UNIVERSITY OF CALIFORNIA
IRVINE

Learning Probabilistic Relational Concept Descriptions

DISSERTATION

submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in Information and Computer Science

by

Kamal Mahmood Ali

Dissertation Committee:

Professor Michael J. Pazzani, Chair

Professor Dennis F. Kibler

Professor Dennis Volper

1996

The dissertation of Kamal Mahmood Ali is approved,
and is acceptable in quality and form
for publication on microfilm:

Committee Chair

University of California, Irvine

1996

Dedication

I would like to dedicate this thesis to my wife, Mary, and to my parents.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to acknowledge the tireless, prompt help of my advisor, Michael Pazzani and the comments of the other members of the dissertation committee: Dennis Kibler and Dennis Volper.

I would also like to acknowledge the frequent and useful feedback on research matters that I received from the other students at Irvine: Cliff Brunk, Pedro Domingos, Pat Murphy, Jim Wogulis. And thanks to David Aha for helpful comments early in my long voyage. And special thanks to Jim Wogulis for help in preparing this dissertation.

A special thanks to Professor Bob Newcombe for his inspiring courses on multivariate statistics which helped me a lot in finishing up the thesis.

Also, thanks to David Schulenberg and Karl Schwamb for help with LISP, Emacs and LaTeX; Thanks also to the support staff of John Romine, David Harnick-Shapiro, Lily Wang and Patrick Murphy for quite a number of useful tips over the years.

I would also like to acknowledge my roommate, Manish Singh, who provided a great model of a hard-working graduate student.

Curriculum Vitae

Kamal Mahmood Ali

ACADEMIC QUALIFICATIONS:

1996: Ph.D (Jan. 96) at University of California, Irvine, in Machine Learning: "Learning probabilisitic relational concept descriptions."

1990: M.S. in Computer Science from University of California, Irvine.

1986: B.S.(First Class Honors) in Computer Science from Sydney University, Australia.

PROFESSIONAL WORK EXPERIENCE:

1993 AI CONSULTANT at Veterans Medical Center, Long Beach, CA. Developed expert system - predicts site of spinal injury.

1989 - 95 RESEARCH ASSISTANT and TEACHING ASSISTANT.

1987 - 88 AI CONSULTANT to Dept. of Remote Sensing, CSIRO. Developed an expert system (C/UNIX) - identifies minerals from reflectance spectra obtained via remote sensing.

1987 PROGRAMMER at Logica Inc., North Sydney, NSW, Australia. Wrote bids (using COCOMO) for software system tenders.

1986 AI CONSULTANT to Newcastle Lead Smelter. Developed a Expert System in C and FORTRAN under UNIX. Used to diagnose faults in a lead production plant.

1985 PROGRAMMER at Software Solutions, St. Leonards, NSW, Aust. Developed technical assistant system in 4GL: INFO.

JOURNALS:

Ali K. and Pazzani M. Error Reduction through Learning Multiple Descriptions. Submitted to *Machine Learning Journal*.

Ali K. and Pazzani M. (1995) (Invited for submission) HYDRA-MM: Learning Multiple Descriptions to Improve Classification Accuracy. *International Journal of Artificial Intelligence Tools 4,1&2*, pp. 115-133. World Scientific.

Ali, K., Lister, R., Horsfall C. (1989). Towards knowledge-based identification of mineral mixtures from reflectance spectra in *Knowledge-Based Systems 2,1*, pp. 5-13. Butterworth Scientific.

BOOK CHAPTERS:

Ali K. and Pazzani M. (1995). Learning Multiple Relational Rule-based Models. In Fisher, D. and Lenz H. *Learning from Data: Artificial Intelligence and Statistics, Vol. 5.* Springer-Verlag.

Ali K. and Pazzani M. (1992). Reducing the small disjuncts problem by learning probabilistic concept descriptions. In Petsche, T., Hanson, S.J. & Shavlik, J. (Eds), *Computational Learning Theory and Natural Learning Systems, Vol. 3.* Cambridge, Massachusetts. MIT Press.

Hirschberg D., Pazzani M., Ali K. (1991). Average Case Analysis of k-CNF and k-DNF Learning Algorithms. In Hanson, S.J., Petsche, T., Kearns, M., & Rivest, R.L. (Eds), *Computational Learning Theory and Natural Learning Systems, Vol. 2.* Cambridge, Massachusetts. MIT Press.

CONFERENCES PUBLICATIONS:

Ali K., Brunk C. and Pazzani M. (1994). On Learning Multiple Descriptions of a Concept. In *Proceedings of the Sixth International Conference on Tools with Artificial Intelligence.* New Orleans, LA: IEEE Press.

Pazzani M., Merz C., Murphy P., Ali K., Hume T. and Brunk C. (1994). Reducing Misclassification Costs. In *Machine Learning: Proceedings of the Eleventh International Conference.* New Brunswick, NJ. Morgan Kaufmann.

Ali K. and Pazzani M. (1993). HYDRA: A Noise-tolerant Relational Concept Learning Algorithm. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.* Chambery, France.

Lister R., Ali, K., Buda, R., Horsfall, C., Buntine W. (1987). GOLD - An expert system for mineral identification from reflectance spectra. In *Proceedings of the Australian Joint Artificial Intelligence Conference.*

WORKSHOPS:

Ali K., Brunk C. and Pazzani M. (1995). Learning Multiple Relational Rule-based Models. In *Preliminary Papers of the 5th International Workshop on Artificial Intelligence and Statistics.* Fort Lauderdale, FL.

Pazzani M., Murphy P., Ali K. and Schulenberg D. (1994). Trading off coverage for accuracy in forecasts: Applications to clinical data analysis. In *Proceedings of the 1994 AAAI Symposium on AI in Medicine* (pp 106-110). Stanford, CA.

Ali, K. Augmenting Domain Theory for Explanation-Based Generalization. (1989). In *Proceedings of the Sixth International Workshop on Machine Learning* , Ithaca, NY.

Abstract of the Dissertation

Learning Probabilistic, Relational Concept Descriptions

by

Kamal Mahmood Ali

Doctor of Philosophy in Information and Computer Science

University of California, Irvine, 1996

Professor Michael J. Pazzani, Chair

This dissertation presents methods for increasing the accuracy of probabilistic classification rules learned from noisy, relational data. It addresses the problem of learning probabilistic rules in noisy, "real-world" data sets, the problem of "small disjuncts" in which rules that apply to rare subclasses have high error rates, and the problems that arise in domains in which the learning algorithm is forced to pick from many rules that *appear* to be equally good.

It is shown that learning a class description for each class in the data - the one-per-class approach - and attaching probabilistic estimates to the learned rules allows accurate classifications to be made on real-world data sets. The thesis presents the system HYDRA which implements this approach. It is shown that the resulting classifications are often more accurate than those made by three major methods for learning from noisy, relational data. Furthermore, the learned rules are relational and so are more expressive than the attribute-value rules learned by most induction systems.

Several results are also presented in the arena of multiple models. The multiple models approach in relevant to the problem of making accurate classifications in "real-world" domains since it facilitates evidence combination which is needed to accurately learn on such domains.

The most important result of the multiple models research is that the amount of error reduction afforded by the multiple models approach is linearly correlated with the degree to which the individual models make errors in an uncorrelated

manner. It is shown that it is possible to learn models that make less correlated errors in domains in which there are many gain ties. The third major result of the research on multiple models is the realization that models should be learned that make errors in a negatively-correlated manner rather than those that make errors in an uncorrelated manner.

Finally, results are presented on the small-disjuncts problem in which rules that apply to rare subclasses have high error rates. It is shown that the one-per-class approach reduces error rates for such rare rules while not sacrificing the error rates of the other rules.

Chapter 1
Introduction

Learning how to classify objects to one of a prespecified set of classes is a facet of intelligence that has been extensively studied, both in humans and in computers. Classification and the accompanying task of learning class descriptions allows us to distill what is important about being a member of a class. This distillation is of enormous use because it helps direct the focus of an intelligent agent, no matter whether that agent is a human being or a machine. For instance, people generally know that if you want to decide whether or not the animal behind the trees is a leopard, you should look for spots rather than focus on the color of its eyes. Being able to differentiate leopards from non-leopards and being able to *learn* to differentiate is a very useful ability - especially so in time-critical situations!

Another benefit of learning class descriptions that are generalizations about what it means to be a member of a class is that such generalizations allow one to classify novel objects to a known class or category. One can only claim to truly understand what it means to be a member of a class if one can classify a novel object to its correct class. Learning class descriptions gives us precisely this ability: the ability to generalize and classify novel objects. Rote memorization is usually not sufficient to constitute understanding. Another benefit of learning class descriptions is that it allows an intelligent agent (human or machine) to make inferences that are true for all members of the class, or come to conclusions that are true for most of the members of the class. This organization allows one to reason about all members of a class rather than reasoning from scratch about each member.

There are two basic kinds of classification tasks that will be considered in this thesis. The first, called the *two-class classification task* is to determine whether or not an object is a member of a given class. So, for example, the class may be to decide whether an object is a member of the class of leopards. In such tasks, the second class is derived from the first class. So, for this illustration, the second class would be the class of all things that are not leopards. The second type of task, the *multi-class classification task*, is a generalization of the two-class classification task. In this task the classifier is given a set of classes and an object and the goal is to classify the object to one of the given classes. The classes are assumed to be disjoint so that any given object cannot be a member of more than one class. So, the set of classes, {human,male,female}, for example, is not allowed because an object can simultaneously be a member of more than one of those classes.

In order to classify an object, a mechanism for describing the object is needed. Say, for instance, that it was necessary to make a record of a medical patient so that a specialist doctor at another hospital could look at the record to make his or her diagnosis. One would need to make a decision about how to describe the patient in sufficient detail so as to allow the specialist to make the diagnosis. In artificial intelligence, one approach that is frequently used to describe objects (such as patients) is to select a set of attributes in anticipation of having to use the same set of attributes to describe any given patient. Therefore, the decision about which set of attributes to use has to be made at the beginning before any objects are actually described. Having made the decision, all objects of all classes have to use the same set of attributes, although the actual values of an attribute typically found for objects of one class might be quite different from the set of values for objects of another class.

To make this more concrete, consider the classification task of diagnosing a medical patient into one of the three classes: Alzheimer's dementia, other dementia, and normal. One attribute that may be chosen to describe a patient could measure the number of strokes that the patient has experienced. Another may measure the age at which the patient began to experience significant memory loss. As is clear from this example, attributes need not only refer to perceptual information: the attribute that measures the age at which the patient began to experience significant memory loss, for example, may need a complex decision-making process.

Attributes are used by computers to represent objects using the following method. Consider representing a patient using the following sequence of attributes:

`(Number-of-strokes, Sex)`

In order to store values for these attributes for a patient in a computer, an ordering of the attributes must be decided. Let us say then, that we decide to order the attributes with the `Sex` attribute first, and the `Number-of-strokes` attribute next. Consider a male patient who has had one stroke. Given the chosen order of attributes, this patient would be represented in a computer by the following list: (male, 1). Such a computer representation of an object is called an *example*. If the set of attributes chosen for the classification task has been chosen well, all relevant information about the object can be captured by the example representation. So, returning to the specialist doctor illustration, the hospital would only have to communicate the chosen order of attributes once at the beginning (say, at the time of installation of the computer system) to the specialist and from then on, each patient's record could be compactly communicated as a list of numbers.

1.1 The learning protocol

In machine learning, objects that are encoded as examples to be used during learning are called *training examples*. The set of training examples is called the *training set*. Each training example is provided to the learning program together with the class of that example. The class that the object belongs to is called the *class label* of that example. In the Alzheimer task, patients are encoded as examples and for each patient, the diagnosis of the patient is stored as the class label. For this illustration, assume that patients can be diagnosed with exactly one of the following conditions: {`Alzheimer's, Other-dementia, Normal`}. If a male patient with one stroke is diagnosed as belonging to class `Alzheimer`, we will write the training example and class label corresponding to this information as (Male,1;Alzheimer's). In order to store such information in a computer, the computer must store each training example and also its associated class label.

The training task faced by the computer can be illustrated by analogy with the situation faced by a medical student. The training set for a medical student would be the patients (from all three classes) that he or she sees during medical training. However the situation faced by the computer is not a faithful model of the situation faced by the medical student since the computer has full access to values for all the attributes of each patient (the example) and access to the diagnosis (the class of the example) of each patient.[1] This learning situation would only have a real-world counterpart if the diagnosis for each patient were somehow available to the student. For instance, the diagnosis might be provided by a senior doctor.

Another way in which the model of learning in the computer differs from that faced by the medical student is that all the training examples are available to the program at the commencement of learning (this type of learning is called *batch learning*). This is akin to the medical student having immediate access to all the records of all the patients she or he will see during the medical training period. Although this type of learning is psychologically implausible, it has the distinct advantage that the learning program is not unduly influenced by the examples seen at the beginning or ending of a learning episode. Research (e.g. Nisbett & Ross (1980)) has shown that humans put undue weight on the first few examples they see in learning a new concept. Another advantage of batch learning programs is that they yield more accurate classifications as compared to most non-batch learning programs because they are not affected by the order of presentation of training examples. It also turns out that batch algorithms are easier to design than equally-accurate non-batch algorithms. However, batch learning programs also have disadvantages. Humans update their class descriptions after seeing each

[1] Machine learning programs do not require that the value of each attribute be known but they do require access to all the values even if some of the values are tagged as unknown. They also do not require that the class label of each training example be correct. They only require that most of them be correct.

Table 1.1. Some simple rules.

1. If the patient has memory loss and s/he has had strokes then the patient belongs to class `Other-dementia`.
2. If the patient has memory loss and depression then the patient belongs to class `Other-dementia`.

example. The advantage of this style of learning (called *incremental learning*) is that it allows classification and learning to be interleaved. Batch learning programs on the other hand, learn once and are then ready to hand their class descriptions over to the classifier program. If more examples become available, the batch learning program cannot take off from where it left before: it has to restart learning from scratch. However, this is not such a disadvantage for computers unless they are in an active environment where interleaved classification and learning is necessary. Usually, computer learning occurs in an environment in which it is acceptable to learn again from scratch (learning between midnight and 6 a.m. for example). For learning programs, incremental ability is not as important as ensuring the accuracy of classifications is maximized.

In batch learning, the learning program searches for patterns which differentiate *all* examples of one class from all examples of all the other classes. As all the training examples are available at the beginning of learning, the learning program can choose to process the training examples in any desired order. The learning program can take as long as it needs, examining each example several times if necessary, before producing class descriptions that can be used by a classifier program. By not being subject to any particular example ordering and by re-examining examples many times, batch learning usually allows the classifier to make more accurate classifications.

The two main kinds of programs that will be discussed in this thesis are learning programs and classifier programs. The learning programs that will be considered in this thesis usually come in two varieties. The first variety of learning program learns a description for each class (the *one-per-class* approach; (Dietterich & Bakiri, 1995)), the second variety learns a description that is used to differentiate between all the classes (the *multi-class* approach). An example of a one-per-class description consisting of two rules for the class `Other-dementia` is shown in Table 1.1.

An example of a multi-class differentiating description for the three-class classification task consisting of the classes {Alzheimer's, Other-dementia, Normal} is:

If	the patient does not have any memory loss		
then	the patient belongs to class `Normal`		
else	if	the patient has had strokes or depression	
	then	the patient belongs to class `Other-dementia`	
	else	the patient belongs to class `Alzheimer's`	

The rules just described are non-probabilistic since the first rule in Table 1.1, for example, encodes the knowledge that if the patient has had strokes and memory loss then the patient *definitely* belongs to class `Other-dementia`. In contrast, most of the rules learned for the research described in this thesis are probabilistic:

> If the patient has memory loss and s/he has had strokes then the patient probably (probability $= 0.90$) belongs to class `Other-dementia`

The second kind of program we will consider are classifiers. The input for a classifier consists of the output of a learning program and an object (described by attributes) whose class must be determined by the classifier program. The output of the classifier for such an object (called a *test example*) is the class that it predicts for that object.

For scientific purposes, it is often necessary to compare the performance of classifiers. For this we need novel objects to test the classification programs. Using the Alzheimer's disease analogy, this corresponds to testing the student after the medical training period. The senior doctor may present a new patient to the student and not disclose to the student what diagnosis the senior doctor would make for the patient (we assume for now that the diagnosis of the senior doctor is always correct). The student is provided with all the values for all the attributes used to describe the patient. The student then has to make his or her own diagnosis. The diagnosis is noted by the senior doctor. However, one patient is clearly not enough to test how well the student has learned: many patients are needed. So, the senior doctor presents many new patients to the student and for each patient compares the diagnosis made by the student to the diagnosis made by the senior doctor. At the end of the test, the doctor can then compute what fraction of diagnoses were correctly made by the student. This score can be used to evaluate the student.

This process is equivalent to the procedure followed in machine learning to determine the classification accuracy after learning. After learning, a set of examples called *test examples* are presented to the classifier. The true class labels of these examples are not presented to the classifier. For evaluation, the classification made by the classifier is compared to the true class of the example. The fraction of test examples correctly classified is called the *accuracy* of the classifier. This rate is often multiplied by 100 to yield the percentage of test examples correctly classified. The fraction that is misclassified is termed the *error rate* of the classifier.

1.2 Learning programs

There are three broad kinds of learning programs that will be discussed in this thesis:

1. Programs that learn a single description that *differentiates* (or *discriminates*) among the classes. Such descriptions are also called *multi-class* descriptions. For the Alzheimer's disease task, the following is an example of a discriminating or multi-class description in rule form (many other forms are also possible):

If	the patient does not have any memory loss	
then	the patient belongs to class `Normal`	
else	if	the patient has had strokes or depression
	then	the patient belongs to class `Other-dementia`
	else	the patient belongs to class `Alzheimer's`

   This rule is a gross over-simplification of the task; the point is that *one* discriminating description is used for all three classes in the task.

2. Programs that learn one description for each class. Such descriptions are also called *one-per-class* descriptions. For the Alzheimer's task, the following are descriptions for the three classes in this task (although rules are used in this example, many other types of descriptions can also be used):

   - **Alzheimer's-** If the patient has memory loss and the patient has not had strokes and does not have depression then the patient belongs to class `Alzheimer's`.
   - **Other-dementia-**
     (a) If the patient has memory loss and s/he has had strokes then the patient belongs to class `Other-dementia`.
     (b) If the patient has memory loss and depression then the patient belongs to class `Other-dementia`.
   - **Normal-** If the patient does not have memory loss then s/he belongs to class `Normal`.

   Note that two distinct rules are required to characterize the `Other-dementia` class. When more than one rule is learned to characterize a class it may signal the presence of sub-classes within that class.

3. This last group of learning programs can only be applied to two-class classification tasks. One of the classes must be specified as the "positive" class. The program learns a description only for the "positive" class. Returning to the Alzheimer's task, imagine that we were only interested in knowing if the patient had some form of dementia (grouping together the formerly separate classes of `Alzheimer's` and `Other-dementia`) or was "normal". Then the three-class task could be restated as a two-class task. If the `dementia` class were specified as the "positive" class, then the program would only learn a description for the `dementia` class. An example of a learned description (consisting of a single rule in this case) for class `dementia` might be:

   If the patient has had memory loss then the patient belongs to class dementia.

Two points about learning are especially worth noting:

1. The aim of learning is usually to *generalize* so that novel objects not seen during training can be accurately classified.
2. There may be many different models that might seem equally good as evaluated on the training data but these models may have have different error rates when used to classify test examples. The problem is to decide how to choose among such models.

As the training set does not contain all possible examples, the learning program needs to learn class descriptions which *generalize* to objects not seen during training.[2] The notion of "all possible examples" may seem foreign to someone new to machine learning. What does the set of "all possible examples" look like? Let us assume for the moment that the attributes completely determine the diagnosis. So it is not possible to have two patients that have the same set of values for all the attributes yet have different diagnoses. Under this assumption, consider the Alzheimer's case study in which the ordered set of attributes is {`Age-of-memory-loss, Number-of-strokes`}. Let us restrict the first attribute to take integer values between 1 and 150 inclusive and the second attribute to take integer values between 1 and 100. Because a patient is represented in a computer merely as a list of values for these attributes, that means there can only be 150×100 different kinds of patients (as far as the computer is concerned with respect to this chosen set and chosen order of attributes). So the set of "all possible examples" in this case is the entire set of the 150×100 lists of length three. This set of "all possible examples" is called the *example space*. Lists of length three instead of two (the number of attributes) are required since we also want to store the class label for each example.

[2] A notable exception is instance-based learning (e.g. (Aha, 1990)) which stores in memory all or most of the examples seen during training and does not make an effort to try to learn class descriptions.

If the training set contained all 150×100 examples (patients), together with the correct class (diagnosis) of each patient, there would be nothing left to generalize. Mere memorization, an easy task for a computer, would always yield correct classifications (assuming that only the two chosen attributes are sufficient to make correct diagnoses) because each possible example together with its class-label has already been seen. In practice, the training examples are just a small subset of all possible examples, so the point of learning is to generalize to novel examples.

If we drop the assumption that the set of attributes uniquely determines a class, then we allow the possibility that two different examples can have the same set of attribute values yet have different class labels. (In such situations, it is necessary to learn probabilistic rules - learning absolute rules leads to the learning of many, inaccurate rules.) Now the set of "all possible examples" becomes larger: $150 \times 100 \times 3$ (assuming there are three classes). The implication of this is that two patients may present with identical symptoms yet have different diagnoses. This may happen if the diagnosis was made using information that is no longer available to the computer or the medical student. Now the set of "all possible examples" consists of $150 \times 100 \times 3$ lists of length 4. So, for instance, the list (65,1,Alzheimer's,0.83) may be in the example space. This list would encode the fact that 83% of the patients that first had memory loss at age 65 and have had 1 stroke belong to class `Alzheimer's`.

