)
                             (temp norm)
damage_date_vs_temp <- 1.0-- (/= area_damaged whole_field)
damage_date_vs_temp <- 1.0-- (= date 6)

date_vs_temp3 <- 1.0-- (= date 6)(temp <_norm)
date_vs_temp3 <- 1.0-- (/= date 6)

pods_vs_spots <- 1.0-- (fruit_pods diseased)(fruit_spots colored)
pods_vs_spots <- 1.0-- (/= fruit_pods diseased)

seed_vs_color <- 1.0-- (seed abnorm)(seed_discolor present)
seed_vs_color <- 1.0-- (seed norm)

date_vs_spots <- 1.0-- (date 9)(fruit_spots colored)
date_vs_spots <- 1.0-- (/= date 9)
```

Bibliography

Alberts, B. (1988). *Mapping and Sequencing the Human Genome.* National Academy Press, Washington, D.C.

Baffes, P., & Mooney, R. J. (1992). Using theory revision to model students and acquire stereotypical errors. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pp. 617–622 Bloomington, IN.

Baffes, P., & Mooney, R. (1993a). Symbolic revision of theories with M-of-N rules. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 1135–1140 Chambery, France.

Baffes, P. T., & Mooney, R. J. (1993b). Extending theory refinement to M-of-N rules. *Informatica, 17*, 387–397.

Berenji, H. (1990). Refinement of approximate reasoning-based controllers by reinforcement learning. In *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 475–479 Evanston, IL.

Buchanan, G., & Shortliffe, E. (Eds.). (1984). *Rule-Based Expert Systems:The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley Publishing Co., Reading, MA.

Buntine, W. (1991). Theory refinement on Bayesian networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 52–60.

Cain, T. (1991). The DUCTOR: A theory revision system for propositional domains. In *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 485–489 Evanston, IL.

Caruana, R. (1989). The automatic training of rule bases that use numerical uncertainty representations. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 347–356.

Clark, K. (1978). Negation as failure. In Gallaire, H., & Minker, J. (Eds.), *Logic and Data Bases.* Plenum Press, New York, NY.

Cohen, W. (1992). Compiling prior knowledge into an explicit bias. In *Proceedings of the Ninth International Conference on Machine Learning*, pp. 102–110 Aberdeen, Scotland.

Connolly, D. (1993). Constructing hidden variables in bayesian networks via conceptual clustering. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 65–72 Amherst, MA.

Cooper, G. (1990). Computational complexity of probabilistic inference using Bayesian belief networks (research note). *Artificial Intelligence*, *42*, 393–405.

Cooper, G. G., & Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, *9*, 309–347.

Davis, L. (1995). Genetic algorithms and trading. In *Proceedings of the Sixth Annual Conference on Advanced Technologies for Trading and Asset Management*, pp. 79–88 New York, New York.

DeJong, G. F., & Mooney, R. J. (1986). Explanation-based learning: An alternative view. *Machine Learning*, *1*(2), 145–176. Reprinted in *Readings in Machine Learning*, J. W. Shavlik and T. G. Dietterich (eds.), Morgan Kaufman, San Mateo, CA, 1990.

Fahlman, S., & Lebiere, C. (1989). The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems, Vol. 2*, pp. 524–532 San Mateo, CA. Morgan Kaufmann.

Feigenbaum, E., Buchanan, B., & Lederberg, J. (1971). On generality and problem solving: A case study involving the dendral program. *Machine Intelligence*, *6*, 165–190.

Feigenbaum, E., & McCorduck, P. (1983). *The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World.* Addison-Wesley Publishing Co., Reading, MA.

Feigenbaum, E., McCorduck, P., & Nii, H. (1988). *The Rise of the Expert Company.* Times BooksAcademic Press, New York, NY.

Frank-Kamenetskii, M. (1993). *Unraveling DNA.* VCH Publishers, Inc., New York, N.Y.

Frean, M. (1990). The Upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, *2*, 198–209.

Fu, L.-M. (1989). Integration of neural heuristics into knowledge-based inference. *Connection Science*, *1*(3), 325–339.

Fu, L., & Lacher, C. (Eds.). (1994). *Proceedings of the International Symposium on Integrating Knowledge and Neural Heuristics, 1994*, Pensacola, FL.

Gallant, S. (1988). Connectionist expert systems. *Communications of the Association for Computing Machinery*, *31*, 152–169.