The second point worth noting about learning is that when the training set does not contain all possible examples, but only a small subset thereof, then many different descriptions may be formed, even when using the same set of attributes and training examples. This is a stumbling block for machine learning programs because sometimes there are many different descriptions that seem equally good for the training data. Traditionally, computer learning programs have had to go with one choice when faced with such a set of apparently equally good descriptions. Some recent programs have begun to store more than one possible description (some of the earliest such programs are described in (Gams, 1989; Buntine, 1990; Kwok & Carter, 1990)). Chapters 8 and 9 show that making classifications based on retaining many distinct descriptions for each class are often more accurate than classifications based on a single description (per class) learned from data. These results are true no matter whether the learned description is of the multi-class kind or whether it is of the one-per-class kind.

The problem of multiple descriptions has the following analogue in the Alzheimer's case study: different doctors may learn different descriptions even from the same set of patients seen during training. Consider two doctors that are both learning discriminatory kinds of descriptions and furthermore assume that they are both using the same set of attributes. Even under these conditions, if the set of attributes being used is large, there may be many different ways of differentiating the three classes of patients. This is especially likely if the number of patients seen so far is small and the set of attributes used to describe patients is large. The different descriptions being used by the doctors may not manifest their differences on the set of training patients but such differences might manifest themselves later, on new patients. The approach of storing and using multiple descriptions can be thought of as equivalent to consulting more than one doctor. Then, the diagnoses of the doctors could be combined in some manner to produce an overall diagnosis. Research in machine learning (e.g. (Gams, 1989; Buntine, 1990; Kwok & Carter, 1990)) has shown that classifications produced by combining individual classifications in this manner usually tend to be more accurate (in the long run) than those produced by any single classifier.

1.3 Classification programs

Corresponding to the three kinds of learning programs listed in the previous section are three kinds of classifiers.

1. *Classifiers that use a single, discriminating description:* Suppose the learning program has learned the following discriminatory description:

   > If the patient does not have any memory loss then the patient belongs to class `Normal`, else if the patient has had strokes or depression then the patient belongs to class `Other-dementia`, else the patient belongs to class `Alzheimer's`.

   This rule would be applied by the doctor (classifier) by first checking if the patient has memory loss If there is no memory loss, the patient would be diagnosed as normal (i.e. the test example would be assigned the class `Normal`). If there is memory loss, the other conditions in the rule would be checked. The disadvantage of making classifications using discriminatory rules is that if even one of the conditions in the rule is wrong, the rest of the rule is completely wrong. However, this approach to classification has the advantage that the basis of the classification can be explained (to another doctor, for example) by stating the rule used for discrimination.

2. *Classifiers that use a description learned for each class:* A doctor using this kind of classification scheme would try to match the present patient against all the descriptions of all the classes. If the patient only matched the description of one class, she or he would be diagnosed with that "disease." If the patient matches descriptions of more than one class, however, some form of *conflict resolution* is needed. That is, if the patient matched descriptions for Alzheimer's and Other-dementia, for example, the doctor could decide that the rule for Alzheimer's is more reliable and so the doctor would diagnose Alzheimer's for that patient. In general, however, there may be

many rules for each class. For example, some of the `Alzheimer's` rules may be more reliable than some rules of class `Other-dementia`. Some rules of `Other-dementia` may be more reliable than some rules of `Alzheimer's`. So a conflict between these two classes, for example, would not always be resolved in the favor of the `Alzheimer's` class.

3. *Classifiers that only work on two-class classification tasks:*
   Using the Alzheimer's disease analogy, this kind of classifier may be used when the (two-class) task is to determine whether the patient has some kind of dementia or if the patient is normal. The doctor may have learned a description for only those patients that have dementia and would try to see if that description applied to the new patient. If so, the new patient would be diagnosed as having a dementia, and if not, the doctor would by default, diagnose the patient as "Normal." Basically, the doctor assumes that learning a description for the dementia class is easier than the alternative, and assumes that any patient not matching the dementia description belongs to the class "Normal." In machine learning terminology, this assumption is called the *closed-world assumption* (Lloyd, 1984).

1.4 Problems addressed in this thesis

This thesis is mainly concerned with learning of one particular kind of class description: rule sets. That is, each class is described using a set of rules, where each rule hopefully identifies a sub-class of the class. For example, consider the following description that may be learned to describe the `Other-dementia` class:

- If the patient has memory loss and s/he has had strokes then the patient belongs to class `Other-dementia`.
- If the patient has memory loss and depression then the patient belongs to class `Other-dementia`.

This rule set describes the class `Other-dementia` using two rules. If sufficient numbers of training examples are used, the hope is that each rule will describe a different sub-class of the class `Other-dementia`. Learning programs usually aim to learn descriptions that are complete and accurate. To be complete for the set of training examples for a class, each training example should be described by at least one rule. To be accurate for the set of training examples, each rule for a given class must not describe any examples of any other class. That means for example, that each rule in the `Other-dementia` class description shown above, cannot describe any training examples of class `Alzheimer's` or class `Normal`. Our hope is that each rule learned for a class describes a separate sub-class within that class.

One difficulty that arises during learning is that there can be many different rules each of which describes all the training examples of that sub-class with perfect accuracy and yet some of those rules may be quite errorful on a separate set of test examples. The way in which the program decides between these apparently equivalent rules is called the *bias* of the program. Hopefully, the bias encodes some prior knowledge about the task at hand that might be used to decide among apparently equally good rules.

In an effort to make computers learn class descriptions consisting of sets of rules, some problems are encountered. Four of these problems form the focus of this thesis. These are:

1. The need to model probabilistic (non-absolute) rules such as:

   > If the patient has memory loss and depression then the patient probably (with probability 0.9) belongs to class `Other-dementia`.

   Absolute rules are those whose conclusion can be drawn with complete certainty. For the probabilistic rule above, two different interpretations of the associated probability are possible. In the first, the rule is a perfect model of an "intrinsically probabilistic" process. In the second, the underlying rule is deterministic, but the "probability" attached to the rule expresses uncertainty about the validity of what has been learned - i.e. that the learned rule is probably not a perfect model of the underlying target rule. HYDRA uses both notions in different contexts.

2. The need to learn accurate rules despite the presence of noise in the training examples. Noise can refer to noisy class labels (e.g. patients seen during the training period for whom the diagnosis provided by the senior doctor was incorrect) or noise can refer to noisy values for attributes (e.g. patients whose medical records contain errors).

3. The small disjuncts problem (Holte et al., 1989). This problem refers to the difficulty observed in rule learning programs whereby rules (also known as disjuncts) that describe a small fraction of the training set are responsible for a relatively larger fraction of the classification errors made on a test set. Therefore, if one wishes to reduce overall classification error, one good place to start would be to try to reduce the error due to such small disjuncts.

4. The problem of generalizing from a small training set: Many different descriptions may appear equally good as measured by statistics on the training set and yet lead to different classifications for test examples. The problem for the learning program is to decide which one to select.

These problems are now considered in turn.

For many medical diagnosis classification tasks, absolute rules are inadequate to model the process of inferring a disease from its symptoms. An example of a rule that is not absolute is the following:

> If the patient has memory loss and depression then the patient probably (with probability 0.9) belongs to class `Other-dementia`

This rule expresses the fact that although most of the patients that satisfy the conditions of the rule (i.e. have memory loss and depression) belong to the class `Other-dementia`, a small percentage (10%) belong to other classes. Such rules will be referred to as probabilistic rules. Many learning programs only search for absolute rules in learning class descriptions. If such a program is run on a task that requires the learning of probabilistic rules however, it usually ends up learning many rules for each class, each rule describing a small percentage of the patients of that class. These rules do not correspond to different sub-classes: the fact that many rules are learned is an artifact of the inability to learn probabilistic rules. A program capable of learning probabilistic rules, on the other hand, will hopefully learn the few correct rules for each class, and hopefully those rules will correspond to different sub-classes. The assumption we are making is that there are tasks for which learning a few probabilistic rules will lead to higher classification accuracy than the accuracy achieved by learning many absolute rules. Therefore, the first problem addressed in this thesis is to learn probabilistic rules.

The second problem addressed in this thesis is the learning of class descriptions from "noisy" training examples. For instance, consider the possibility that the medical student has to learn from a set of patients, some of whose diagnoses (provided by the senior doctor, for example) are incorrect. This kind of error is referred to as *class noise*. In machine learning terms, errors in the data are referred to as noise and because this kind of error involves the class label of training examples, it is called class noise. Another kind of noise would correspond to errors in the medical records of the patient. For example, someone might have made a mistake in measuring the blood-pressure of the patient. This kind of noise is referred to as *Gaussian noise*. Gaussian noise has the property that the probability of small measurement errors occurring is greater than that of greater errors occurring. A third kind of noise is analogous to what might be caused if the patient's medical record is being electronically transmitted. A noise in the transmission may corrupt the value for a measurement of the patient. Typically, such kinds of corruption reset the value of the measurement to some random value. This kind of noise is different from Gaussian noise because small perturbations of the correct value are as likely as large ones. This form of noise is called attribute noise. Consider what would happen to the class description for `Other-dementia` if a patient who really has memory loss is recorded as not having memory loss. Recall that memory loss is a condition in both the rules for class `Other-dementia`. If absolute rules are being learned and we are assuming that there is no attribute noise then this patient might signal that for a rare sub-class of patients of class `Other-dementia`, memory loss is not necessary. This would lead to the learning of a third rule for `Other-dementia`. In general, attribute or class noise leads to the learning of many rules that are not modeling the class but rather, are modeling the noise. This is called the problem of *over-fitting the noise*. In this thesis, we will attempt to make

Table 1.2: A class description consisting of a large disjunct describing 95% of the training examples of class `Other-dementia`, and a small disjunct describing 5% of the training examples of class `Other-dementia`. The small disjunct is responsible for a disproportionate percentage (50%) of all the errors of commission made by both rules. Disjunct1: If the patient has memory loss and strokes then the patient belongs to class Other-dementia. Disjunct2: If the patient has depression then the patient belongs to class Other-dementia.

Learned rule (disjunct)	Fraction of training examples of `Other-dementia` described	Number of test examples of `Other-dementia` described	Number of test examples of other classes described	Responsible for fraction of errors of commission
Disjunct1	0.95	190	10	0.50
Disjunct2	0.05	10	10	0.50

accurate classifications when learning from training data that has been affected by attribute noise and class noise only. Gaussian noise will not be addressed directly but treated as attribute noise.

A third problem (the small disjuncts problem) that afflicts rule-learning programs is that learned rules that each describe a small fraction of the training examples frequently have high error rates on test examples. The error rate of a rule for class C is the fraction of test examples of other classes that match the rule. So, for instance, consider the following learned rule:

> If the patient has depression then the patient belongs to class `Other-dementia`.

This is the rule that is learned, but the correct rule that it should have learned is:

> If the patient has memory loss and depression then the patient belongs to class `Other-dementia`.

Note that the conditions of the learned rule are too general as compared to the correct rule. That is, they will also apply to some patients that are not of class `Other-dementia`.

As an example of the small disjuncts problem consider the two-class classification task in which the goal is to try to differentiate patients in class `Other-dementia` from the other class (which comprises patients that are either "Normal" or have Alzheimer's dementia). Assume that the class `Other-dementia` has been nominated as the "positive" class so a class description is only learned for it and not for the other class. Assume the class description learned for `Other-dementia` is as given in Table 1.2.

This class description reflects the fact that most of the patients in class `Other-dementia` are in that class because they have suffered strokes and memory loss. Relatively few are in the class due to depression. Assume that the learned conditions of the first rule are overly general because the correct rule is:

> If the patient has memory loss and has had more than 2 strokes then the patient belongs to class `Other-dementia`.

From examining the correct rule it is apparent that examples of other classes in the *test* set who have had memory loss and fewer than 2 strokes will match the first, overly general rule learned for `Other-dementia`. Such patients will be incorrectly classified by the learned rule as belonging to class `Other-dementia`. This kind of classification error is called an *error of commission.* Rules that cover a small fraction of the training examples often make errors of commission. Overly general rules may be learned if there is noise in the training set or if there are many rare sub-classes in the class or if not enough examples of the other classes have been included in the training set.

Let us also assume that the condition of the second learned rule is also too general so it too will lead to errors of commission. The learned class description given in the table illustrates the small disjuncts problem because the small disjunct (the second rule) is responsible for 50% of the errors yet it describes a much smaller percentage (5%) of the training examples of class `Other-dementia`.

The final problem that will be addressed by this thesis is the problem in which many rules *appear* to be equally good, as measured on the training data, but have quite different error rates on test examples. One reason this situation may occur is if some attributes are almost the same as each other. For instance, an attribute measuring whether loss of appetite has occurred may be highly correlated with the attribute indicating depression. In fact, the correlation may be so strong that there may not be any exceptions (patients who have depression but do not have loss of appetite, for example) in the *training* data. If relatively few examples are used for training, the difference between these two attributes may not be manifest on the training set and so rules involving the attribute `Loss-of-appetite` would appear to be just as good as rules involving the attribute `Depression`.

Briefly, a rule is considered "good" if it describes a significant fraction of the training examples (i.e. it has high coverage or high completeness) of the class corresponding to its conclusion *and* if it is accurate. Accurate rules are sometimes referred to as *pure* rules. In the Alzheimer's disease analogy, if all the learned rules are accurate then when a patient satisfies the conditions of a rule for class `Other-dementia`, for example, then one can confidently conclude that the patient does indeed have the "Other-dementia" disease. As an example of a rule that has high coverage, consider the first rule in Table 1.2 which covers (i.e. describes) a significant fraction (0.95) of its class (`Other-dementia`). As an example of a rule that satisfies only the accuracy condition *with respect to the training set only*, consider the second rule in Table 1.2 which is quite accurate (90%) on the training set but which has low coverage (5%). Note that this rule is *not* accurate on the test set since its error rate on the test set is 50%.

In situations where there are many correlated attributes, rules containing the correlated attributes may look equally good (as measured on the training set). In such situations, learning programs have to make an arbitrary decision about which rule to choose. Although this may not seem like much of a problem: after all, the attributes are correlated so it should not matter much which one is picked, it can lead to differences in classification accuracy of a few percentage points. Although the difference is only a few percentage points, those few points in medical diagnosis, for example, may well be worth quite a lot in human terms and financial terms.

1.5 Proposed solutions

It is proposed that rule learning programs can be modified in three ways to solve the first three problems. Each modification will affect more than one of the problems. The modifications are described first and then a description of the manner in which they address the problems is given.

1. During learning a rule for some class C the program should allow rules as long as the set of examples for which the conditions of the rule holds is mostly comprised of examples of class C. The proportion of examples of other classes described by the rule should be small. It is hoped that the rules learned in this manner will correspond to the kinds of probabilistic rules that are needed to model medical diagnosis, for example.
2. A measure of the reliability of the rule should be estimated using the training set and it should be associated with the rule.
3. In two-class data sets, learning a description per class should allow more accurate classifications than would be made by an approach that simply learns a description for one of the classes and everything which is not described by that description is assumed to belong to the other class.

These modifications address the first three problems described in the previous section in the following manner. Recall that the first problem required the modeling of probabilistic rules. This is achieved by the first approach listed above. By learning rules that describe a small number of examples of other classes, the learned rules are no longer absolute. That is, they mostly describe examples of one class but their conditions are also true for a few examples of other classes. So such rules can model the kinds of probabilistic rules that are needed in medical diagnosis applications.

The second problem is to learn accurate rules from noisy training examples. All three of the proposed modifications help to address this problem. Consider learning a rule for class `Other-dementia` that describes a few training examples of class `Alzheimer's`. However, if there is class noise in the training data then some of the patients that are diagnosed as `Alzheimer's` may actually belong to `Other-dementia`. Therefore, a rule for `Other-dementia` that appears to describe a few examples of `Alzheimer's` may actually be describing `Other-dementia` examples that were misclassified as `Alzheimer's`. In this way, it is hoped that patients who really have the "Other-dementia" disease but are misdiagnosed as `Alzheimer's` can be identified because they more closely follow the regularities pertaining to the `Other-dementia` patients rather than the regularities pertaining to the `Alzheimer's` patients.

Learning a description for each class and making reliability estimates, work in tandem to allow more accurate classifications to be made when learning from noisy data. Consider what might happen in a two-class task (`Normal` versus the other two classes grouped together into a new class that will be referred to as `Abnormal`) if class `Normal` was identified as the "positive" class, and rules were only learned for `Normal`. Assume that a rule learned for `Normal` has overly general conditions. That is, the rule may describe some examples that are truly not of the `Normal` class. Therefore, the rule would end up misclassifying some test examples of the `Abnormal` class as belonging to `Normal`. Consider instead what would happen if a description were learned for each class and the rules were augmented with reliability estimates. Now, if an overly general rule is learned for class `Normal` it may lead to fewer misclassifications because of the following process: say the test example (whose true class is `Abnormal`) matches the conditions for the `Normal` rule but now because a description was learned for the other class, it also matches a rule of its correct class - `Abnormal`. This represents a situation with conflicting evidence. One way to resolve this conflict is to use the more reliable piece of evidence. It is hoped that on average, erroneous rules will be assigned lower reliabilities. Therefore, in such a conflicting situation, the erroneous rule will be rejected in favor of the more reliable, correct rule. This discussion illustrates how learning a description for each class and using reliability estimates overcomes some of the problems of learning from noisy training examples.

The problems due to the small disjuncts problem are also lessened by learning a description for each class and using reliability estimates. The small disjuncts problem refers to the high error rates of (overly general) rules (i.e. disjuncts) that describe a small fraction of the training examples. But our hypothesis is that such rules will be assigned lower reliabilities. Therefore, in a situation of conflicting evidence, small disjuncts will lose to more reliable, larger disjuncts. Therefore, what used to be a misclassification due to the small disjunct will have been averted.

A proposed solution to the the problem of learning for tasks in which many different rules appear to be equally good is now discussed. The proposed solution is to learn several descriptions for each class. This is called the multiple models approach. That is, if there are many apparently equally good rules, all of which are pretty good models for a sub-class then instead of making an arbitrary choice among them, one should retain more than one of them. This solution enjoys theoretical support from Bayesian probability theory (e.g. (Buntine, 1990)) and also empirical support in that it has been experimentally shown to usually lead to more accurate classifications (e.g. (Gams, 1989; Buntine, 1990; Kwok & Carter, 1990; Kononenko & Kovacic, 1992)). Chapter 7 provides empirical evidence that this approach also allows more accurate classifications to be made when learning from noisy data. In this method, evidence needs to be combined from many descriptions of a class so this method also provides a way of modeling probabilistic rules. Therefore, this approach addresses the problem of learning probabilistic rules and the problem of learning from noisy data.

The proposed multiple models approach raises two questions: 1) how should more than one model be learned from a given training set? and 2) how should the classifications of those models be combined to form an overall classification? Different solutions to these questions are compared in Chapter 7. The multiple models approach is being suggested as an aid to generalizing from small sample sizes and since a small disjunct may correspond to a rare sub-class with small sample size, the multiple models approach should provide help for the small disjuncts problem.

1.6 Contributions

Through a more in-depth exploration of the multiple-models approach than has been done in previous research, the research described in this thesis allows a better understanding of how multiple models help improve accuracy. The few situations in which they do not lead to increased accuracy are also characterized and explained. A model explaining the *degree* of error reduction afforded by the multiple models approach is also presented - previous work has not analyzed the degree of error reduction. Chapter 8 also proposes a new measure (called *gain-ties*) for predicting how helpful the multiple models approach will be for any given learning task. The important issue of whether multiple models help because they involve more search for good descriptions or whether they help because they vote together in making classifications is also explored. The multiple models results also contribute to research in the area of using probabilistic rules since evidence combination form multiple descriptions of one class is needed in the multiple models paradigm. Three methods for generating multiple models from one training set are compared, and four methods for combining evidence from such models are also compared. The general conditions under which each method is superior are characterized.

Another contribution of the research described in this thesis is the demonstration that learning a description for each class and using reliability estimates leads to more accurate classifications when learning from noisy data. This approach also reduces overall classification error in learning tasks affected by the small disjuncts problem. Whereas the initial attempt (Holte et al., 1989) to reduce overall error in such situations was not successful[3] the approach of using reliability estimates and learning a description for each class leads to reduced error rates for the small disjuncts and also to reduced overall error rates. It is also shown experimentally that learning a description for each class and using reliability estimates allows more accurate learning from small, noisy, training sets than two other approaches for dealing with noisy data: Reduced-Error Pruning (Quinlan, 1987; Brunk & Pazzani, 1991) and MDL (Quinlan, 1990).

1.7 Overview of dissertation

The next chapter presents rigorous definitions for the classification task and learning task and for the terms used in this thesis. Chapter 3 reviews foundational research needed for a technical understanding of the problems addressed in this thesis. Chapter 4 presents HYDRA, the rule-learning algorithm used to address these problems and compares variants of HYDRA. The technical solution to the small disjuncts problem is then given in Chapter 5. Chapter 6 shows how HYDRA is able to learn accurate relational probabilistic definitions from noisy data. The main method of dealing with the problem of "many, apparently equally good rules" is the multiple models approach and it is presented in Chapter 7. Chapter 7 also compares different ways of generating multiple models and different ways of combining evidence from such models. Chapter 8 explores various issues relating to multiple models; in particular, the hypothesis that multiple models are able to reduce error to the extent that the models make uncorrelated errors is explored.[4] This analysis (Ali & Pazzani, 1995d) is also done for decision trees (Quinlan, 1986) and artificial neural networks (Rumelhart & McClelland, 1986). A method for predicting (from the training set) the amount of error reduction due to the use of the multiple models approach is also evaluated.

---

[3] In the attempt of Holte *et al.*, the error rates of small disjuncts were reduced but unfortunately, there was a corresponding increase in the error rates of the other disjuncts such that the overall error remained the same.

[4] The research linking the amount of error reduction and the extent to which models make uncorrelated errors for rule-set models is available in technical report form in (Ali & Pazzani, 1995a).

Chapter 2
Definitions

This chapter begins with definitions of terms related to the representation of objects as examples. Next, an intuitive description of first-order logic (also known as predicate logic; e.g. (Lloyd, 1984)) as a language for representing rules is given. This is followed by a rigorous definition of the syntax of first-order logic. The syntax defines what kinds of rules are permitted in first-order logic. The semantics of rules in first-order logic are presented next. The second half of the chapter deals with the definitions of classification tasks and learning tasks. Two kinds of classification tasks are defined: the two-class classification task and the multi-class classification task. The chapter concludes with the definitions of three kinds of *learning* tasks compared in this thesis: two-class learning, multi-class learning and learning from noisy examples.

2.1 Example representation

In preparation for learning, a decision about how to represent objects in the computer must be made by the user. Assuming, for example, that the task is to learn to differentiate patients who have breast cancer from those that do not, one would need to decide what set of attributes would be used to represent the patient. Having made that decision, an *example* is defined to be a list that represents an object in a computer with respect to a chosen sequence of attributes. The example is the list of values that the object has for those attributes. So, for instance, the list (1,65) may represent a patient with respect to the following sequence of attributes:

`Number-of-strokes, Age-of-memory-loss`

This example tells us that the patient represented by (1,65) has had one stroke and had memory loss starting mainly at the age of sixty-five years. This kind of representation is called an *attribute-value* representation. Although lay use of the term "example" requires a referent class, in machine learning this need not be the case. The term "example" may be used without reference to the class that it is an example of. The term "example" only needs a sequence of attributes as a referent. Then, if one wants to state that the example is a member of a class C, the phrase "positive example of C" is used. "Negative example" is used when the example is not a member of the class currently under discussion. So, for instance, an example representing a woman would be a negative example of the class of all males. An example representing a man would be a positive example of that class.

A *class* is defined to be a set of examples. The set of positive examples that define a class is called the *extensional definition* of the class. Let V_i denote the i-th attribute. Then the *domain* of V_i, denoted $dom(V_i)$, is defined to be the set of permissible values for V_i. The *example space* of an example with respect to the ordered-set (sequence) of attributes $(V_1, \ldots, V_n)$ is the product of all the domains of the attributes V_1 through V_n. The *dimension* of an example space is the number of attributes used to construct that space.

A quite different way of looking at a class is to consider a class to be a set of tuples where each tuple is a list of objects (in database terminology, objects are called *entities*). So, for instance, the list `(adam,eve)` consisting of the object `adam` and the object `eve` would be a positive example of the class `married-couple`. When examples are used in this way, the class that they are members of is called a *relation* and the example is called a *tuple*. A tuple of length n is called a n-tuple. Elements of a tuple are called *components*. Generally, an example in the attribute-value representation represents different attributes of a single object. A tuple, on the other hand, usually represents an instance of a relationship between more than object. There are no attributes in the relational view. A tuple is just a list of numeric or symbolic values.

In learning, two kinds of examples are used: *training examples* and *test examples*. The set of training examples is referred to as the *training set* and the set of test examples is referred to as the *test set*. Training examples are used to produce the learned concept descriptions. Test examples are necessary to evaluate the classification accuracy of the output of the learning algorithm. First, the learning algorithm is run on the training examples together with their class labels. That is, each example, together with the class of which it is a member is presented to the algorithm. The algorithm is run on these examples in a phase called *learning* or *training*. Usually there is a classification algorithm coupled with the learning algorithm. After learning is complete, this associated classification algorithm is tested on a set of examples - this test set is usually disjoint (or has little overlap) with the training set. The class labels of test examples are not presented to the classification algorithm. The task of the classification algorithm is to take as input a test example and the output of the learning algorithm and produce a class label (the predicted class for that example). The fraction of test examples correctly classified by the classification algorithm is called the *accuracy* of the learning/classification algorithm combination. Having defined what an example is and what a class is, the next set of definitions pertain to the output of the learning algorithm. The

Table 2.1. Computer representation of a rule.

breast-cancer $\leftarrow$ Age $>$ 50, Residence $=$ smoky-city, Degree-of-Malignancy $=$ 3.

output of most of the learning algorithms discussed in this thesis will be one or more class descriptions in first-order logic.

2.2 First-order Logic

This section first gives an intuitive presentation of how rules are used and an intuitive description of what kinds of rules are allowed in first-order logic. Next, it provides a rigorous definition of the syntax and semantics of first-order logic which is the language used to represent the rules discussed in this thesis.

Chapter 1 presented rules in English form but the computer stores the rules more in the form of the format shown in Table 2.1. This is an attribute-value rule in attribute-value representation and it is read as follows. Given an example with a value for each of the attributes `Age`, `Residence` and `Degree-of-Malignancy`, that example should be classified to class `breast-cancer` if the value for the attribute `Age` is greater than 50 and the patient lives in the city with name `smoky-city` and if the value for the attribute `Degree-of-Malignancy` is greater than 3. The conclusion of the rule (`breast-cancer`) is also called the *head* of the rule or the *consequent*. The preconditions of the rule (`Age > 50, Residence = smoky-city, Degree-of-Malignancy = 3`) are also called the *body* or the *antecedents* of the rule.

A rule is an attribute-value rule if no single precondition of the rule involves more than one attribute or if the rule involves attributes just of a single object. Rules that involve attributes of more than one object are called first-order (or *relational*) rules and can most easily be represented and learned using first-order representations. First-order representations can also easily represent attribute-value rules but to go the other way is much more difficult. It is difficult to represent or learn first-order rules written in attribute-value representation. Table 2.2 gives the first-order form of the rule first presented in Table 2.1.

Table 2.2. First-order representation of a rule.

breast-cancer(Age,Residence,Degree-of-Malignancy) $\leftarrow$ $>$(Age,50),
$=$(Residence,smoky-city),
$>$(Degree-of-Malignancy,3)

The attributes are now represented by *variables* in the head of the rule. The attributes or variables are used to represent an example (a patient for instance). This is a first-order representation because the head of the rule is a syntactic form containing variables. The rule is still attribute-value, however, since none of its preconditions involves more than a single attribute (variable).

If a clause or rule is augmented with a probability estimate, such a rule is called a *probabilistic rule*. Imagine that the `breast-cancer` rule just given is a probabilistic rule with the probability signifying the proportion of tuples that are in the relation `breast-cancer` *given* that those tuples satisfy the body of the rule. Let us assume that this probability is 0.90. This means, that 90% of the patients for whom the body of the rule is true have breast cancer. Such a rule can be written in conditional probability form:

$$
\begin{aligned}
&p(\text{breast-cancer}(\text{Age},\text{Residence},\text{Degree-of-Malignancy}) \mid \\
&\qquad >(\text{Age},50), =(\text{Residence},\text{smoky-city}), >(\text{Degree-of-Malignancy},3)) \\
&= 0.90
\end{aligned}
$$

See (Bacchus, 1990) for further exploration of the interaction of first-order logical quantification with probability theory.

The following is an example of a first-order rule represented in a first-order representation:

$$\text{uncle}(X,Y) \leftarrow \text{male}(X), \text{sibling}(X,Z), \text{parent}(Z,Y).$$

This rule says that X is the uncle of Y if X is male and X has a sibling Z and Z is the parent of Y. Note that although this rule has the same syntax as the previous rule, it has conditions that involve more than one object and thus it is a first-order rule. For instance, the condition `sibling(X,Z)` involves two objects. This rule is also different from the previous rule in that the variables in the head of the rule take on values that represent objects, not attributes of objects. So, for instance, the variable `X` would take on the value `jim` which is an object with its own attributes. A third difference to attribute-value rules is that the preconditions of the rule may involve references to objects not mentioned in the head of the rule. The `uncle` rule expresses a relationship between two objects (`X` and `Y`) using an intermediate object (`Z`) that is not mentioned in the head of the rule. In general, a first-order rule may involve any number of such intermediate objects and may reference attributes of one or more of those objects. A program capable of learning first-order rules could take as input, a relational database of patients (and their relatives) and their records and it could use attributes of the patient and any relative to learn a rule that discriminates patients that get breast-cancer from those that do not. The attribute-value learning program, on the other hand, could only use the attributes of the patient at hand to learn discriminatory rules. Or, the designer of the database would have to decide in advance, that the attributes of the mother, for instance, were also relevant, and should be added to the set of attributes used to learn the discriminatory rule. The point is that the algorithm cannot make the decision - it has to be guessed at in advance by the designer of the attribute-value database. The naming convention followed in this thesis is that predicates (e.g. `uncle`) and constants (e.g. `microsoft`) will be written starting with lower-case letters and attributes (e.g. `Number-of-strokes`) and variables (e.g. `Z`, `Degree-of-malignancy`) will be written starting with upper-case letters.

Concepts that involve more than one object can have compact first-order descriptions that are easy to learn whereas the corresponding propositional descriptions can be lengthy and hence difficult to learn. For instance, 3-tuples that satisfy the following rule:

$$\text{family(M,F,C)} \leftarrow \text{male(M), female(F), married(M,F), parent(F,C).}$$

can be converted into the following lengthy propositional description by adding boolean attributes (such as *Married*) in the head of the clause (rule) that correspond to the outcome of relational tests (such as *married(M,F)*):

$$\begin{aligned}\text{family(M,F,C,Married,Parents-have-child)} \leftarrow\ & \text{male(M), female(F),}\\ & \text{Married, Parents-have-child.}\end{aligned}$$

Therefore, for a propositional learning algorithm to learn this concept, it or the user must know in advance what relational tests may be required and it must add the results of the same as variables in the head of the clause. Adding attributes corresponding to all possible relational tests would greatly increase the dimensionality of the space which has to be searched for descriptions in a way that makes learning difficult. For these reasons, if the concept being learned has a compact first-order description but a complex propositional description (with respect to some predetermined set of attributes) then using an algorithm that directly learns first-order class descriptions will be more likely to yield accurate descriptions.

2.2.1 Syntax

First, the syntax of first-order logic is introduced. The description learned by a learning algorithm will be a well-formed formula in the first-order logic defined below. First-order logic has two parts: syntax and semantics. The syntax pertains to an arrangement of symbols in a well-formed formula, the semantics pertain to truth assignments to symbols in the formula. A first-order theory is defined over some alphabet which contains seven classes of symbols:

(a) variables (e.g. X)

(b) constants (e.g. pi, 2.8, jim)

(c) functions (e.g. square)

(d) predicates (e.g. male)

(e) connectives (e.g. the symbol denoting logical or: $\vee$)

(f) quantifiers (e.g. the symbol $\forall$)

(g) punctuation symbols (e.g. ",")

The most basic syntactic construct in a first-order language is a *term* which is defined inductively as follows: a variable is a term, a constant is a term and if f is a function and $t_1 \ldots t_n$ are terms, then $f(t_1 \ldots t_n)$ is a term. Intuitively, one can think of functions as mapping from one or more objects to an object. For instance, `husband` is a 1-ary function that maps a person to their husband. So, for instance, the function `husband` might map the term `eve` to the term `adam`. To think of it another way, the *value* of the term `husband(eve)` would be `adam`. To map syntactic constructs to values needs a way of checking what the value of a term such as `husband(eve)` might be. An *interpretation* forms the link between syntactic forms and their semantic values. So, for instance, under one interpretation, the function `husband` might map `eve` to the term `adam`. That is, the interpretation requires full definitions for all the functions and predicates in the given alphabet. But as far as just the syntax is concerned, terms do not have values, they are simply arrangements of symbols that can be manipulated by a strictly specified set of syntactic operations.

A *well-formed formula* (wff) is defined as follows (Lloyd, 1984):

1. If p is a predicate and $t_1 ... t_n$ are terms then $p(t_1 ... t_n)$ is a formula (also called an *atom* or *atomic formula*).
2. If F and G are formulae, then so are $\neg F$, $F \wedge G$, $F \vee G$, $F \rightarrow G$, $F \leftrightarrow G$.
3. If F is a formula, then so are $\forall x F$ and $\exists x F$.

The *first-order language* with respect to some given alphabet (alphabets usually only vary in providing different sets of predicates and functions or different attributes for a propositional language) is the set of all well-formed formulae with using predicates and functions only from that language. Note the difference between a language and alphabet: decision trees and first-order logic are examples of different kinds of language. Varying the set of attributes in a decision tree does not change the language, it merely changes the alphabet.

An *atom* is defined as the syntactic construct: $p(t_1 \ldots t_n)$ where p is a predicate and t_1 through t_n are terms. Again, given an interpretation, a non-syntactic way to think of a predicate is as a mapping from a sequence of objects to a value in the set {`true`,`false`}. So, for instance, the predicate `married` might map the sequence `(adam,eve)` to the value `true`. A *literal* is an atom or the negation of an atom (e.g. ¬`parent(adam,eve)`). The semantics of the ¬ connective are that if an atom A has some truth value, then $\neg A$ has the other truth value. Here is an example of a literal with a variable X: *married(X,eve)*. In order to determine the truth value of a literal, one needs to assign values to the variables in the literal and one needs an interpretation. So, for instance, if X is assigned the value *eve*, the literal will be false (i.e will have the truth value false) under the usual interpretation in which `eve` is not married to herself. On the other hand, if X is assigned the value `adam`, the literal will have truth value `true` under the usual interpretation. Literals without the $\neg$ connective are called *positive literals*, else they are called *negative literals*. Literals without variables are called *ground literals* or in relational database terminology: *tuples*.

A literal is used in the definition of a *Horn clause* as follows. A Horn clause is defined to be the formula

$$\forall X_1 \ldots \forall X_s (A \vee \neg L_1 \vee ... \vee \neg L_n)$$

where A and L_i are atoms and X_i is the set of variables occurring in $(A \vee \neg L_1 \vee ... \vee \neg L_n)$. The clause can also be written as

$$\mathrm{A} \leftarrow L_1 \ldots L_n$$

where A is the positive literal. A is called the *head* or *consequent* of the clause. $L_1 \ldots L_n$ is called the *body* or *antecedent* of the clause. Note that the body is an *unordered set* of literals so permuting the order of literals in the body of a clause does not change that clause. Another special kind of clause is called a *unit clause* (or *fact*) which has the empty body:

$$A \leftarrow$$

(or equivalently, $A \leftarrow \mathit{true}$). An *intensional definition* for a concept C is a unordered set of rules in each of which the predicate in the head of the rule is C. The term *rule-set* is used in machine-learning to refer to this kind of definition. Note that because the definition is a set, rearranging the order of the clauses does not lead to a different definition - this definition excludes the "cut" operator of Prolog from being allowed in an intensional definition. An intensional definition is variously called a *class description*, a *class definition*, *concept description* or *concept definition*. A *logic program* is a set of definitions for each predicate in the alphabet. A *rule-set model* is a set of class descriptions, one for each class in the data set. In research work in learning, it is useful to divide the set of predicates into two groups. The predicates in the first group correspond to classes and predicates in the second group correspond are called *background predicates* and their associated semantic concepts are called *background concepts*. The extensions of background predicates are called *background relations*. Background predicates are used to form class descriptions for the class predicates. If recursive class descriptions are allowed, then the class descriptions can also contain literals whose predicates are class predicates.

The preceding paragraph provides a narrow definition for the term "intensional definition" which works when a class description is a set of rules. In machine learning, "intensional definition" has a broader usage. Given any language $\mathcal{L}$, called the *hypothesis language*, and a set of background predicates (or attributes), an intensional definition is a member of that language. So, the language may be a first-order language with some set of background predicates, or it may be neural networks (e.g. Rumelhart and McClelland, 1986) with respect to some attributes, or decision trees (e.g. Quinlan, 1986) with respect to some set of attributes. In general, a intensional definition may be any fully specified classification structure whose functional semantics are that the structure accepts as input one example and produces as output a class.

A learning algorithm is said to produce a set of *one-per-class* descriptions (one for each class in data) when its output is set of class descriptions such that each training example of that class satisfies that class description and such that no training example of any other class satisfies that class description. When the learning algorithm produces a set of rules for each class in the data, it is producing one-per-class descriptions. A learning algorithm can also produce a *differentiating* description (e.g. decision tree, neural network) of the entire data. A differentiating description is any data structure that can be used by a classification algorithm to produce a class label for an example.

When the learning algorithm produces a set of one-per-class descriptions, that set is called a *model*. When the algorithm produces a single differentiating description for the entire data, that description is called a *model*. So, "model" refers to the total output of the learning algorithm. The term "model" is used differently in statistics and logic: the use in statistics refers to an intensional or compact summary of the data whereas in logic, a model is a mapping of well-formed-formulae to semantic truth-values. The statistics definition will be used in this thesis.

2.3 The classification task

The general classification task (also called the performance task) is to take as input an example of unknown class and a model and to decide the class that should be assigned to that example. As this thesis is primarily concerned with learning class descriptions that consist of a set of first-order rules for the class, the following discussion of the classification task assumes that first-order rule-set class descriptions are being used for classification.

The central task of a first-order classification algorithm is to determine if an example *satisfies* a rule (this is also referred to as the example *matching* a rule, or the rule *covering* the example). (Recall that being an example does not imply that the example is a positive example for a given class.) Let us consider the simplest case where the definition of the class `married` consists of the following fact (a fact is a rule whose precondition is the truth value `true`):

Contents of logic program:

```
married(adam,eve) ←
```

and we want to determine if the example `(adam,eve)` is a positive example of the class `married`. To do so in a mechanical fashion that can be easily programmed in a computer, we create a *goal* which is formed by concatenating the name of the class for which we want to test membership and the example. This yields the goal which is traditionally written as:

```
← married(adam,eve)
```

Next, we try to match this goal against facts in the logic program. To do so, we try to find a fact which *unifies* (see (Lloyd, 1984) for details) with the goal. For this illustration, there is a successful unification between the goal and the sole rule in the logic program which is

```
married(adam,eve) ←
```

so the goal *succeeds* so we conclude that `(adam,eve)` is a member (positive example) of the class `married`.

Consider a more complicated example in which the logic program additionally contains a rule that is not a fact:

Contents of logic program:

```
married(adam,eve) ←
like-each-other(X,Y) ← married(X,Y)
```

and we want to know if the example `(adam,eve)` is a member of the class `like--each-other`. Lloyd (1984) gives a full description of the process, SLD resolution, by which it is determined whether the example is a member of the class. SLD resolution is used by the programs presented in this thesis. Here, a more intuitive presentation is given. The basic idea is as follows:

> Satisfaction condition: An example is a member of a class, if there is a rule whose head contains the predicate corresponding to that class and for which the example satisfies that rule.

An example can satisfy a rule in one of two ways. In the first way, the example satisfies the rule if the rule is a fact and the example unifies with the fact as was illustrated in the previous paragraph. In the second way, to determine if an example satisfies a rule that is not a fact, a variable assignment is formed between the head $C(V_1 \ldots V_n)$ of the rule and the example $(t_1 \ldots t_n)$ in which V_i is assigned the value t_i. Next, the variables in the body of the rule are replaced by their corresponding value. So, given the goal ($\leftarrow$ `like-each-other(adam, eve)`) and the rule

$$\texttt{like-each-other(X,Y)} \leftarrow \texttt{married(X,Y)}$$

we assign the value `adam` to variable X and value `eve` to variable Y. Then, the process replaces variables in the body of the rule with their assignments (also called bindings) and makes a goal out of each literal in the body. In our example, there is only one antecedent in the body so the created goal (also called *subgoal*) is:

$$\leftarrow \texttt{married(adam,eve)}$$

Finally, we are ready to present a recursive definition of when a goal g is satisfied:

> A goal g is satisfied with respect to a logic program $\mathcal{L}$ if
> i) there is some fact in the logic program that unifies with the goal, or
> ii) there is some non-fact rule for which the head of the rule unifies with the goal g under some variable assignment θ and all the subgoals resulting from θ can be satisfied with respect to $\mathcal{L}$.

In our illustration, the subgoal ($\leftarrow$ `married(adam,eve)`) is formed but we already know that will succeed when the fact (`married(adam,eve)` $\leftarrow$) is present in the logic program. Hence we can conclude that the example `(adam,eve)` satisfes the rule `like-each-other(X,Y)` $\leftarrow$ `married(X,Y)`. Hence we have met the satisfaction condition so we can conclude that the example `(adam,eve)` is a member of the class `like-each-other`.

2.3.1 The two-class classification task

The two-class classification task assumes that two classes, respectively called the "positive" and "negative" class, partition the example space. The task takes as input an example to be classified and a class description for the positive class (therefore, the model for such a task consists of just a single class description). The task is to determine if the example satisfies the description for the positive class. Only two outcomes are possible: either the example satisfies the description or it does not. If it does, the example is assigned to class positive otherwise it is assigned to class negative. This policy is called the *closed-world assumption*

(Lloyd, 1984) because it assumes that any example that is outside the closure (the "world") of the positive class description must be a member of the other class. The *closure* of a class description for class C is defined to be the set of examples which satisfy that class description. Note that the closure for a class description will be identical to the extension of that class if and only if that description is a completely accurate description for that class with the accuracy being measured over the entire example space.