Ginsberg, A. (1988). Theory revision via prior operationalization. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 590–595 St. Paul, MN.

Ginsberg, A. (1990). Theory reduction, theory revision, and retranslation. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 777–782 Detroit, MI.

Heckerman, D. (1986). Probabilistic interpretations for Mycin's certainty factors. In Kanal, L. N., & Lemmer, J. F. (Eds.), *Uncertainty in Artificial Intelligence*, pp. 167–196. North Holland, Amsterdam.

Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pp. 1–12 Amherst, MA.

Kandel, A., & Langholz, G. (Eds.). (1992). *Hybrid Architectures for Intelligent Systems*. CRC Press, Inc., Boca Raton, FL.

Kohavi, R., & John, G. (1995). Automatic parameter setting by minimizing estimated error. In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 304–312 San Francisco, CA. Morgan Kaufman.

Koppel, M., Feldman, R., & Segre, A. M. (1994a). Bias-driven revision of logical domain theories. *Journal of Artificial Intelligence Research*, *1*, 1–50.

Koppel, M., Segre, A., & Feldman, R. (1994b). Getting the most from flawed theories. In *Proceedings of the Eleventh International Conference on Machine Learning*, pp. 139–147 New Brunswick, NJ.

Kornberg, A. (1974). *DNA Synthesis*. W.H. Freeman and Company, San Francisco, CA.

Lacher, R. (1992). Node error assignment in expert networks. In Kandel, A., & Langholz, G. (Eds.), *Hybrid Architectures for Intelligent Systems*, pp. 29–48. CRC Press, Inc., Boca Raton, FL.

Lacher, R., Hruska, S., & Kuncicky, D. (1992). Back-propagation learning in expert networks. *IEEE Transaction on Neural Networks*, *3*(1), 62–72.

Langley, P., & Simon, H. A. (1995). Applications of machine learning and rule induction. *Communications of the Association for Computing Machinery*, *38*(11), 55–64.

Ling, X., & Valtorta, M. (1991). Revision of reduced theories. In *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 519–523 Evanston, IL.

Ma, Y., & Wilkins, D. C. (1991). Improving the performance of inconsistent knowledge bases via combined optimization method. In *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 23–27 Evanston, IL.

Merz, C., Murphy, P. M., & Aha, D. W. (1996). Repository of machine learning databases `http://www.ics.uci.edu/~mlearn/mlrepository.html`. Department of Information and Computer Science, University of California, Irvine, CA.

Mezard, M., & Nadal, J. (1989). Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics*, *A22*(12), 2191–2203.

Michalksi, R., Mozetic, I., Hong, J., & Lavrac, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp. 1041–1045 Philadelphia, PA.

Michalski, R. S., & Chilausky, S. (1980). Learning by being told and learning from examples: An experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *Journal of Policy Analysis and Information Systems*, *4*(2), 126–161.

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, *1*(1), 47–80.

Mooney, R. J., & Ourston, D. (1991). A multistrategy approach to theory refinement. In *Proceedings of the First International Workshop on Multistrategy Learning*, pp. 115–130 Harper's Ferry, W.Va.

Musick, R. C. (1994). *Belief Network Induction*. Ph.D. thesis, University of California at Berkeley.

Noordewier, M. O., Towell, G. G., & Shavlik, J. W. (1991). Training knowledge-based neural networks to recognize genes in DNA sequences. In *Advances in Neural Information Processing Systems*, Vol. 3 San Mateo, CA. Morgan Kaufman.

O'Neill, M., & Chiafari, F. (1989). Escherichia coli promoters. *Journal of Biological Chemistry*, *264*, 5531–5534.

Opitz, D. W., & Shavlik, J. W. (1993). Heuristically expanding knowledge-based neural networks. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 512–517 Chamberry, France.

Ortega, J. (1995). On the informativeness of the dna promoter sequences domain theory. *Journal of Artificial Intelligence Research*, *2*, 361–367.

Ourston, D. (1991). *Using Explanation-Based and Empirical Methods in Theory Revision*. Ph.D. thesis, University of Texas, Austin, TX. Also appears as Artificial Intelligence Laboratory Technical Report AI 91-164.

Ourston, D., & Mooney, R. (1990). Changing the rules: A comprehensive approach to theory refinement. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 815–820 Detroit, MI.

Ourston, D., & Mooney, R. J. (1994). Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, *66*, 311–344.