2.3.2 The multi-class classification task

The multi-class classification task takes as input an example to be classified and a model and produces as output a class label. The task is to use the model to decide the class to which the example should be assigned.

When the model consists of a set of class descriptions, three outcomes are possible:

1. The example only satisfies the description for one class.
2. The example satisfies descriptions for more than one class.
3. The example does not satisfy any description of any class.

The class to which the example should be assigned is clear in outcome 1. Outcomes 2 and 3 are possible when the class descriptions make class assignments that are not consistent with the true class memberships of the examples. That is, the learned descriptions are consistent over the training data but not over examples outside the training data set. Such descriptions may occur when the description has been learned on a small set of examples. When a model consisting of such descriptions is tested on examples not present in the training set of examples, it may make classification errors. Classification algorithms differ in how they address outcomes 2 and 3. The approach used in HYDRA is described in Section 4.1.

2.4 The learning task

The following is a definition of the learning task:

Learning task: Given

1. a set of classes $\mathcal{C} = C_1 ... C_k$ with disjoint extensions in an example space S,
2. one of the following:

   (a) (for first-order logic) a set of background predicates
   (b) (for propositional logic, decision trees, neural networks, Bayesian networks, ...) a sequence of attributes, $V_1 ... V_n$, used to describe an example,

3. a *training set* consisting of pairs (e, c) where e is an example in the example space and c is the class to which the example belongs (for first-order logic, e is a ground literal, for other languages, e is an element in the example space $S = dom(V_1) \times ... \times dom(V_n)$),
4. a language $\mathcal{L}$ (called the *hypothesis language*) for building definitions

learn a model, $m \in \mathcal{L}$, (called the *target model*) of the data that is consistent with all examples in the entire example space S. Furthermore, the learning algorithm is required to produce the least syntactically complex model of all models that are consistent over the entire example space (this prevents extensional definitions of the classes from being acceptable solutions for the learning task).

A model is consistent with respect to an example if the model assigns the correct class label for that class. The model can either be a *differentiating* description or a set of *one-per-class* descriptions, one per class in the data.

The set of all possible models with respect to a given hypothesis language and alphabet (i.e. set of attributes or background predicates) is also called the *model space* or *hypothesis space*. The *classification accuracy*, $a(M, S)$, of a model M with respect to a set S of examples is the fraction of that set which is correctly classified by M. The accuracy of a model over the entire example space is called the *true accuracy* of that model. In practice, one can only approximate this value by using a large set of examples which is nevertheless much smaller than the entire example space. The *true error rate*, $e(M, S)$, of a model is defined as $1 - a(M, S)$.

A description D for class C is *complete* with respect to some set S of examples if all examples of class C in S satisfy D. A description D for class C is *sound* with respect to some set S of examples if all examples (of classes from $\mathcal{C}$) that satisfy D are positive examples of C. A description that is complete and sound is termed *consistent* with respect to $\mathcal{C}$ and the set S of examples. If the hypothesis language consists of models that consist of a set of one-per-class descriptions, each of those descriptions in the target model is called a *target class description*.

Assuming that the target model is known, a simple way to check whether a learned model has satisfied the learning task is to check for syntactic equivalence. However, it is possible that a model may be syntactically dissimilar to the target model yet have an accuracy (over the entire example space) as high as the target model. Such a model would be a solution to the learning task yet be rejected by the syntactic-equivalence test. Unfortunately, the only way to detect such models is to test over the entire example space.

Few learning algorithms are able to satisfy the strict definition for the learning task just given. Therefore, we define another task, called the *noise-free, local learning task*. For a learning algorithm to satisfy this task, all it needs to do is output a model in $\mathcal{L}$ that is consistent only *with the training data*. It is assumed that the training data has no noise.

The two-class learning task is simply a special case of the general learning task just defined with k, the number of classes, restricted to 2. In addition, one class in C, the set of classes, needs to be identified as the special class (called the *positive class*) for which a concept description should be learned. This is needed since some learning algorithms only learn a class description for the positive class in a two-class scenario. In the context of the two-class learning task, examples of the positive class are called *positive examples*. Examples of the negative class are called *negative examples*.

2.5 Learning in the presence of noise

Class noise is present in a set of examples if for at least one training example, the class label associated with that example as presented to the learning algorithm is not the true class label of that example. Another kind of noise is *attribute noise* (also called tuple noise). Attribute noise is present if for some example, the value for some component (or attribute) of the example as included in the training set is not equal to the true value of that component. There are two kinds of attribute noise: Gaussian noise or transcription noise (Angluin & Laird, 1988). In Gaussian noise, smaller perturbations of the value are more probable than larger perturbations. In transcription noise, all perturbations are equally likely. This kind of noise may occur, for instance, in transmission of data.

If any of these kinds of noise are present, the goal of the learning algorithm is no longer to build descriptions that perfectly discriminate the positive examples of each class. Consider, for instance, what might happen if class noise is present and the learning algorithm insists on learning rules that cover exactly none of the training examples of other classes. Consider a simple two-class learning situation and assume that a concept description for class `a` is being learned. Now due to class noise it may happen that some of the examples labeled as class `b` are really class `a` and hence share the same structure as other examples of class `a`. By "share the same structure" we mean that the examples mislabeled as belonging to class `b` typically satisfy rules satisfied by many of the true examples of class `a` whereas very few of the true examples of class `b` satisfy those rules. The learning algorithm will have difficulty finding rules that cover many examples of class `a` but exclude these mislabeled examples. So, instead it will end up learning many rules, each of which covers a few training examples. Experiments (Muggleton et al., 1992) have shown that for most problems, rules that cover few training examples tend to make more classification mistakes on test examples and so lead to an overall reduction in classification accuracy. In such circumstances, the learning algorithm is not modeling the class, but instead is modeling the noise. In this case, the algorithm is said to be *overfitting* the data.[1] To summarize: noise poses a problem for algorithms that insist on learning rules that cover exactly none of the examples labeled as belonging to other classes. Such algorithms (called *noise*-in*tolerant algorithms*) learn many rules which in turn leads to poor classification accuracy.

So, the task of learning from noisy examples is no longer to learn a description that is consistent with the training set. The task of learning from noisy data is then defined as follows:

Task of learning from noisy data: Given

1. a set of classes $\mathcal{C} = C_1...C_k$ (whose extensions no longer need to be disjoint) in an example space S,
2. one of the following:
   (a) (for first-order logic) a set of background predicates
   (b) (for propositional logic, decision trees, neural networks, Bayesian networks, ...) a sequence of attributes, $V_1...V_n$, used to describe an example,
3. a *training set* consisting of pairs (e, c) where e is an example in the example space and c is the class to which the example belongs (for first-order logic, e is a ground literal, for other languages, e is an element in the example space $S = dom(V_1) \times ... \times dom(V_n)$),
4. a language $\mathcal{L}$, called the *hypothesis language*, in which the learned class descriptions are expressed,

learn a model, m, in $\mathcal{L}$ such that no other model in $\mathcal{L}$ has a higher accuracy than m where the accuracy is measured over the entire example space S. Furthermore, the learning algorithm is required to produce the least syntactically complex model of all models that have the highest accuracy over the entire example space.

Note that for class descriptions, all descriptions that are reorderings of the rules and/or reorderings of the literals in the rules are considered equivalent. Note also

[1] See the right figure in Figure 6.2 which shows that on noisy data FOIL learns too many rules and overfits the data and obtains a lower accuracy than a less overfitting algorithm such as HYDRA.

that there may be more than one model (that is not a reordering of any other model) that shares the optimal accuracy.

Unfortunately, in general, the only way to accurately judge the accuracy of the learned model on test examples is to actually run the model on the test examples. Statistical tests try to identify error-prone rules but are not nearly as good as actually testing the rule on test examples. Learning methods that try to prevent overfitting sometimes produce accuracy estimates for the model but in general, these estimates are not very close to the true accuracy of the learned model.

There are at least two distinct situations that are generally considered under the heading of learning from noisy data. In the first, noise has been artificially added (by the machine learning experimenter, for example) to data in which the target descriptions for all the classes are known. In the second, the noise is intrinsic in the data. These situations will now be considered in more detail.

2.5.1 Learning when noise is artificially added

Here, it is assumed that noise has been added to an initially noise-free data set. It is also assumed that there is a model in $\mathcal{L}$ whose true accuracy is 100%. In order to exactly understand the learning task in the context of artificially-added noise, it is useful to view the noise as a filter or process that comes between the learning algorithm and the original data. That is, the noise filter takes pairs of the form (e, c) where e is an example and c is the true class of e and maps it to the pair $(n_1(e,c), n_2(e,c))$ where n_1 and n_2 are functions that characterize the noise. This point of view is useful because it makes clear that under the noise there are well defined target descriptions to be learned. Therefore,

> *Task of learning from artificially-noisy data:*
> Let the original data set be $\{(e,c)\}$ and let the data set after the application of the noise filter be $\{n_1(e,c), n_2(e,c)\}$. Then, we say that a learning algorithm $\mathcal{A}$ is able to learn from noise (n_1, n_2) on the given training set $\{n_1(e,c), n_2(e,c)\}$ if the learned model is syntactically equivalent to the model that would be learned by $\mathcal{A}$ on the original data set $\{(e,c)\}$.

The definition above only applies for a given pair of training and test sets. What if we want to answer the more general question: "Is the learning algorithm able to remove the effect of noise on this domain?" To answer this question, the following procedure can be used: Let $t = \{t_1, ..., t_n\}$ represent a sequence of training sets and $T = \{T_1, ..., T_n\}$ represent a corresponding sequence of (noise-free) test sets and let $A = \{a_1, ..., a_n\}$ represent the accuracies obtained by learning on t_i

and testing on T_i. Now, let $A' = \{a'_1, ..., a'_n\}$ denote the accuracies obtained by training on the noisy versions of t and *testing on the noise-free test sets.* In order to know with high certainty that a learning algorithm is successfully circumventing the noise, one would apply the paired, two-tailed t-test between A and A'. If the resulting p value is very low, the one could say that is quite likely that the learning algorithm is satisfying the noisy learning task for artificially-added noise for the given problem. In summary, by testing on noise-free test sets, it is possible to determine (with high certainty) whether the learning algorithm is able to undo the effect of noise.

2.5.2 Learning from naturally-noisy data

In this situation, the noisy data may be about a natural phenomenon which is not fully understood. For instance, the data may contains examples corresponding to patients that got breast cancer and those that did not. The relevant factors for getting breast cancer are not fully understood, so the author of the data set may take a guess at what factors (attributes) would be a help in differentiating the cancerous patients from the others. With respect to these attributes, and some model language such as first-order logic, there may not be a target description that has 100% accuracy over the entire example space. So, even with an infinitely large data source and infinite computation power, it would not be possible to learn a completely accurate class definition for each class *for the given set of attributes.* When learning from naturally noisy data, the task is to find the model in the hypothesis language $\mathcal{L}$ which has the lowest true error of all models in $\mathcal{L}$.

A useful notion in noise-tolerant learning is that of the *Bayes error rate.* For examples x in some example space S, let the most frequently occurring class $c(x)$ for x occur with probability $p(x)$.[2] Then, the optimal strategy is always to guess $c(x)$ for x. However, even this optimal strategy will suffer an error rate of $1 - p(x)$ on x. So, over the entire example space, the Bayes error rate of example space S with respect to the classes $C_1, \ldots, C_k$ is:

$$\text{Bayes-error: } B(S) = \textstyle\sum_x f(x) \times (1 - p(x))$$

where $f(x)$ is the probability of x. Note that for any given hypothesis language, $\mathcal{L}$, even the most accurate model in $\mathcal{L}$ may not have error rate as low as the Bayes error rate.

The situation of learning from naturally-noisy data is even more difficult than that in the previous section since for most naturally-noisy data sets, the

[2]For most domains, there is only one class associated with an example so $p(x)$ is always 1. But for domains for which the given attributes are not sufficient to distinguish all the classes, there may be some examples x for which $p(x) < 1$.

Bayes error rate is not known. Hence, it is not possible to use some statistical test such as the t-test to verify if the learning algorithm has "solved the task of learning from naturally-noisy data" for a given domain. Thus, experiments on naturally-noisy domains are most useful for making comparisons between learning algorithms rather than for checking if the algorithm is able to "solve" the task.

2.6 Evaluation

One important evaluation criterion of empirical machine learning research is to measure the error rate of a learning-algorithm/classification-algorithm pair on examples not seen during training. This allows one to reach conclusions about which learning approach is best for some data set or to reach conclusions about the data itself.

One way to measure the error rate is to form many independent partitionings of the data set. Researchers typically use twenty or thirty partitionings. A partitioning is also called a *trial*. In each partitioning, 70% of the data is reserved for training and the rest is reserved for testing. For each partitioning, the learning algorithm learns class descriptions using the training data. Then, the classification algorithm uses the learned descriptions to predict the class of each test example. The rate at which those predictions disagree with the true class of the test examples is the error rate of that pair of algorithms (the learning algorithm and the classification algorithm) on that partitioning. Such an error rate is calculated for each partitioning of the data and the error rates are averaged to provide a single, average error rate for those algorithms. Typically, the approach of using a test set of examples is much better than measuring the error on the training set because algorithms that overfit noise can easily achieve a zero error rate on the training set yet have very high error rates on new examples. In fact, an algorithm that simply *memorizes* the training examples together with their class labels will achieve zero error rate on the training set but will suffer from high error rates on new examples. The error rate on such examples is called the *off-sample* or *generalization* error rate. Another related measure that is often used instead of the error rate is *accuracy*. Accuracy, A, is related to error, e via $A = 100 \times (1 - e)$. The accuracy that would be obtained on a data set just by always guessing the most frequently occurring class is called the *base accuracy* or *default accuracy*.

If the data set from which the partitionings are to be formed is very small (say 100 examples), then the method of independent partitionings would allocate 70 examples for training. Another method, N-fold cross-validation (e.g. (Weiss & Kulikowski, 1991)) allows larger training set sizes whilst still ensuring that the resulting error rate is unbiased and reflective of the data set as a whole. In N-fold cross-validation the data is divided (as equally as possible) into N partitions (also

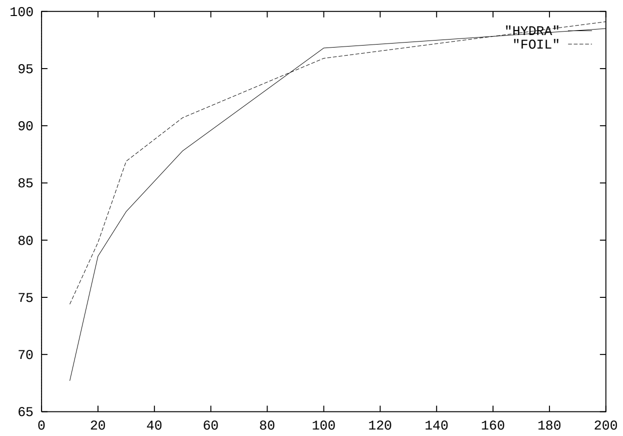


Figure 2.1: A comparison of two learning curves, one per learning algorithm: x-axis: number of training examples; y-axis: accuracy on test examples.

called *folds*). $N-1$ of the folds are used for training and the left-out fold is used for testing. By leaving out a different fold each time, N estimates of the error rate can be obtained. Averaging these yields the final error rate. The advantage of this method over the independent partitionings method is that in this method, if N is 10, 90% of the data is used for training, whereas in the independent partitionings method, only 70% of the data is used. Thus, the N-fold cross-validation approach allows a greater proportion of the data to be used for training whilst also ensuring that each example in the data set is used once for testing. Doing N-fold cross-validation several times will give an even better estimate (an estimate with lower variance) of the true error rate.

For very small data sets, a special form of N-fold cross-validation, called *leave-one-out testing* (or *jack-knifing*; (Weiss & Kulikowski, 1991) is used. Let D denote the size of the data set. Then, leave-one-out testing is equivalent do doing D-fold cross-validation: each fold consists of a single example. The resulting error rates are unbiased estimators of the true error rate but have high variance.

Another important notion in machine learning and cognitive psychology is that of the *learning curve*. The learning curve plots the classification accuracy of a learning-algorithm/classification-algorithm pair as a function of the size of the training set. So, given a data set of 100 examples, one would form say thirty independent partitionings, each consisting of 10 training examples and 90 testing examples. This would produce an average accuracy rate for 10 training examples. Then the same would be done with 20, 40, 60 and 70 training examples. Learning curves are useful because it is often the case that at very large training set sizes, all learning algorithms do equally well. However, at small training set sizes, some algorithms do much better than others. And in “real-world” situations, sometimes the data set sizes are severely limited. In such situations, it is preferable to use the learning algorithm that attains the highest accuracy at small training set sizes (assuming also, that its asymptotic accuracy is comparable to those of other learning algorithms).

In practice, the only way an algorithm could be said to "solve the learning task" for a domain is if the true accuracy of the target model is known for that domain. Then, one could conduct a large number of training and testing trials and if on all those trials, the learned model had the same accuracy as the target model, then one could say that the algorithm is able to solve the learning task for that domain. If there is a model in the hypothesis space that is 100% accurate over the example space (and hence, also over any test set), the learned model would have to achieve 100% accuracy on each trial.

Chapter 3
Background

This chapter describes foundational material necessary to understand the research described in the rest of the thesis. The chapter begins with a description of FOIL (Quinlan, 1990), an algorithm that learns class descriptions consisting of relational clauses. Four limitations of FOIL provide motivation for the research described in this thesis. These are its limited ability to learn from noisy data, inability to learn on data sets containing more than two classes, the small disjuncts problem and the large gain ties problem. FOIL is used in the development of HYDRA (Ali & Pazzani, 1993) which is used to address these limitations.

The chapter also describes the canonical decision tree learning algorithm in the machine learning field: ID3 (Quinlan, 1986; Quinlan, 1993). This work is foundational because it built upon by Buntine (1990) in his application of Bayesian learning theory to decision tree learning. The work described in Section 4.9, in turn, builds on Buntine's work by applying Bayesian decision theory to the learning of class descriptions consisting of rule sets. Finally, the chapter describes foundational material on Bayesian statistics. Bayesian statistics provides the theoretical underpinning for the multiple models approach in which many alternate descriptions are learned for each class in the data. Multiple definitions can be learned for each class from one training set because the training set is usually a small subset of the example space so for each class, there is usually more than one description in the hypothesis space that is consistent with the training set. Chapters 7 and 8 present experiments using the Bayesian approach to learn multiple models. Use of multiple models leads to increases in classification accuracy compared to the accuracy of class descriptions learned by single definition algorithms.

3.1 FOIL: A First-order rule learning algorithm

The input to FOIL is C, the class for which a definition is to be learned, a set of positive and a set of negative examples for C and complete extensional definitions for the set B of background concepts. The output of FOIL is an intensional description of C which is a set of clauses (rules) that conclude for C.

Table 3.1. Pseudo-code for FOIL.

```
FOIL(POS,NEG,Metric,Concept):
 Let POS be the positive examples.
 Let NEG be the negative examples.
 Separate: (begin a new rule)
 Until POS is empty do:
  Let NewRule be the output of Build-rule(POS,NEG,Metric,Concept)
  Remove from POS all positive examples that satisfy NewRule.
 End FOIL

--------------------------------------------
Build-rule(POS,NEG,Metric,Concept)
  Set NewRule to ``Concept if TRUE'' (this rule true for all POS and NEG)
  Until NEG is empty do:
   Conquer: (build a rule body)
    Choose a literal L using Metric
    Conjoin L to body of NewRule.
    Remove from NEG examples that do not satisfy NewRule.
  Return NewRule
  End Build-rule
```

FOIL learns a clause at a time, removing training examples covered by the clause in order to learn subsequent clauses. This is called the "separate and conquer" (Quinlan, 1990) or "covering" (Michalski & Stepp, 1983) approach. FOIL tries to ensure that each clause covers no negative training examples while covering as many positive training examples as possible. The algorithm terminates when each positive training example is covered by at least one clause.

Each clause is learned in the following manner. The clause is initialized to

$$C(V_1 \ldots V_n) \leftarrow$$

which is a Horn clause whose body is the truth-value `true`. Consequently, this clause is true for all positive and negative training examples of C (and in fact, all examples in the example space). FOIL then searches for a literal to add to the body of this clause and it chooses the literal L whose addition to the body of the clause would maximize the literal selection measure. FOIL keeps adding literals in this manner until the resulting clause covers no negative training examples.

For the literal selection measure, ideally we would like a literal that satisfies two criteria: *positive coverage* and *purity*. That is, we would like a literal that covers as many positive training examples as possible (coverage) whilst also covering as few negative examples as possible (purity). This is usually difficult because literals that tend to cover many positive training examples also tend to cover many training examples: they have high *coverage* but low *purity*. Let Pos′ denote the set of positive examples covered by the clause and let Neg′ denote the set of covered negative examples. Then, a simple literal selection measure would be the ratio $\frac{|Pos'|}{|Pos'|+|Neg'|}$ where $|S|$ denotes the cardinality of the set S. For each candidate literal, this quantity would be measured before and after addition of that literal to the body of a clause. The literal that offers the maximal increase would be selected. This would seem to be a reasonable measure but has the disadvantage that a literal that covers just one positive training example and no negatives would receive as high a score as a literal that covers 100 positive and 0 negative examples. Another problem is that a literal that covers 99 positive and 1 negative would be given a lower score than a literal that covers 1 positive and 0 negative examples. It may be a good idea to prefer the literal covering (99,1) because that sole negative example may be excluded later by adding another literal to the body of the clause so that finally, the clause may cover (80,0). However, in order to *ensure* that such a beneficial second literal exists, it would be necessary, in general, to evaluate all pairs of literals. That is, instead of deciding which literal to add, one would need to decide what pair of literals together would yield the greatest increase in the literal selection measure. However, computing the measure for all pairs is very computationally expensive so most algorithms do not do it. They simply consider one literal at a time. This approach is termed *one-step lookahead* and is a member of a more general set of approaches called *greedy search* methods.

Although one might envisage many metrics to correct for these difficulties, it would be preferred if the metric had some basis for supremacy over other metrics. One metric that has appealing theoretical properties and works well in practice is the information gain metric (Quinlan, 1990). Information gain is based on the entropy measure (Shannon & Weaver, 1949) and is formally defined in (Quinlan, 1990) as follows: If Pos and Neg denote the set of positive and negative examples covered by the clause before addition of the literal L and Pos' and Neg' denote the sets after addition of L then the information gain attributable to L is defined as:

$$gain(L) = T^{++} \times (log_2 \frac{|Pos'|}{|Pos'| + |Neg'|} - log_2 \frac{|Pos|}{|Pos| + |Neg|}) \tag{3.1}$$

where T^{++} is the number of tuples[1] in Pos which have one or more extensions in Pos'. Briefly, a tuple becomes extended when the literal being added contains a variable not previously present in the clause. If the literal introduces variables not already in the clause, $T^{++} \geq Pos'$ else $T^{++} = Pos'$. From the gain formula one can see that positive information gain means the proportion of positive and negative examples covered has changed in favor of the positives. In addition, the measure also gives preference to literals that cover many examples. So, this theoretically-based measure trades-off the two desirable factors of purity and coverage.

[1]A tuple is another representation of an example - the only difference is that the length of the tuple may increase under certain circumstances.

Positive training examples of `uncle`:
(walt,jim)

Negative training examples of `uncle`:
(fred,jim)
(sam,jim)
(sarah,jim)

Background knowledge:
sibling(walt, john).
sibling(walt, johnathan).
sibling(sam,karen).

male(walt).
male(sam). male(fred).

parent(johnathan,jim).

Table 3.2: Positive and negative training examples of class `uncle` and definitions of background concepts.

The following example, based on a similar example in Wogulis (1994), illustrates how a tuple becomes extended. The class being learned is *uncle* and this is the definition which FOIL will eventually learn:

$$\text{uncle(X,Y)} \leftarrow \text{male(X),sibling(X,Z),parent(Z,Y)}$$

Assume that a positive training example for `uncle` is `(walt,jim)` and a negative training example is `(fred,jim)`. The background concept `sibling` encodes the fact that `walt` has two siblings (`john` and `johnathan`) and `fred` has one sibling (`sam`). Background concept `male` encodes the fact that all the most of the individuals in this story are males. Therefore, the situation is as shown in Table 3.2. FOIL starts with the clause with the empty body:

$$\text{uncle(X,Y)} \leftarrow$$

This clause covers the positive example and both negative examples. FOIL maintains two sets of tuples: positive tuples and negative tuples. The set of positive tuples is initialized to the set of positive training examples. The set of negative tuples is initialized to the set of negative training examples. As literals are added to the clause under construction, these sets are updated. When the negative set becomes empty, the clause is considered complete. For this illustration, the sets are initialized as follows:

Positive tuples:	(walt,jim)
Negative tuples:	(fred,jim), (sarah,jim), (sam,jim)

Assume that FOIL decides to add the literal `male(X)` because it has highest information gain. Then, the clause becomes

$$\text{uncle(X,Y)} \leftarrow \text{male(X)}$$

Addition of the literal `male(X)` excludes the negative tuple `(sarah,jim)` since `sarah` is not a male.

In technical terms, to check if the tuple `(sarah,jim)` should be included, a binding for this tuple is generated ($\{X = sarah, Y = jim\}$) and the variables in the clause are replaced with their values as specified by the binding and the resulting body of the clause becomes:

$$\text{male(sarah)}$$

Then, the truthfulness of this term is checked against the background knowledge. Since the term `male(sarah)` is not in the background knowledge, it is concluded that the truthfulness of this term is `false` so the list of negative tuples is updated to exclude the tuple `(sarah,jim)`.

Now the set of positive tuples becomes `((walt,jim))` and the set of negative tuples becomes `((fred,jim), (sam,jim))`. Now assume that the literal `sibling(X,Z)` has highest information gain. So, the clause now becomes:

$$\text{uncle(X,Y)} \leftarrow \text{male(X),sibling(X,Z)}$$

The newly added literal contains a variable (`Z`) not previously present in the clause so this will cause the tuples to become extended. The positive tuple `(walt,jim)` corresponds to values of variables `X` and `Y` respectively. But there are two values of `Z` consistent with these values of `X` and `Y` and the clause

$$\text{uncle(X,Y)} \leftarrow \text{male(X),sibling(X,Z)}$$

Those values are `john` and `johnathan`. Thus, the tuple `(walt,jim)` will give rise to two *extended* tuples: `(walt,jim,john)` and `(walt,jim,johnathan)`. There is no binding for the negative tuple `(fred,jim)` that is consistent with the clause so the tuple is removed from the set of negative tuples. Intuitively, one can check that `fred` has no sibling, so the tuple does not satisfy the new clause. The other negative tuple, `(sam,jim)` has one value of Z consistent with the clause and that value is `karen`. So, it is extended to become `(sam,jim,karen)`. The ability to introduce new variables allows subsequent literals to test attributes of the objects (e.g. `john` or `johnathan`) corresponding to the values of the newly added variable. This gives first-order clauses a greater expressiveness than attribute-value rules.

At this point, the clause

$$\text{uncle(X,Y)} \leftarrow \text{male(X),sibling(X,Z)}$$

is true for two extended positive tuples:
(walt,jim,john) and (walt,jim,johnathan) and a single extended negative tuple (sam,jim,karen). So FOIL searches for a condition it can add that will yield a consistent binding for the positive tuples and none for the negative tuple. Assume that the literal parent(Z,Y) has the highest information gain so it is added. Now the clause becomes:

$$\text{uncle(X,Y)} \leftarrow \text{male(X),sibling(X,Z),parent(Z,Y).}$$

This clause is not true for the extended negative tuple (sam,jim,karen) so the clause is deemed complete. Since all the positive training examples (there is only one!) are covered by the learned clauses, no more clauses need to be learned and training is finished.

The set of possible literals from which FOIL chooses a literal to add to a clause is formally defined as follows. Let C be the class for which the clause (with head $C(V_1 \ldots V_{n_C})$) is being learned and let n_C be the arity of C. Let $\{B_i\}$ be the background concepts with arities $\{n_i\}$. Let W denote the set of variables in the clause so far (this set is initialized to $\{V_1 \ldots V_{n_C}\}$). Then, FOIL will consider all literals of the form $L(W_1 \ldots W_m)$ where L denotes some predicate B_i, m is the arity of B_i and each W_i is drawn from the set $W \bigcup \{V_{i,1}, \ldots, V_{i,n_i}\}$ with the restriction that at least one of the variables in the newly added literal L is a variable that was present before addition of L. This search space is large so in practice a limit is set to the number of "new" variables that a literal can introduce. For example, if FOIL is searching for a literal to add to the clause

$$\text{uncle(X,Y)} \leftarrow$$

and the background predicates are male(_) and sibling(_,_) then FOIL will consider the following set of literals: male(X), male(Y), sibling(X,X), sibling(X,Y), sibling(Y,X), sibling(Y,Y). This example illustrates the need for two additional types of constraints that could be used to prune the search space. The first is the commutativity constraint for a binary predicate P which means that $P(V_1, V_2) = P(V_2, V_1)$. Applying this constraint to sibling would mean that only one of the literals (sibling(X,Y), sibling(Y,X)) would need to be checked because we know that sibling(X,Y) is true if and only if sibling(Y,X) is true. The second is the *unique variables* constraint which will eliminate the need for generating literals such as sibling(V,V) because we know that the semantics of the concept sibling excludes anyone from being their own sibling. FOIL also uses other admissible pruning heuristics (Quinlan, 1990) to limit its search.

Classification: After having learned a set of clauses, FOIL is used for classification as follows. FOIL classifies a test example by checking to see if that example satisfies any of the learned clauses. If it does, it is classified as belonging to the class otherwise FOIL uses the closed world assumption (Lloyd, 1984) and classifies the example as belonging to the other class.

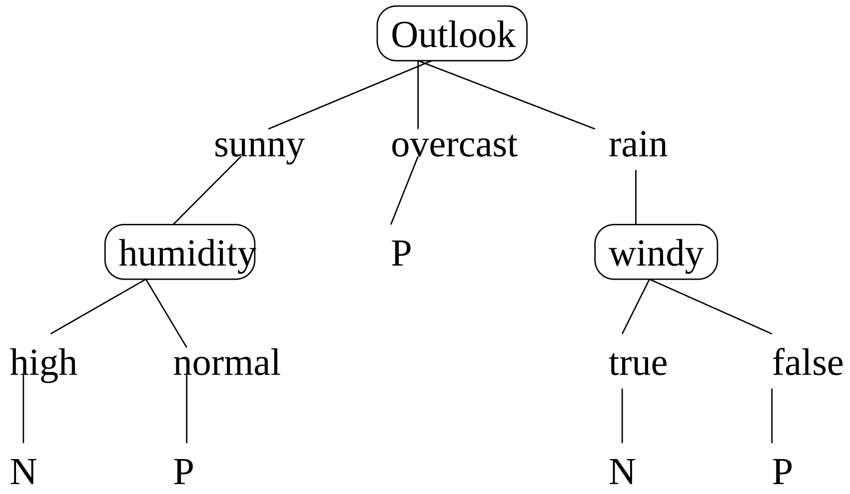


Figure 3.1. A simple decision tree

3.2 Decision trees

Although most of this thesis is concerned with learning models of the data in which the model consists of a set of class descriptions consisting of rule sets, some of the work is related to learning decision trees (e.g. ID3 - (Quinlan, 1986); CLS - (Hunt et al., 1966)) A *decision tree* is a single structure used to discriminate all the classes. Given an ordered sequence of attributes and an example (a list of values for those attributes), and given a set of classes, the task of the decision tree is to decide the class to which that example belongs.

The mechanism of classification in a decision tree is exceedingly simple and is best illustrated by an example (this example is taken from (Quinlan, 1986)). Assume that the task is to classify a day into one of two classes: `P` and `N`. Assume that each day is described by the following sequence of attributes: `(Weather--outlook, Temperature, Humidity, Windy)`. Assume that the decision tree has been learned as shown in Figure 3.1. In the figure, identifiers in boxes correspond to the names of attributes. Assume that the test example to be classified is `(sunny, hot, normal, true)`. To classify this example using the decision tree, the value of the attribute corresponding to the root of the tree (`Outlook`) is tested. For this example, the value of `Outlook` is `sunny` so the subtree corresponding to the value `sunny` is now checked. This subtree has at its root, the attribute `humidity`, so the value for that attribute is checked. In the test example, the value of that attribute is `normal` so the subtree corresponding to that value is checked. But that subtree is just the *leaf* `P` and so the classification procedure terminates and class `P` is predicted for this test example.

Decision trees are used in this thesis when dealing with the large gain ties problem. They are also necessary to understand how the application of Bayesian statistics to decision trees by Buntine (1990) is adapted for Bayes-based classification using class descriptions consisting of sets of rules.

3.3 Bayesian statistics for modeling uncertainty

This section presents foundations of Bayesian statistical theory that underlies the Bayesian approach to learning and classification. The Bayesian approach differs from classical statistics in two major ways:

- **Interpretation of probability:** The interpretation of the probability of a random boolean variable in classical statistics is linked to the long term expectation of how often that boolean variable will be `true`. The probability of a proposition or random variable, is a fixed quantity, independent of the observer. Different observers may make different *estimates* for the probability of the same proposition, but the "true" probability of the proposition does not vary from observer to observer. By contrast, Bayesian statistics allows another interpretation of probability. *A Probability can be regarded as a degree of belief.* Therefore, in Bayesian statistics, it is possible to make statments such as "The probability that the burglar's vehicle was a Ford pickup given the poor viewing conditions is 0.9." This statement of degree of belief does not make sense in classical statistics because probabilities in classical statistics are independent of the viewer: either the truck is a Ford pickup or it is not. Probabilities are not assigned to single events, only to classes of events. However, the ability to attach probabilities to single events provides powerful capabilities: namely, the ability to manipulate degrees of belief using the axioms common to classical and Bayesian probability theory. In particular, we can use model uncertain evidence (the burglary example above) using probabilities which have well understood properties. Uncertain conclusions:

  > "given what we know of this congressman's voting record on welfare measures (known with complete certainty), he is probably (0.95) a Republican."

  can also be modeled with subjective probabilities. Note that this statement could also not be made in classical statistics which does not assign probabilities to single events. Such use of probabilities allows a classification system to signal its uncertainty about classifying a test example. It also allows the system to use uncertain evidence and model evidence combination using the axioms of probability theory. Pearl (1988) (1st chapter) gives an excellent presentation of the non-intuitive conclusions of evidence combination methods (such as those used in MYCIN, (Buchanan & Shortliffe, 1984)) that do not obey the probability axioms.

- **Use of priors:** Because Bayesian probabilities can be used to model degrees of belief, it follows that the "probability" of a proposition can vary as more evidence is assimilated. A special case of this occurs when no evidence has been gathered. The probability of some proposition given that "no" evidence has been gathered is called the *prior probability* of that proposition. The probability of that proposition after evidence is called the *posterior probability* of the proposition. This is often written $p(C|E)$ where C denotes the proposition and E denotes the evidence. Prior probability almost always just means "relatively prior" - that is that there is no immediate evidence for the proposition, but general knowledge of that proposition allows us to make some prior (subjective) estimate of its probability. The prior distribution of hypotheses is frequently used to model the bias or background knowledge provided by an expert. In this paradigm, one can explain differing classifications by two experts using the same evidence as due to their differing prior distributions (corresponding to different biases and different background knowledge).

3.3.1 Bayesian statistics in classification

In the Bayesian approach to classification, given a test example, one should make the classification whose expected posterior probability is the greatest with the expectation being taken over the entire hypothesis space. This process is called *Bayes averaging*. This policy has the following intuitive justification: given a test example and a class, because the true hypothesis is unknown, the test example should be assigned to the class which is indicated to the greatest degree by the all the hypotheses in the hypothesis space. However, some hypotheses are going to be much more probable given the training data we have seen, therefore they should be given greater weight when computing the expectation. Mathematically, we should assign the test example x to the class c with the maximal expected posterior probability given the test example x, training examples $\vec{x}$ and their associated class labels $\vec{c}$:

$$E_{\mathcal{H}}(p(c|x, \vec{x}, \vec{c})) = \sum_{h \in \mathcal{H}} (p(c|x, h) \times p(h|\vec{x}, \vec{c})) \tag{3.2}$$

In this notation, based on Buntine (1990), $E(..)$ denotes the expectation operator, c denotes a class, h denotes a hypothesis and $\mathcal{H}$ denotes the hypothesis space. $p(h|\vec{x}, \vec{c})$ denotes the posterior probability of the hypothesis h given the training data. This posterior probability is used as the voting weight of that hypothesis during classification. The term, $p(c|x, h)$ can be thought of as the degree to which the hypothesis h endorses class c for the test example x. Therefore, Bayes averaging can be thought of as weighted voting to decide what class to assign to the example. The vote is taken over the entire hypothesis space and the weight of the hypothesis is its posterior probability $p(h|\vec{x}, \vec{c})$. Bayes averaging, is not simple weighted voting, however, because it also involves another term: $p(c|x, h)$ - the degree to which h endorses c for x. Equation 3.2 is only true when $\mathcal{H}$ is a finite or enumerable set. If not, as is the case with hypothesis languages that permit continuous variables (such as classification accuracy) to be associated with clauses or decision tree leaves, one needs to integrate over all continuous variables (pg. 99 in (Buntine, 1990)).

Intuitively, the posterior probability of a hypothesis should depend on the prior probability of the hypothesis and also on the amount of evidence for the hypothesis. To compute a posterior probability, Bayes' rule (also called Bayes' theorem) is used:

$$p(H|E) = \frac{p(E|H)p(H)}{p(E)} \propto p(E|H)p(H) \tag{3.3}$$

The $p(E)$ term, the probability of the evidence (i.e. the training data) can be ignored for our purposes because it will affect all hypotheses equally and we are only interested in the *relative* weights (posterior probabilities) of the hypotheses. The term $p(H|E)$ is the posterior probability of the hypothesis H given the evidence E and $p(E|H)$, the posterior probability of the evidence given the hypothesis, is referred to as "the likelihood of the hypothesis." It measures how likely H is to produce[2] the evidence E. For classification, it is used in calculating how likely a hypothesis is to generate a set of training examples. $p(H)$ is the prior probability of the hypothesis H. Applying Bayes rule to calculate the posterior probability of a hypothesis given a set of training examples, $\vec{x}$, and class labels, $\vec{c}$, yields:

$$p(h|\vec{x}, \vec{c}) \propto p(\vec{x}, \vec{c}|h)p(h) \tag{3.4}$$

$p(h)$ is the prior probability of the hypothesis h and it is determined by the prior distribution of the hypotheses. The uniform distribution is used if there is no *a priori* reason to favor one hypothesis over another. Priors from such a distribution are called *non-informative priors*. In practice, machine learning researchers often use a distribution that assigns higher *a priori* probabilities to hypotheses that have syntactically small descriptions with respect to the given set of background concepts and hypothesis language. This encodes the widely held belief that simpler hypotheses are more probable *a priori* than more complex ones.

In order to apply this theory, a way of calculating the likelihood function, $p(\vec{x}, \vec{c}|h)$, from data is needed. Buntine (1990) derives the following form used to approximate the posterior of a decision tree:

$$p(T|\vec{x}) \propto p(T) \times \prod_{k=1}^{V} \frac{B(n_{1,k} + \alpha, \ldots, n_{C,k} + \alpha)}{B(\alpha, \ldots, \alpha)} \tag{3.5}$$

[2]Sometimes it is useful to think of conditional probabilities of evidence conditioned on hypotheses, $p(E|H)$, as representing *causal* information. Conversely, probabilities of hypothesis conditioned on evidence, $p(H|E)$, can be thought of as representing *diagnostic* information.

where $p(T)$ is the prior of the tree, B is the beta function, V is the number of leaves of the tree, α is a parameter, C is the number of classes in the learning task, and $n_{i,j}$ denotes the number of training examples of class i at the j-th leaf of the tree. The C-dimensional beta function is defined as

$$B_C(m_1,\ldots,m_C) = \frac{\prod_{i=1}^{C}\Gamma(m_i)}{\Gamma(m_1+\ldots+m_C)}$$

and the gamma function is defined for integers as $\Gamma(n) = (n-1)! = 1\times 2\times \ldots \times (n-1)$ Hence,

$$B(n_{1,k}+\alpha,\ldots,n_{C,k}+\alpha) = \frac{n_{1,k}!\times\ldots\times n_{C,k}!}{(N+C\alpha-1)!} \tag{3.6}$$

where N is the total number of training examples (of all classes) at leaf j.

To summarize the preceding sections, in order to decide which class to assign to a test example using equation 3.2 one needs to know two quantities: i) the posterior probability of each tree given the training examples and ii) the degree to which the tree endorses each class for the test example x. In order to compute the degree of endorsement and posterior probability of a tree, it is sufficient to record the number $n_{i,j}$ of training examples of class i at leaf j. These numbers are then used to compute the degree of endorsement and posterior probability as follows:

- **Degree of endorsement:** This is given by

$$p(c|x,h) \cong \frac{n_{i,j}+\alpha}{N_j+C\alpha} \tag{3.7}$$

  where n_{ij} is the number of training examples of class c at the leaf j of the tree h reached by x, C is the number of classes in the learning task and N_j is the total number of training examples of all classes at leaf j, given by $N_j = \sum_{i=1}^{C} n_{i,j}$. α is a parameter that can be thought of as a weight for the prior probability $p(c|x,h) = \frac{1}{C}$ at the leaf. If there are no examples at the leaves ($n_{ij} = 0$ and $N_j = 0$), expression 3.7 becomes $\frac{1}{C}$. Note that this estimate (Good, 1950; Cestnik & Bratko, 1991) is different from the maximum likelihood estimate ($\frac{n_{i,j}}{N}$) used in classical statistics. The advantage of this estimate is that it does not assign the maximum possible value of 1 to a rule or leaf covering just 1 positive and 0 negative examples.

- **Posterior Probability:** By equation 3.5, the posterior probability of a decision tree is the product of the prior probability and a product of beta-functions, one for each leaf. In order to compute the value of the beta for a leaf, one just needs to use equation 3.6. For the prior distribution, commonly used distributions (e.g. as in (Buntine, 1990)) are the uniform distribution or the distribution that favors less complex trees: $p(h) = exp(-V)$ where h is a tree and V is the number of leaves in h.