Pazzani, M., & Kibler, D. (1992). The utility of background knowledge in inductive learning. *Machine Learning*, *9*, 57–94.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, Inc., San Mateo,CA.

Portugal, F., & Cohen, J. (1977). *A Century of DNA: A History of the Discovery of the Structure and Function of the Genetic Substance*. MIT Press, Cambridge, MA.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo,CA.

Rada, R. (1985). Gradualness facilitates knowledge refinement. *IEEE transactions on Pattern Analysis and Machine Intelligence*, *7*(5).

Ramachandran, S. (1995). Refinement of Bayesian networks by combining connectionist and symbolic techniques. cs.utexas.edu FTP archive.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, *14*, 465–471.

Rosenfield, I., Ziff, E., & van Loon, B. (1983). *DNA For Beginners.* Writers and Readers Publishing Cooperative Ltd., London.

Rumelhart, D. E., Hinton, G. E., & Williams, J. R. (1986). Learning internal representations by error propagation. In Rumelhart, D. E., & McClelland, J. L. (Eds.), *Parallel Distributed Processing, Vol. I*, pp. 318–362. MIT Press, Cambridge, MA.

Samuel, A. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, pp. 211–229.

Schwalb, E. (1993). Compiling Bayesian networks into neural networks. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 291–297 Amherst, MA.

Shafer, G. (1976). *A Mathematical Theory of Evidence.* Princeton University Press, Princeton, NJ.

Shafer, G., & Pearl, J. (Eds.). (1990). *Readings in Uncertain Reasoning.* Morgan Kaufmann, Inc., San Mateo,CA.

Shavlik, J. W., Mooney, R. J., & Towell, G. G. (1991). Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, *6*, 111–143. Reprinted in *Readings in Knowledge Acquisition and Learning*, B. G. Buchanan and D. C. Wilkins (eds.), Morgan Kaufman, San Mateo, CA, 1993.

Shortliffe, E. (1976). *Computer-Based Medical Consultations: MYCIN.* American Elsevier Publishing Company, INC., New York, NY.

Shortliffe, E., & Buchanan, B. (1975). A model of inexact reasoning in medicine. *Mathematical Biosciences*, *23*, 351–379.

Swartout, W. (1981). Explaining and justifying in expert consulting programs. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pp. 203–208 Vancouver, BC.

Thompson, K., Langley, P., & Iba, W. (1991). Using background knowledge in concept formation. In *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 554–558 Evanston, IL.

Towell, G., & Shavlik, J. (1991). Refining symbolic knowledge using neural networks. In *Proceedings of the First International Workshop on Multistrategy Learning*, pp. 257–272 Harper's Ferry, W.Va.

Towell, G., & Shavlik, J. (1992). Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules. In Lippmann, R., Moody, J., & Touretzky, D. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 4. Morgan Kaufmann.

Towell, G. G., Shavlik, J. W., & Noordewier, M. O. (1990). Refinement of approximate domain theories by knowledge-based artificial neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 861–866 Boston, MA.

Towell, G., & Shavlik, J. (1992). Using symbolic learning to improve knowledge-based neural networks. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 177–182 San Jose, CA.

Towell, G., & Shavlik, J. (1994). Knowledge-based artificial neural networks. *Artificial Intelligence*, *69*.

Valtorta, M. (1988). Some results on the complexity of knowledge-base refinement. In *Proceedings of the Sixth International Workshop on Machine Learning*, pp. 326–331 Ithaca, NY.

Valtorta, M. (1990). More results on the complexity of knowledge-base refinement:belief networks. In *Proceedings of the Seventh International Conference on Machine Learning*, pp. 419–424 Austin, TX.

von Heijne, G. (1987). *Sequence Analysis in Molecular Biology: Treasure Trove or Trivial Pursuit.* Academic Press, San Diego, CA.

Wade, N. (1995). Rapid gains are reported on genome. *The New York Times*, *Section A*, 13.

Watson, J., Roberts, H., Steitz, J., & Weiner, A. (1987). *The Molecular Biology of the Gene.* Benjamin-Cummings, Menlo, Park, CA.

Yu, Y.-H., & Simmons, R. (1990). Descending epsilon in back-propagation: A technique for better generalization. Tech. rep. AI90-130, Artificial Intelligence Laboratory, The University of Texas at Austin, Austin, TX.

Zadeh, L. (1965). Fuzzy sets. *Information and Control*, *8*, 338–353.