3.3.2 Bayesian statistics in learning

Besides its use for classification, Buntine has observed that the posterior probability can also be used as a metric during learning. The metric is used during learning decision trees by choosing to add the decision node to a tree such that the posterior probability of the new tree is maximized. For learning a binary tree from two classes, the following instantiation of equation 3.5 defines a gain metric:

$$pr_2(n_{11}, n_{21}, n_{12}, n_{22}) = p(T) \times \frac{B(n_{11} + \alpha_1, n_{21} + \alpha_2)}{B(\alpha_1, \alpha_2)} \times \frac{B(n_{12} + \alpha_1, n_{22} + \alpha_2)}{B(\alpha_1, \alpha_2)} \tag{3.8}$$

where n_{11} and n_{21} respectively denote the numbers of positive and negative examples in the left branch of the node and n_{12} and n_{22} denote the numbers for the right branch. $p(T)$ denotes the prior probability of the tree resulting from addition of the decision node. This gain metric is called the Bayes metric. In Section 4.9 this material is built upon to apply Bayesian learning theory to learning class descriptions consisting of rule sets.

3.4 Discussion

Descriptions for classes that have compact first-order definitions but large attribute-value definitions can be learned easily if the learning algorithm can build first-order definitions. FOIL is an algorithm that can build first-order definitions given positive and negative examples of a nominated class. It is limited to learning tasks with two classes: the nominated class and its complement.

FOIL cannot be directly used for multi-class problems because it assumes the data consists of just two classes and it only learns a description for the class labeled as the "positive" class. The one-per-class approach will be used in this thesis for learning class descriptions for multi-class learning tasks. In the one-per-class approach, a class description is separately learned for each class. Other approaches to multi-class tasks do not build separate definitions for each class. ID3 (a decision tree learning algorithm - (Quinlan, 1986)) for example, builds a single definition that discriminates examples of all classes.

Besides its limitation for multi-class problems, FOIL does not learn accurate class descriptions if there is noise in the training data. When learning from noisy examples, approaches that only consider rules that cover only examples of the positive class lead to the learning of overly complex definitions. Such definitions have poor classification accuracy on test examples as compared to more compact definitions that do not over-fit the noise (i.e. approaches that permit the learning of clauses that may cover a few training examples that appear to be of other classes). This latter approach makes sense if the data contains class noise because the apparently negative training examples may actually be from the positive class. One of the central tasks when learning from noisy examples is to decide how to prevent over-fitting of the noisy data.

Bayesian statistics offers a principled mechanism for modeling uncertainty in classification. It does this by modeling the degree of belief of the classifier by a "subjective probability." Modeling uncertainty and combining evidence using subjective probabilities has advantages over ad-hoc evidence combination schemes such as those used in MYCIN (Buchanan & Shortliffe, 1984). However, there is still considerable debate (e.g. (Cheeseman, 1986)) over the merits of modeling uncertain inference with subjective probabilities over other approaches such as fuzzy logic (Zadeh, 1980), the Dempster-Shafer approach (Shafer, 1976) and Nilsson's probabilistic logic (Nilsson, 1986).

Bayesian statistics can be applied to classification by choosing the class that has the maximum expected posterior probability. In the Bayesian approach, rather than modeling the data using a single model (hypothesis), all models in the hypothesis space are considered. Evidence for each class is combined from all the models in the hypothesis space weighted by the posterior probability of the hypothesis given the training examples. The posterior probability of a hypothesis depends on two factors: the prior and the likelihood of the hypothesis given the training data. Hypotheses that are more likely given the training examples will have higher posterior probabilities. Weighting only by likelihood leads to preferring hypotheses that overfit the training data. By incorporating the prior that gives a lower value for more complex hypotheses that are likely to be overfitted representations of the training data, the posterior trades off two goals: i) that the hypothesis should be likely given the data, and ii) that there should be a preference towards simpler hypotheses. The Bayesian approach can also be used during learning by using posterior probability as a gain metric and searching for models with high posterior probabilities.

Chapter 4
HYDRA

This chapter presents HYDRA (Ali & Pazzani, 1993) the noise-tolerant, multi-class learning algorithm used to conduct the research presented in this thesis. The first three sections of the chapter deal with classification, knowledge representation and learning in HYDRA. The next section explains the experimental protocol used for experiments described in this thesis. The last five sections investigate alternative solutions to issues such as "how to classify an example that satisfies rules of more than one class," "how to combine evidence from more than one rule for a class," and "how to classify an example that "nearly" satisfies a rule."

HYDRA is derived from FOCL (Pazzani & Kibler, 1991; Pazzani et al., 1991) which added to FOIL the ability to use background knowledge defined in terms of partially correct rules during learning. HYDRA extends FOCL in three important ways:

1. HYDRA learns a class description consisting of a set of rules for *each* class so that each description can compete to classify test examples. In (Ali & Pazzani, 1993) we show that this allows HYDRA to learn more accurate descriptions from noisy data than FOIL. Learning a description for each class allows the boundary between any two classes to be modeled twice. It is our hypothesis that on noisy, real-world tasks this allows more accurate classifications to be learned than learning a single, discriminating description for all the classes which only models each boundary once. The situation where rules of more than one class are satisfied for a test example is called a *conflict.*
2. HYDRA attaches a measure of rule reliability to each rule. These rule reliabilities are used to resolve conflicts. The method used to resolve such conflicts is called the *conflict-resolution strategy.* If the attached reliability estimate is a probability estimate, such rules are called *probabilistic rules.*
3. The metric used by HYDRA (ls-content) to rank literals during learning trades off rule coverage against rule purity more in favor of coverage than is done by the information gain metric. A rule is pure if it only covers training examples of one class. A rule has high coverage if it covers many training

examples of the class for which it is being learned. Learning totally pure rules is not a good idea if there is class noise in the data because some examples that are labeled as belonging to other classes will in fact be examples of class C. Only allowing pure rules to be learned will lead to the learning of many rules, each covering a few training examples of class C. Rules that cover few examples tend to be inaccurate. Therefore if there is class noise in the data, an algorithm that only allows pure rules will end up learning a large number of inaccurate rules and will have poor classification accuracy. Table 4.4 in Section 4.9 compares the information gain metric which learns many pure rules against the Bayes gain metric which learns fewer rules, that are not as pure. The superior overall accuracy of the Bayes gain metric is support for the learning of fewer, slightly impure rules in noisy data sets.

4.1 Classification

In understanding how HYDRA classifies a test example, the notion of *representative rule* is useful. The representative rule, $r(R, x)$, of a class description R (a set of rules that all conclude for the same class) and a test example x, is the rule with the highest reliability chosen from the subset of R that were satisfied by x. A representative rule is selected for each class for which x satisfied at least one rule. The test example is then classified to the class whose representative rule has the highest reliability. When a test example satisfies no rule of any class, HYDRA guesses the most frequent class. In the representative rule approach, if more than one rule from a rule set is satisfied, that is not taken as greater evidence that the test example belongs to this class. Our experiments have shown (Ali & Pazzani, submitted) that this approach works well when the target description is a compact DNF form with respect to the given set of attributes.

For example, consider the task of predicting whether a student X has satisfied a course requirement for course C. Assume that the target description for class `satisfies-course-requirement` is:

satisfies-course-requirement(X,C) $\leftarrow$ done-course(X,C,uci)
satisfies-course-requirement(X,C) $\leftarrow$ done-course(X,C,Y), $\neq$(Y,uci)

Now assume that student `pedro` has done the course both at UCI and at another university. Therefore, both rules will be satisfied for `pedro`. For classes such as these, the fact that both rules are satisfied should not be taken as *greater* evidence that X is a member of class `satisfies-course-requirement`.

4.2 Knowledge Representation

Rules learned by HYDRA are accompanied by a measure of rule reliability. One type of rule reliability measure is LS - the degree of logical sufficiency that the body of a rule has for the head of the rule. Normally, the body of a rule is taken to be totally sufficient to conclude the head of the rule. LS is a generalization of this notion. The degree of logical sufficiency (Duda et al., 1979) of the *j-th* rule of the class description for the i-th class is defined as follows:

$$ls_{ij} = \frac{p(rule_{ij}(\tau) = true \mid \tau \in Class_i)}{p(rule_{ij}(\tau) = true \mid \tau \notin Class_i)} \tag{4.1}$$

where τ represents an arbitrary example. Muggleton *et al.* (1992) show that training coverage (the number of training examples of $Class_i$ covered by a rule for $Class_i$) is a more reliable indicator of the true accuracy of a rule than apparent accuracy measured from the training data. The accuracy of a rule is defined as $p(Class_i|rule_{ij}(\tau) = true)$. Accordingly, we choose to use LS as a reliability measure because it has the flavor of measuring both coverage and accuracy $(p(Class_i|rule_{ij}(\tau) = true))$. The true accuracy of a rule is the accuracy of that rule on the entire example space. The apparent accuracy is often an overestimate of the true accuracy because learning algorithms expressly learn rules that cover few or no training examples of other classes.

We use the Laplace estimate (Kruskal & Tanur, 1978) for a probability to compute the ratio of probabilities in the definition of LS. The Laplace estimate for the probability of a random variable taking on a particular value, $p(X = v)$, given that X has been observed to take on value v on f occasions from a sequence of T trials is

$$p(X = v) = \frac{f+1}{T+k} \tag{4.2}$$

k denotes the size of the set of possible values that X can take. Therefore, replacing the probabilities in the definition of LS by their Laplace estimates yields

$$ls_{ij} = ls(p, n, p_0, n_0) \approx \frac{(p+1)/(p_0+2)}{(n+1)/(n_0+2)} \tag{4.3}$$

where p_0 and n_0 respectively denote the numbers of positive and negative training examples (of $Class_i$) in the entire data set and p and n denote the numbers of examples covered by the rule. Thus, LS is a measure of the proportion of the training examples of $Class_i$ for which the rule is true divided by the proportion of the training examples of other rules for which the rule is true.

Another candidate for a rule reliability measure is the Laplace estimate of the accuracy of a rule. Given that the rule is true for N training examples and that n of those are from $Class_i$, the maximum likelihood estimate of the rule accuracy,

$p(Class_i|rule_{ij}(\tau) = true)$, is n/N whereas the Laplace estimate is $(n+1)/(N+2)$.[1] Unfortunately, the maximum likelihood estimate of the accuracy of a rule covering 1000 positive training examples and 0 negative training examples is the same as that for a rule covering 1 positive training example and 0 negative training examples. This is undesirable because rules that cover few training examples are less accurate on test examples. The Laplace estimate, however assigns a value of $\frac{1000+1}{1000+2} = 0.999$ to the accuracy of the first rule, and only $\frac{1+1}{1+2} = 0.667$ to the accuracy of the rule covering just 1 positive and 0 negative examples. Section 4.7 shows how using LS or Laplace accuracy affects classification accuracy.

4.3 Learning in HYDRA

HYDRA uses the same separate and conquer strategy used in FOCL and FOIL. However, HYDRA uses the ls-content metric to rank candidate literals during learning rather than using information gain. Ls-content is defined as follows. Assume the j-th rule is being learned for $Class_i$ and it covers p positive training examples (of $Class_i$) and n negative examples (examples of other classes) and p_j and n_j respectively denote the number of positive and negative examples remaining after the first $j-1$ learned rules. Then, ls-content is defined as the product of the reliability of the rule (LS) and the coverage (p) of the rule:

$$\textit{ls-content}(p, n, p_j, n_j) = ls(p, n, p_j, n_j) \times p$$

where $ls(p, n, p_j, n_j)$ denotes the ratio of probabilities in the definition of LS (Equation 4.1) computed from the subset of the training data not covered by previously learned rules. Using this metric HYDRA adds the literal whose addition to the rule maximizes gain with respect to ls-content. When there are no literals that cause an increase in ls-content or if the rule no longer covers any negative examples, HYDRA removes examples covered by the rule from the training set and learns subsequent rules on the remaining examples. The ls-content metric leads to the learning of rules that more often cover negative training examples (compared to information gain) because as learning progresses towards the end of a rule, it can happen that there is no literal with positive gain with respect to ls-content.

Now consider what is required for positive gain with respect to the ls-content gain metric. Let p and n denote the numbers of examples covered before the addition of some literal $\mathcal{L}$ and let $p\prime$ and $n\prime$ denote the numbers after addition of $\mathcal{L}$. Then, $\mathcal{L}$ yields positive gain if

$$ls(p\prime, n\prime, p_j, n_j) \times p\prime > ls(p, n, p_j, n_j) \times p$$

[1] We use "2" in the denominator rather than k, the number of classes, because in learning a description for $Class_i$, all examples of other classes are grouped together. Thus, during learning, there are only two "virtual" classes.

Rearranging, we have positive gain if

$$ratio_{ls} \times ratio_{coverage} > 1$$

where $ratio_{ls} = \frac{ls(p\prime,n\prime,p_j,n_j)}{ls(p,n,p_j,n_j)}$ and $ratio_{coverage} = \frac{p\prime}{p}$. Addition of a literal usually increases the purity of the rule, ($ratio_{ls} > 1$) but decreases the coverage ($ratio_{coverage} < 1$). Therefore, positive gain only results if the increase in purity offsets the decrease in coverage. On noisy data sets, information gain tends to learn pure rules with small coverage. LS-content will only do so if the increase in purity is great enough to offset the decrease in coverage. HYDRA learns more accurate rules using ls-content than it does using information gain (Ali & Pazzani, 1993) for noisy data sets because ls-content puts a *relatively* greater emphasis on coverage and less on purity.

In addition to being used during learning, LS is also used during classification. After all the rules have been learned, HYDRA forms an estimate of the degree of logical sufficiency ls_{ij} associated with rule j of concept i. After learning, the LS is estimated from the entire set of training examples, not just from the examples uncovered by previously learned rules as used by Equation 4.3. The motivation for this is as follows. Consider a rule learned towards the end of the learning process. Most of the training data has been covered by previously learned rules and so is unavailable for learning. Thus the rule may cover just a few of the remaining training examples but it may cover many examples of the set that were removed by previously learned rules. Using those examples in order to estimate LS allows a more reliable estimate. Finally, it may happen that once the rule is evaluated over the entire training set that the LS of the rule becomes less than 1. Such rules are discarded.

Unknown values are handled as follows. During learning, an unknown value for a component in an example is signified by the symbol `:unknown`. This symbol will match any variable in any intensional description. So, for example, if the rule is `class1(X,Y)` ← `foo(X,Y).` and the example of `class1` is the ground literal `(adam,:unknown)` then this literal will satisfy the rule if it can find a binding for the variable corresponding to the non-unknown variable. If the literal satisfies the rule, `X` will be bound to value `adam` and variable `Y` will be bound to value `:unknown`. This functionality is implemented by altering the definition of the background predicate `foo`. Specifically, the definition of `foo` is wrapped by a LISP function which checks to see if any of the arguments in the ground literal are the special value `:unknown`. The LISP function returns the value `true` if it can successfully bind the other ground values against the true definition (extensional or intensional) of `foo`. The current implementation is very simple and does not perform consistency checking. So, for instance, the following rule `class1(Y)` ← `=(Y,3),=(Y,4)` will succeed if `Y` is bound to the value `:unknown` since HYDRA currently does not impute a value for `Y` when `Y` is bound to the special value `:unknown`. It simply maintains a binding list which in this case binds variable `Y` to value `:unknown`. Then, the literal `=(Y,4)` will successfully match agains the ground literal `(:unknown,4)` since this check is only made locally and a value for `Y` was not imputed.

4.4 Experimental protocol

For all the experiments presented in this thesis, domains were chosen randomly from the UCI repository of machine learning databases (Murphy & Aha, 1992) ensuring that at least one domain from each of the major groups (molecular biology, noisy natural data sets, ...) was chosen. These include molecular-biology domains (2), medical diagnosis domains (7), relational domains (6 variants of the King-Rook-King (KRK) domain, Muggleton *et al.* (1992)), a chess domain with a "small disjuncts problem" (KRKP; Holte *et al.*, (1989)), and attribute-value domains (4 LED variants and the tic-tac-toe problem).

The data sets in Table 4.1 are grouped in four groups: the first group contains noise-free training data from artificial concepts (for which we know the true class descriptions), the second group contains noisy data from artificial concepts the third contains data sets from molecular biology domains and the final group contains probably noisy data from medical diagnosis and other "real world" domains. The domains in the last group are sorted so that those with the highest single model accuracies appear first. These divisions simplify analysis because sometimes an algorithm will work well for one group but not well for another. Base accuracy (also called default accuracy) is the accuracy that would be obtained by always classifying an example to the most frequent class in the training data.

For most of the domains tested in this thesis, we used thirty independent trials, each time training on two-thirds of the data and testing on the remaining one-third. The same set of thirty testing and training sets were used for all the algorithms presented in this thesis. The exceptions to this are the DNA promoters domain for which leave-one-out testing has traditionally been used and we follow this tradition to allow comparability. Other exceptions are trials involving the King-Rook-King domain. For this domain, the training and test sets are independently drawn (rather than being mutually exclusive) from the set of all 8^6 board configurations. Whenever possible we tried to test learned models on noise-free examples (including when learning on noisy variants of the KRK and LED domains) but for the natural domains we tested on possibly noisy examples. The "large" variant of the Soybean data set (Murphy & Aha, 1992) was used and the 5-class Heart data set variant was used.

The training set sizes were selected to be compatible with previous research whenever possible (e.g. Tic-Tac-Toe, <u>K</u>ing-<u>R</u>ook-<u>K</u>ing-<u>P</u>awn). For simple data sets such as Mushroom and LED, we chose a small training set size to allow differences between algorithms to come out. At large training set sizes, all algorithms perform equally well on these simple data sets. This setup allows maximum comparability with previous work allowing for the variation in accuracy that may occur due to the use of differing sets of training sets.

4.5 Experiments with HYDRA

The following five sections investigate alternative solutions to issues such as "how to classify an example that satisfies rules of more than one class." The experiments presented in those sections are designed to answer the following questions:

1. What should be done if more than one rule from a class is satisfied? How should the probability or evidence for the class be increased? This has been referred to as the "rule overlap" problem (Segal & Etzioni, 1994).
2. What to do if rules of more than one class are satisfied? In this thesis, if this situation occurs, the reliability weights associated with each rule are used to resolve this conflict situation. Two forms of reliability weights are compared in Section 4.7. These are the degree of logical sufficiency of the rule (LS) and the Laplace accuracy of the rule.
3. What to do if the example satisfies no rules of any class? The two approaches that are compared are the "guess the most frequent class" approach and the *partial clauses* approach. In the partial clauses approach, an attempt is made to try to find rules that were "nearly satisfied" by the example.
4. What search metric to use during learning? Given that we are doing greedy hill-climbing learning of rules in a general to specific manner, what gain metric is best? Section 4.9 compares three metrics: information-gain, gain in posterior probability (the Bayes metric) and gain in ls-content - which is roughly the product of the purity and coverage of the rule.
5. An important problem in learning is to avoid overfitting noisy data. How sensitive are accuracies obtained on the data sets to metrics that try to avoid overfitting?

4.6 The rule-overlap problem

In this section we will address the question of how to compute the posterior probability of a class given that more than one rule from its description was satisfied for a test example. This is the problem of "evidence combination" (within a class description). Chapter 7 will explore the details of evidence combination between different class descriptions for a given class.

Both methods compared here aim to estimate the posterior probability of each class given the set of rules that were satisfied. The methods will first estimate the posterior odds of each class, and then convert those odds to posterior probabilities. The LS weights associated with each rule are more conveniently combined using odds rather than probabilities so we begin by presenting an odds-form of evidence combination. The odds of a proposition with probability p are $p/(1-p)$ and are monotonically related to the probability of that proposition. The goal is to determine the class with the greatest posterior odds. In order to do this, the odds form of Bayes rule is used (Duda et al., 1979):

$$O(H|E) = \frac{O(H)O(E|H)}{O(E)} \tag{4.4}$$

In this equation, $O(H|E)$ denotes the posterior odds of the class H given the evidence E (the evidence being the set of rules that were satisfied), $O(H)$ denotes the prior odds of the class and $O(E|H)$ denotes the likelihood of the evidence given the hypothesis. The prior odds of a class is simply determined by $p_0/(1-p_0)$ where p_0 is the prior probability of the class, estimated from the fraction of training examples that belong to the class. The $O(E)$ factor is the same for all classes and because we are only interested in the relative values of $O(H|E)$ this factor can be ignored.

Two basic methods will be compared for computing the remaining term - $O(E|H)$. The first is the "maximizing" method. In this method, if more than one rule from a class description is satisfied, only the most reliable one is considered. The motivation for this is that for some of the data sets, the *target* descriptions are compact DNF with respect to the given set of attributes.[2] For such data sets, the learning algorithm should produce rules that cover disjoint subspaces in the example space. Or, if the subspaces are not disjoint, at the least the probability of drawing an example from their intersection should be negligible. For such data sets, a negligible proportion of the examples in the example space lie in the intersection of the subspaces covered by rules in the target description. However, rules learned from a small training set may not correspond well to the rules in the target description and so may have rule overlap. For such data sets, if more than one rule is satisfied for a test example, it is hypothesized that the best solution (leading to the most accurate classifications) is just to use the most reliable rule and ignore any other less reliable rules. The alternative, to further increase the posterior odds of the class when a rule-overlap occurs will lead to over-estimation of the posterior odds. In the "use only the most reliable rule" solution, if $ls_{i,1}, ..., ls_{i,n_i}$ denotes the

[2] This is in contrast to the real-world data sets where for the given set of attributes, there may not exist a target description for some of the classes. That is, there may not exist a description that is consistent with all examples of that class.

LS's of the rules of class H_i satisfied by the test example, where $ls_{i,1} \geq ... \geq ls_{i,n_i}$, then $ls_{i,1}$ is used for $O(E|H_i)$ and the remaining LS's are ignored. So all the terms on the RHS of Equation 4.4 can be computed so the test example can be assigned to the class with the greatest posterior odds and hence, greatest posterior probability.

In the second method for computing $O(E|H)$, we assume that we can decompose the term $O(E|H_i)$ for class H_i into the following product:

$$O(E|H_i) = O(E_1, \ldots, E_n|H_i) = \prod_j O(E_j|H_i) \tag{4.5}$$

where we assume that pieces of evidence $E_1, ..., E_n$ are conditionally independent given the hypothesis H_i. This assumption $(O(E_i, E_j|H_k) = O(E_i|H_k)O(E_j|H_k))$ is clearly not met for compact DNF targets which assume that the probability of drawing an example from the subspace where two rules are both satisfied is close to zero: $0 = O(E_i, E_j|H_k) << O(E_i|H_k)O(E_j|H_k)$. However, for other data sets, this evidence combination method may be superior to the maximization policy. Letting $ls_{i,1}, ...ls_{i,n_i}$ denote the LS's of the satisfied rules of class H_i, we set assign the value of $ls_{i,j}$ to $O(E_j|H_i)$. So, now it is possible to compute $O(E|H_i)$ as the product of the LS's of all the satisfied rules for class H_i.

Table 4.1 compares accuracies of the maximizing and product evidence combination methods. Accuracies are averaged over 30 trials except in the case of DNA for which leave-one-out testing was used so the number of trials was equal to the number of examples (106). Information gain was used, rather than ls-content, to demonstrate the differences between maximizing and product for a well known gain metric. Similar results should hold for the ls-content metric. In the table, a trailing "i" after a data set identifier indicates number of irrelevant attributes, "e" indicates number of training examples, "a" indicates level of attribute noise and "c" indicates level of class noise. $x\%$ class noise means that the class assignments of $x\%$ of the examples were randomly reassigned - for a two class problem, this means $\frac{x}{2}\%$ of the examples will bear incorrect class labels.

The table shows that the maximizing and product strategies yield surprisingly similar accuracies. Overall, the maximizing strategy is a little more accurate (average of 84.4 versus product's average of 84.0). The only interesting differences arise on the two LED data sets with irrelevant attributes (maximizing is significantly better) and on the LED data sets with attribute noise (the product bias is "substantially" but not statistically significantly better).

The table shows that for the two noise-free LED data sets (with 8 and 17 irrelevant attributes respectively) maximizing gives a significantly higher accuracy than taking the product. In the LED domain each target class description consists of just one rule so the only time the product and maximization strategies should differ is when more than one rule has been learned and that in turn is an

Table 4.1: A comparison of the maximizing and product solutions to the rule-overlap problem. Rules were learned using information gain. The first strategy (maximizing) only uses the LS of the most reliable rule. The product approach, combines evidence from all rules of a class that were satisfied, multiplying together their LS values to yield a value for that class. Priors were estimated from class frequencies in the entire training data. A "+" indicates a significantly higher accuracy as given by the paired, 2-tailed t-test at 95% confidence.

Data set	Accuracy		% of test egs on which more than 1 rule satisfied	Base accuracy	# of training examples
	Maximizing	Product			
LED8i	87.2	85.2	3.5%	10.0%	30
LED17i	83.7+	80.6	5.5%	10.0%	30
Tic-Tac-Toe	99.0	98.9	35.1%	65.3%	670
KRKP	94.6	94.5	44.0%	52.0%	200
KRK100e	95.1	95.3	30.6%	66.7%	100
KRK200e	98.3	98.3	35.8%	66.7%	200
KRK160e,5a	91.9	92.2	45.0%	66.7%	160
KRK320e,5a	94.8	95.0	61.4%	66.7%	320
KRK160e,20c	89.6	89.6	52.7%	66.7%	160
KRK320e,20c	92.5	92.5	61.3%	66.7%	320
LED20a	88.0	89.3	16.3%	10.0%	50
LED40a	77.3	80.3	21.7%	10.0%	50
DNA	67.9	67.9	39.6%	50.0%	105
Splice	85.3	85.1	40.3%	53.4%	200
Mushroom	97.5	97.5	20.6%	50.0%	100
Hypothyroid	95.2	95.5	64.4%	90.0%	200
Iris	93.3	92.9	19.6%	33.3%	50
Wine	92.9	93.3	51.4%	39.8%	118
B.Canc.Wisc.	93.6	93.9	75.1%	65.5%	200
Voting	93.5	93.7	46.5%	62.0%	100
Soybean	87.4	88.3	18.8%	14.6%	288
Horse-colic	85.1	85.0	52.4%	63.4%	245
Hepatitis	78.7	79.6	50.2%	79.6%	103
Lymph.	77.9	78.1	44.8%	54.7%	110
Audio	71.8	72.0	14.3%	25.3%	150
Diabetes	71.9	71.9	46.5%	65.1%	200
Prim. tumor	38.8	38.7	31.8%	24.7%	225
Heart	54.2	53.8	37.3%	54.1%	200
B.Cancer	69.9	68.9	47.6%	70.2%	190
AVERAGE	84.4	84.0			

indicator that the first learned rule was incorrect. Taking the product of LS's in such a situation over-estimates the posterior odds for that class and therefore, is an unsuitable bias in this domain. The table also shows (as expected) that these LED data sets are the same data sets for which rule-overlap (column 4) occurs on the smallest fraction of the test examples.

Now consider the data sets on which the product bias beats the maximizing bias by the greatest amount (LED with attribute noise). Consider what happens to a LED that has attribute noise added independently to each of its 7 segments. Let $p(i|C)$ denote the probability of segment i being "on" for *training* examples of class C. Let $p(i,j|C)$ denote the probability that segments i and j are both "on." Then, because of the nature of the LED problem, we have for any given pair of i and j, either $p(i,j|C) \approx 0$ or $p(i,j|C) \approx 1$. Clearly, the probabilities $p(i|C)$ and $p(j|C)$ are not independent (given C). But as more and more attribute noise is added, these probabilities become more and more independent (given C), and at 100% attribute noise, we have complete independence. Therefore, as attribute noise increases, the independence assumption of the product bias is met more often and accordingly, it performs better than the maximizing bias which becomes brittle and unreliable.

4.7 Effect of type of reliability weight

In this section, two alternatives solutions for the issue of "what to do if rules of more than one class are satisfied" are compared. Specifically, we will explore the effect of changing the type of reliability weight associated with each rule. Two types of weights will be compared. The degree of logical sufficiency of the rule:

$$LS = \frac{p(rule_{ij}(\tau) = true \mid \tau \in Class_i)}{p(rule_{ij}(\tau) = true \mid \tau \notin Class_i)}$$

and the accuracy of the rule:

$$p(Class_i | rule_{ij}(\tau) = true)$$

Laplace estimates are used for all these probabilities. Table 4.2 shows how classification accuracy varies as a function of these kinds of reliability weights. It also compares two forms of priors: uniform priors (the prior probability and odds of each class are equal) and data priors (the prior probability of each class is estimated from its frequency in the entire training data). The first point to note is that using rule LS with data priors (column 3) is mathematically equivalent to using rule accuracies (column 1).[3] However, odds and probabilities are monotonically

[3]To see this consider the top 2 classes, let their labels be Class1 and Class2. Denote their most reliable rules as Rule1 and Rule2. Therefore, the Laplace accuracy method (assuming

Table 4.2: A comparison of the effect of type of reliability weight on classification accuracy. All three variants use the maximizing evidence combination within a class description. The class descriptions were learned using the information-gain metric. A "+" indicates significantly superior accuracy as deemed by a paired, 2-tailed t-test at 95% confidence. The comparison was done between the second and third columns. The approach used in the third column is mathematically equivalent to that used in the first column - the small differences are due to randomized tie-breaking.

Data set	Accuracy		
	Using Laplace Accuracy weights	Using LS weights and uniform priors	Using LS weights and data-driven priors
LED8i	87.1	87.2	87.2
LED17i	83.6	83.6	83.7
Tic-Tac-Toe	99.0	99.2	99.0
KRKP	94.6	94.6	94.6
KRK100e	95.1	95.6	95.1
KRK200e	98.3	98.5	98.3
KRK160e,5a	91.9	92.9	91.9
KRK320e,5a	94.8	95.7	94.8
KRK160e,20c	89.6	89.4	89.6
KRK320e,20c	92.5	92.9	92.5
LED20a	87.0	87.3	88.0
LED40a	79.0	79.0	77.3
DNA	67.9	67.9	67.9
Splice	85.3	84.4	85.3
Mushroom	97.5	97.5	97.5
Hypothyroid	95.2	95.2	95.2
Iris	93.2	93.4	93.3
Wine	92.9	93.0	92.9
B.Canc.Wisc.	93.6	93.9	93.6
Voting	93.5	93.7	93.5
Soybean	87.5	89.3+	87.4
Horse-colic	85.1	84.5	85.1
Hepatitis	78.7+	73.4	78.7
Lymph	78.1	77.2	77.9
Audio	71.9	72.4	71.8
Diabetes	71.9+	71.1	71.9
Prim. tumor	38.8+	34.1	38.8
Heart	54.2+	50.2	54.2
B.Cancer	69.9	68.4	69.9
AVERAGE	84.4	84.0	84.4

related so

$$(p(Class1|Rule1) > p(Class2|Rule2)) \Leftrightarrow (Odds(Class1|Rule1) > Odds(Class2|Rule2))$$

That is, the LS method with data priors should produce exactly the same classifications as the Laplace accuracies method (assuming no ties). The only difference between these columns is on test examples for which two or more classes end up with the same posterior value. For such test examples, a random tie-breaker is invoked and because this is a non-deterministic process it explains the small variations between columns 1 and 3.

The second point to note from the table is that because the accuracy of a rule measures the purity of the rule, two rules (from different-sized classes) that cover 0 examples of other classes will both end up with the same Laplace accuracy value. Consider a two class data set with p_1 training examples of $Class1$ and p_2 training examples of $Class2$. Assume that there is a pair of rules, one for $Class1$, the other for $Class2$ and that they both cover n training examples of their respective classes and cover no training examples of other classes. In such a situation, both rules will have identical Laplace accuracy values. However, their LS values will be different. LS is a measure of the proportion of the training examples of $Class_i$ for which the rule was true divided by the proportion of the training examples of other rules for which the rule was true. The LS value for the rule for $Class1$ will be $\frac{(n+1)/(p_1+1)}{K}$ and the LS value for the rule for $Class2$ will be $\frac{(n+1)/(p_2+1)}{K}$, $K = \frac{1}{n+2}$. Assume without loss of generality that $Class1$ is much more frequent than $Class2$. Then, the LS value for $Class1$ will be much less than for $Class2$ since LS measures the fraction of the class covered by the rule divided by the fraction of the other classes covered by the rule. Therefore, in situations where both these rules are satisfied for a test example, the Laplace accuracy approach will equally favor both classes. The LS approach will favor the less frequent class, arguing that the satisfied rule of the less frequent class covers a greater proportion of that class. The LS approach is reasonable if the frequencies of the classes do not vary a lot. But if they do vary, then the LS of a rule covering 1 example out of 2 training examples of that class will be greater than the LS of a rule covering 50 training examples out of 99 training examples of that class. However, using data priors rather than uniform priors will correct these anomalies.

The data set on which the differences between the Laplace accuracy approach and the LS (uniform priors) approach is the greatest is the Hepatitis data set. In the Hepatitis data set, $Class1$ consists of approximately 20% of the data and

maximizing) will assign a test example to Class1 if $p(Class1|Rule1) > p(Class2|Rule2)$ and the LS method with data priors will assign a test example to Class1 if $Odds(Class1|Rule1) > Odds(Class2|Rule2)$. Let $Odds(Class1|Rule1) = \frac{p_1}{1-p_1}$ and $Odds(Class2|Rule2) = \frac{p_2}{1-p_2}$. Then multiplying $\frac{p_1}{1-p_1} > \frac{p_2}{1-p_2}$ by $(1-p_1)(1-p_2)$ yields $p_1(1-p_1p_2) > p_2(1-p_1p_2)$. Expanding and canceling out p_1p_2 yields the desired inequality: $p_1 > p_2$. Therefore the odds and Laplace accuracy methods will rank classes in the same order.

$Class2$ consists of approximately 80%. Conflicts occur on 12% of the test examples of $Class1$ and on 10% of test examples of $Class2$. When Laplace accuracies are used, almost all the conflicts (99%) are resolved in favor of the majority class whereas when LS weights are used, almost all the conflicts (99%) are resolved in favor of the minority class. This results in the overall classification accuracy of the Laplace accuracy approach being better by 5.3%. The argument in favor of Laplace accuracy can also be seen as an argument in favor of using data-estimated priors rather than uniform priors. Another situation in which the LS approach does poorly is when most of the classes are rare classes. On the Primary-tumor data set, the rarest 50% of classes constitute just 11.3% of the examples. In this data set, using non-uniform priors yields a 4.7% increase in accuracy.

4.8 The partial clauses approach

The basic form of knowledge representation used in HYDRA is the rule - augmented with a reliability weight. In this section, a slightly different type of class description, in which the clauses are augmented by *partial clauses* is considered. The motivation for partial clauses comes from the DNA domain. On the DNA promoters domain, 13% of the test examples of class `promoter` failed to satisfy any rule of any learned class description. The same occurred for 21% of test examples of class `non-promoter`. In such cases HYDRA is forced to guess the most frequent concept. Guessing the most frequent class is a particularly bad strategy in the DNA domain since both classes occur equally frequently so guessing the most frequent class is equivalent to making a random choice. A better alternative is to determine if a rule *nearly* covers an example. The partial clauses approach is an implementation of this approach.

Partial clauses are created in a post-processing step by taking each learned rule (clause), $C \leftarrow A_1 ... A_n; LS_n$, and creating rules corresponding to the prefixes of the original rule:

$$\begin{array}{lll} C \leftarrow & A_1 & LS_1 \\ C \leftarrow & A_1, A_2 & LS_2 \\ & \ldots & \\ C \leftarrow & A_1, ... A_{n-1} & LS_{n-1} \end{array}$$

These rules have greater coverage but usually have lower purity (and hence, lower LS values) than the original rule. Since the last condition learned for a rule is usually the least reliable (in the sense that it is more likely not to be part of the target description), it may happen that the rule is correct except for the last condition. Therefore, the test example may satisfy all but the last condition in the rule. In such a case, a test example is deemed to have "nearly" satisfied the rule. The partial clauses approach offers a way of implementing the notion of

Table 4.3: Effect of using partial clauses. Weights attached to rules were Laplace estimates of the accuracies of the rules and the maximizing evidence combination strategy is used. The rules were learned using the information gain metric. A "+" indicates statistically significant accuracy according to the paired, 2-tailed t-test at 95% confidence.

Data set	Accuracy without partial clauses	Accuracy with partial clauses	Omissions rate w/o partial clauses	Omissions rate with partial clauses
LED8i	87.1	93.2+	15.9%	3.8%
LED17i	83.6	90.9+	19.8%	3.0%
Tic-Tac-Toe	99.0	98.7	2.7%	0.8%
KRKP	94.6	95.0	5.4%	0.3%
KRK100e	95.1+	89.4	7.6%	4.7%
KRK200e	98.3+	93.0	2.7%	1.0%
KRK160e,5a	91.9+	89.0	10.2%	3.5%
KRK320e,5a	94.8+	92.2	5.7%	0.5%
KRK160e,20c	89.6	90.5	13.6%	3.0%
KRK320e,20c	92.5	93.7+	8.4%	1.3%
LED20a	87.0	89.7	4.7%	0.7%
LED40a	79.0	80.7	9.3%	1.3% %
DNA	67.9	73.6	27.4%	3.8%
Splice	85.3	89.1+	20.0%	3.3%
Mushroom	97.5+	93.4	3.2%	1.5%
Hypothyroid	95.2	97.6+	6.6%	1.4%
Iris	93.2	93.4	7.2%	5.0%
Wine	92.9	93.1	5.4%	2.9%
B.Canc.Wisc.	93.6	93.4	3.3%	1.4%
Voting	93.5	94.7+	6.8%	1.8%
Soybean	87.5	89.3+	?%	?%
Horse-colic	85.1	85.6	17.4%	3.5%
Hepatitis	78.7	80.0+	25.4%	13.8%
Lymph.	78.1	78.3	20.3%	2.6%
Audio	71.9	77.0+	25.7%	6.1%
Diabetes	71.9	73.4+	30.1%	6.6%
Prim. tumor	38.8	42.5+	55.8%	26.0%
Heart	54.2	57.0+	49.4%	32.2%
B.Cancer	69.9	72.0+	34.2%	18.7%
AVERAGE	84.4	85.5		

"nearly satisfied" that leads to improvements in classification accuracy as shown in Table 4.3.

Table 4.3 shows that partial clauses are especially useful on the real-world data sets (mushroom to cancer). On these data sets, partial clauses help increase accuracy by a statistically significantly on 60% of the data sets and lead to minor improvements on most of the other data sets. Partial clauses are also a help on artificial data sets (e.g. noise-free LED data sets) that have high omission rates. The omission rate for a class is the fraction of test examples of that class that failed to satisfy any rules of that class. The omission rate for a data set is the average of the omission rates for classes in that data set, weighted by the class probabilities (when an example of class C does not match any of the learned rules for class C, an *error of omission* is said to occur). To try to understand why partial clauses help, consider noisy data sets. For such data sets, HYDRA may overfit the noise and learn many rules, each of which covers a few examples (low coverage) but each of which has high purity. The probability that the last condition in the rule is incorrect is increased for such noisy data sets. Furthermore, the class may not even be describable using completely pure rules so the partial rule may be the best rule to use.

The table also shows that the partial clauses approach is a bad idea on the King-Rook-King data set with low or zero noise and on the mushroom data set. To understand this, consider what happens on data sets with sufficient training set sizes that do not have noise but do have small disjuncts. For such a data set, if a conflict occurs then the rule with greater purity should be favored even if it covers many fewer examples than a competing (impure) rule that covers many examples. Laplace accuracy may give a greater weight to the slightly less pure rule with much greater coverage. These are the circumstances which occur for the four King-Rook-King variants in which the partial clauses approach does significantly worse. The mushroom data set is probably also quite noise-free as can be deduced from the very high accuracies that are obtained on that data set.

It is also interesting that for the heart and breast-cancer data sets (on which many algorithms have difficulty achieving accuracy rates much better than base accuracy) even with partial clauses, the omission rates remain high. That indicates that there are not enough training examples in these data sets to form rules with large coverage with the given set of attributes. It may also be the case that the given attributes are all irrelevant so even at large training set sizes it may not be possible to learn rules that have large coverage.

In summary, partial clauses are one way of increasing accuracy on noisy, real-world data sets but may lead to lower accuracy on sufficient-sample noise-free data sets.

4.9 Effect of the gain metric

In this section, the choice of the gain metric on accuracy is examined. The gain metric is used during learning to decide which node to add to a decision tree or to decide which literal to add to a rule. Three gain metrics are compared: information gain, the Bayes gain metric (Buntine, 1990) which learns models by maximizing the posterior probability of the model and the ls-content gain metric (Ali & Pazzani, 1992). Of particular interest are the hypotheses that the Bayes gain metric and the ls-content gain metric lead to the learning of fewer, more general rules than the information gain metric when used on noisy, real-world data sets and hence, lead to more accurate classifications.

The information gain metric was defined in Equation 3.1 in Chapter 3. The ls-content metric is defined in Equation 4.3. The Bayes gain metric for decision trees was presented in Equation 3.8 in Subsection 3.3.2. Here we describe how to adapt it for learning rules.

As HYDRA learns a rule by successively specializing the rule (i.e. by greedily adding conditions to the body of the rule), the condition $\mathcal{L}$ which maximizes the gain in posterior probability is added:

$$gain(\mathcal{L}) = p(h + \mathcal{L}|\vec{x}, \vec{c}) - p(h|\vec{x}, \vec{c}) \tag{4.6}$$

Here $h + \mathcal{L}$ denotes the model h with the addition of the condition, $\vec{x}$ denotes the training data and $\vec{c}$ denotes the class labels of that training data.

This general principle (of maximizing gain in posterior probability) is instantiated for rule-learning as follows. During learning for class i, there are only two "virtual" classes: training examples of class i are called "positive training examples" and examples of other classes are called "negative training examples." The addition of the condition $\mathcal{L}$ to the rule separates out these two virtual classes into two subsets: T: the examples that match the new rule and, F: those that do not. By observing that this situation is analogous to adding a binary test to a decision tree, we obtain the following for the contribution to the posterior probability by $\mathcal{L}$:

$$gain(\mathcal{L}) = p(\mathcal{L}) \times \frac{B(p + \alpha_1, n + \alpha_2)}{B(\alpha_1, \alpha_2)} \times \frac{B(P - p + \alpha_1, N - n + \alpha_2)}{B(\alpha_1, \alpha_2)} \tag{4.7}$$

where the rule covers p out of P positive training examples and n out of N negative training examples. Once a condition has been added (and if the rule still covers some negative examples) P is set to p, N is set to n, and learning of the next literal of the rule is started.

Note however that Equation 4.7 is symmetric: that is, it will assign the same credit to a condition that includes p positive and n negative examples as to a condition that includes $P - p$ positive and $N - n$ negative examples. This is a problem because we want to prefer the condition that minimizes the number of negative examples covered and maximizes the number of negative examples excluded. This symmetry poses no problem for decision trees for which further learning then takes place for all the resulting subtrees. But when learning rules, we are only interested in modeling the T subset - the subset of examples that are *included* by the rule. Therefore, we modify the definition of gain as used in rule set models, $gain\prime$, so that $gain\prime$ is 0 if the ratio of positive examples to negative examples decreases as a result of addition of $\mathcal{L}$:

$$gain\prime(\mathcal{L}) = \begin{cases} gain(\mathcal{L}) & if \frac{p}{p+n} > \frac{P}{P+N} \\ 0 & \text{otherwise} \end{cases}$$

Table 4.4 compares the effect that the gain metric has on classification accuracy. All the results are for a maximizing evidence combination scheme using

Table 4.4: A study of effect of gain metric on classification accuracy. Paired, 2-tailed t-tests at 95% confidence were used to compare the algorithms. A t-test was done between each pair of gain metrics. A sole "+" on a line indicates that that gain metric was significantly superior to the other two metrics. A sole "–"indicates that the metric was significantly inferior. The only exception to this pattern is the Tic-Tac-Toe data set in which all three metrics were statistically distinguished from each other: information gain did best, followed by Bayes and then ls-content.

Data set	Accuracy using info. gain	Accuracy using the Bayes gain metric	Accuracy using ls-content	# rules using info. gain	# rules using Bayes gain	# rules using ls-content
LED8i	87.1	90.8+	87.9	12.1	11.3	11.7
LED17i	83.6	87.6+	84.2	12.5	11.7	12.1
Tic-Tac-Toe	99.0+	98.4	79.8–	24.7	19.7	11.1
KRKP	94.6	94.7	95.1	17.1	13.5	16.2
KRK100	95.1	95.3	94.7	9.1	8.6	10.2
KRK200	98.3	98.3	98.0	12.3	11.1	14.3
KRK160e,5a	91.9	93.5	92.2	32.2	14.3	13.5
KRK320e,5a	94.8	96.6+	95.5	24.1	14.1	16.1
KRK160e,20c	89.6	92.1+	90.1	44.2	17.3	21.0
KRK320e,20c	92.5	95.5+	91.5	60.9	16.6	14.0
led7,20	87.0	91.3	87.3	17.5	14.1	13.5
led7,40	79.0	79.7	80.0	20.1	14.6	14.6
DNA	67.9	65.1	77.4+	9.8	8.8	11.8
Splice	85.3	85.9	84.4–	15.3	13.9	17.9
Mushroom	97.5	97.6	98.5	6.6	5.5	8.3
Hypothyroid	95.2	95.2	97.7+	2.5	2.2	5.7
Iris	93.2	92.9	93.7	5.0	4.5	5.4
Wine	92.9	93.4	96.3+	7.5	7.8	11.2
B.Canc.Wisc	93.6	94.0	93.4	9.8	7.9	12.0
Voting	93.5	93.3	94.0	5.7	5.6	7.1
Soybean	87.5–	88.5	88.4	32.8	27.7	37.5
Horse-colic	85.1	84.6	84.6	21.0	20.1	25.4
Hepatitis	78.7	77.5	81.7+	7.6	7.1	9.9
Lymph.	78.1	77.5	79.4	15.1	13.4	15.8
Audio	71.9	71.8	68.9	42.9	38.7	47.8
Diabetes	71.9	72.9+	71.5	28.7	24.3	27.7
Prim. tumor	38.8–	40.9	41.0	129.9	25.7	81.8
Heart	54.2	54.7	54.9	46.5	38.5	47.8
B.Cancer	69.9	70.4	71.4	39.6	14.5	9.7
Average	84.4	85.2	84.6	24.6	14.9	19.0

Laplace accuracy weights. Overall, the Bayes gain metric does best, and there is little overall difference between ls-content and information-gain. The only data set in which information-gain did significantly better than the others was the Tic-Tac-Toe data set. This is presumably because the information gain metric puts a relatively larger emphasis on learning pure rules rather than rules with greater coverage. This is a good bias in the Tic-Tac-Toe domain because there is no noise in that data set and a large number of training examples is used. The Tic-Tac-Toe domain also illustrates one disadvantage of using the ls-content metric. The ls-content metric learns rules that value coverage over purity and consequently in this data set, learns rules that cover many negative training examples. In this respect, the Bayes metric seems to be a better way (than ls-content) to trade-off purity and coverage.

On the noise-free, artificial data sets, the Bayes metric was the overall winner, only losing significantly on the Tic-Tac-Toe data set to information-gain. It was always as good or significantly better than ls-content. The number of rules learned by all three metrics were roughly comparable for this group of data sets.

On the six noisy artificial data sets, the Bayes metric was significantly better than the other two metrics on three data sets and did equally well on the other three. Ls-content and information-gain performed similarly on these data sets. Looking at the numbers of rules learned on these data sets, one can see that information-gain overfits the noise, learning many more rules, on average, than the other metrics. On the King-Rook-King data set with 320 examples and 20% class noise, for example, information gain learned an average of 25 rules for class `illegal`. It is known that the correct class description for this class requires only 8 rules. The Bayes gain metric, on the other hand, learned only 8.2 rules on average. Ls-content learned, on average, 6.4 overly-general rules for that class.

For the molecular-biology data sets, the only significant result was that the ls-content metric did poorly on the Splice data set - there were no other significant differences in accuracies. Ls-content also learned more rules than the other gain metrics on the Splice data set.

For the natural data sets (fourth group of rows in Table 4.4) the ls-content metric had its best showing, attaining significantly higher accuracies than the two other metrics on three data sets. Interestingly, it learned more rules on all three of these occasions than the other metrics. The information gain metric never achieved significantly higher accuracies on any of these natural (most of them noisy) data sets. The Bayes metric obtained significantly higher accuracies than the information-gain metric on three of these data sets (Soybean, Diabetes and Primary-tumor) but was significantly *less* accurate than ls-content on three data sets and significantly more accurate only once.

This fourth group of data sets is arranged so that those for which lower accuracies are attained appear at the bottom of the group. For these very noisy real-world data sets (Primary-tumor, Heart, Breast Cancer) information gain learns many more rules than the other two metrics and achieves a significantly lower accuracy on the Primary-tumor data set. The median number of rules learned for these more noisy (from Soybean downwards) real-world data sets were: information-gain: 32.8, Bayes: 24.3, ls-content: 27.7. On the less noisy real-world data sets, the median number of learned rules were: information-gain: 6.2, Bayes: 5.5, ls-content: 7.7. For these natural data sets ls-content was the best.

Overall, the Bayes metric did best. It was almost always better than the information-gain metric and only worse than ls-content on a few occasions for the noisy, natural data sets.

4.10 Trading off purity for coverage

In this section, the following generalized form of ls-content is used:

$$ls(p, n, p_j, n_j) = ls(p, n, p_j, n_j)^{1-\alpha} \times p^{\alpha} \qquad (4.8)$$

Setting α to 0.5 yields the conventional ls-content metric.[4] A value of 1 for α yields a gain metric that maximizes coverage and is not sensitive to purity. This value is not particularly useful since it totally ignores purity. A value of 0 for α yields a metric that maximizes purity and learns many rules each with low coverage.

The maximizing evidence combination scheme was used and Laplace accuracy was used as a rule reliability measure.

Figures 4.1 through 4.4 show how accuracy varies as a function of α. For each data set, 30 trials (except as noted earlier) were run at each of the following values of α: 0.0, 0.2, 0.4, 0.6 and 0.8. α corresponds to the horizontal axis in each figure. The vertical axis corresponds to accuracy. Note that the vertical range is different for each data set. In particular, for Hypothyroid and Iris the true variance with respect to α is quite small despite the appearance in the graph. Figure 4.1 shows that except for the noise-free LED data sets, increasing α so as to give a relatively greater emphasis to coverage increases accuracy up to a point but then leads to a drastic decrease if the emphasis on purity is even further decreased.

The LED training sets only contain 30 examples so putting a greater emphasis on coverage is a good idea. This is the same reason that partial clauses were such a help on these data sets. For the Tic-Tac-Toe data set, on the other hand, there are

[4] Because the two factors (purity and coverage) that need to be maximized are combined as a product, a geometric average is used instead of an arithmetic average.

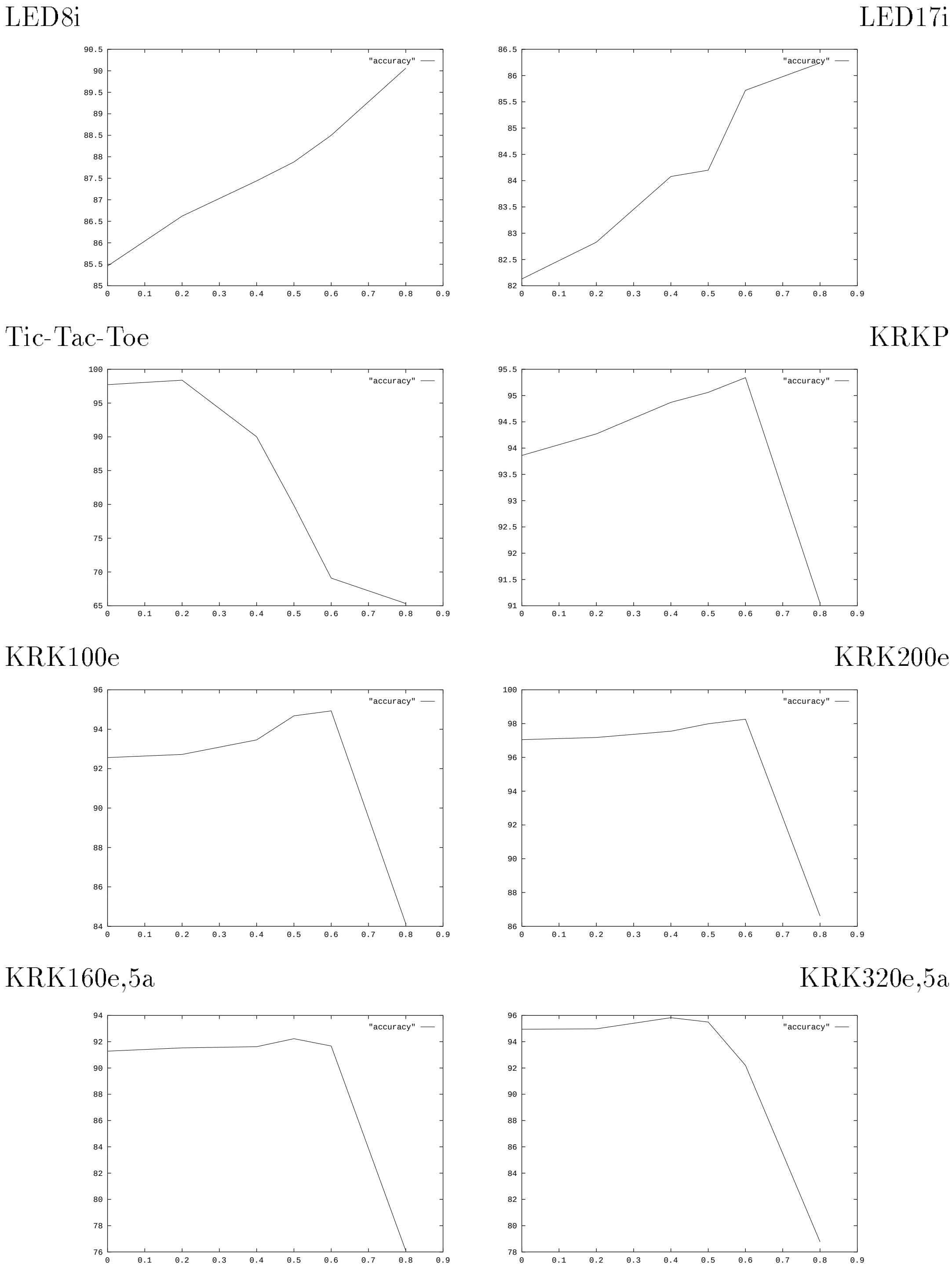


Figure 4.1. Accuracy (y-axis) as a function of parameter α (x-axis) ls-content.

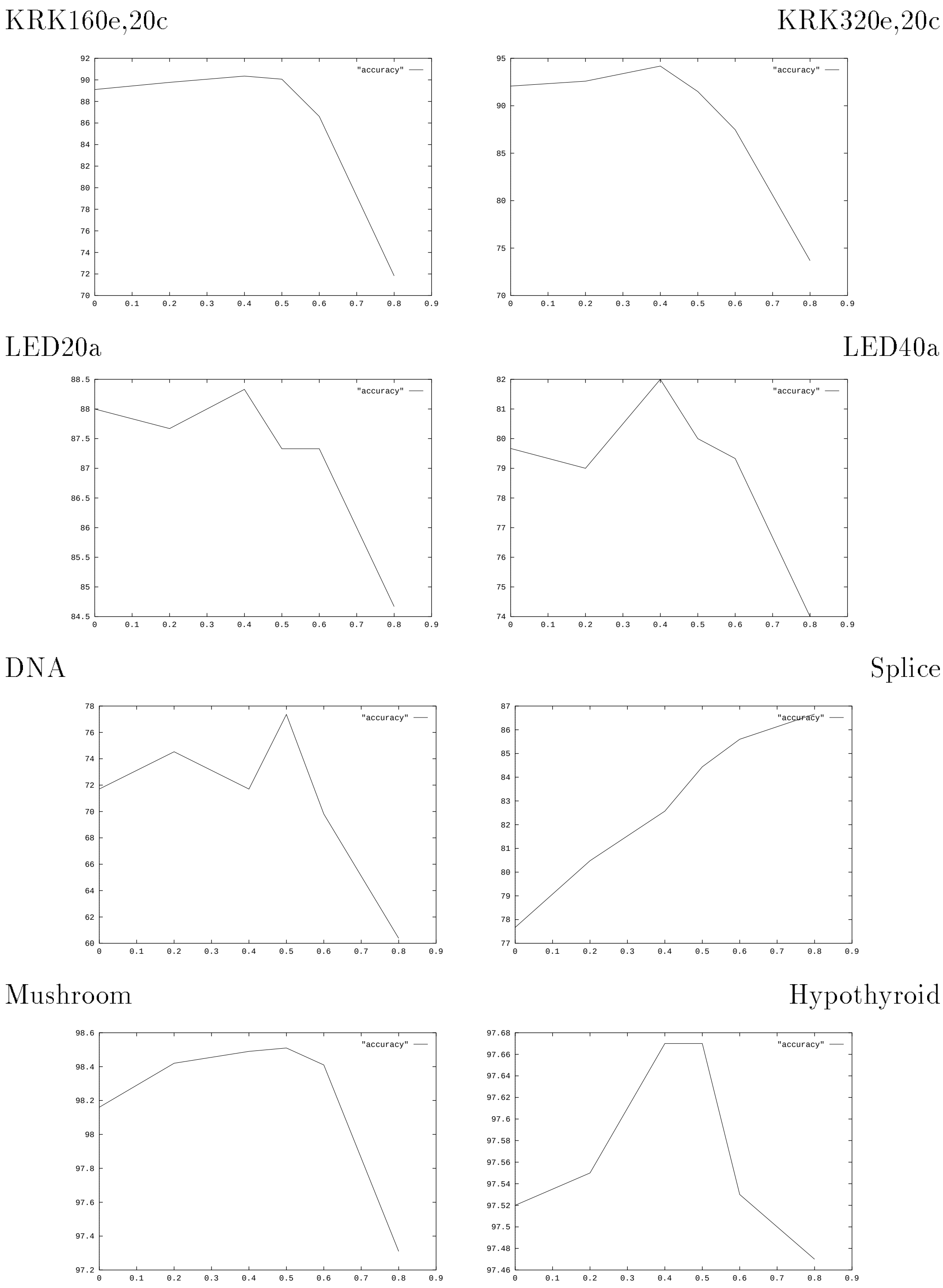


Figure 4.2. Accuracy (y-axis) as a function of parameter α (x-axis) ls-content.

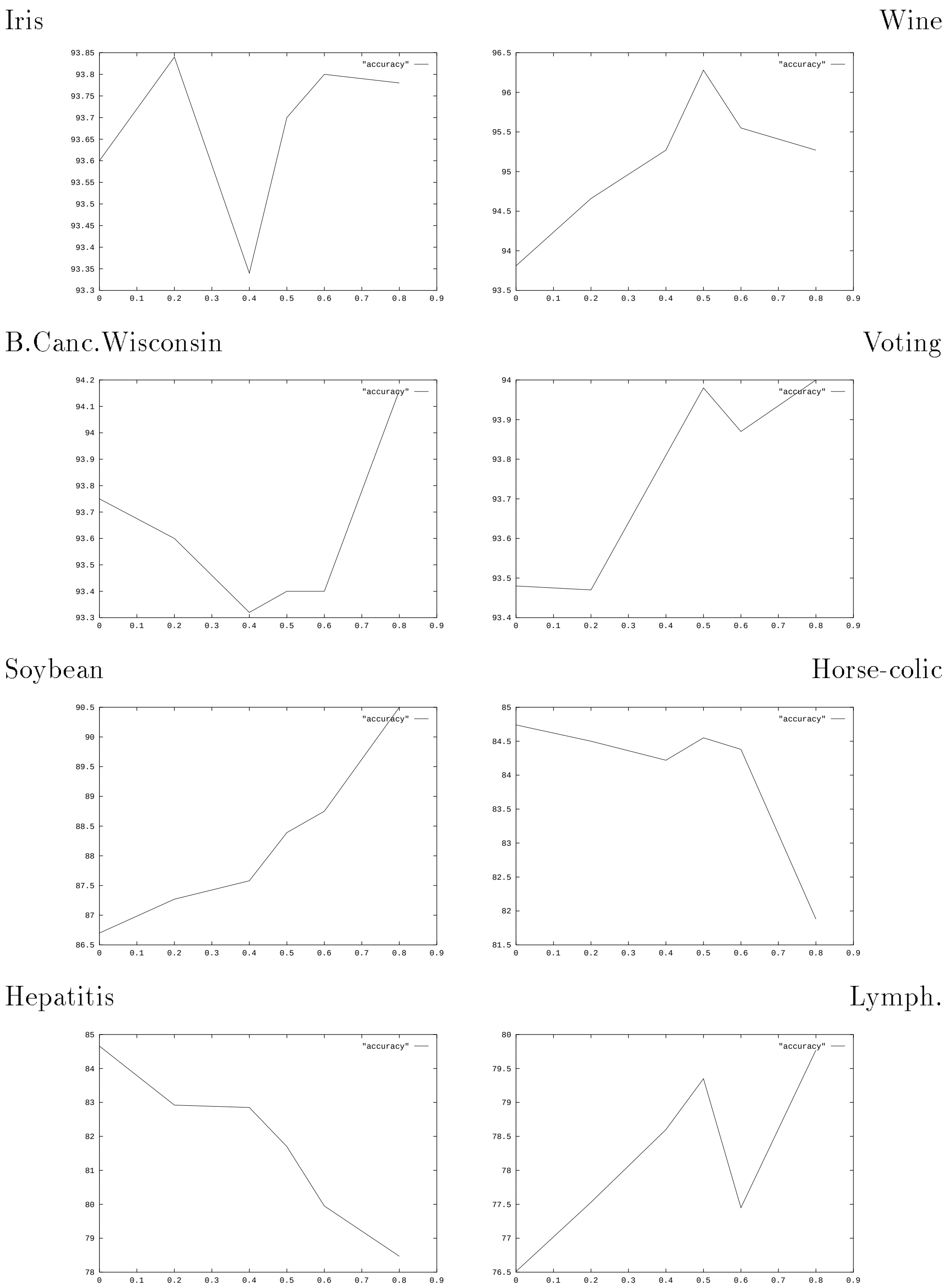
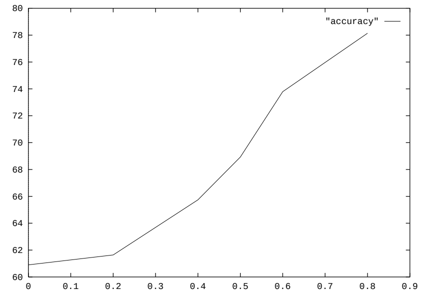
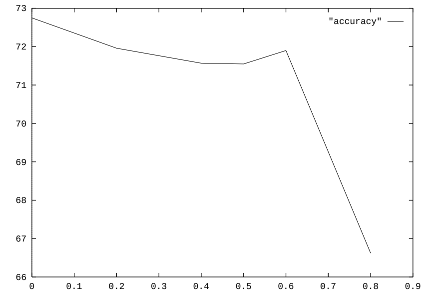


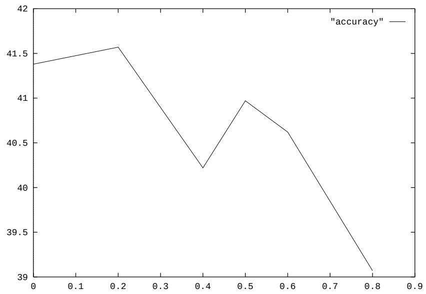
Figure 4.3. Accuracy (y-axis) as a function of parameter α (x-axis) ls-content.

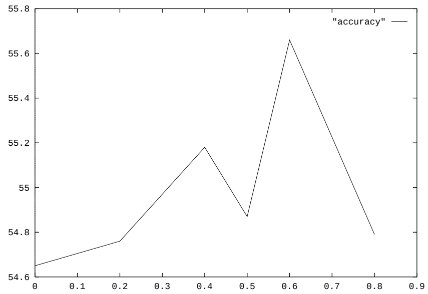
Audio. Diabetes





Primary-tumor Heart





Breast Cancer

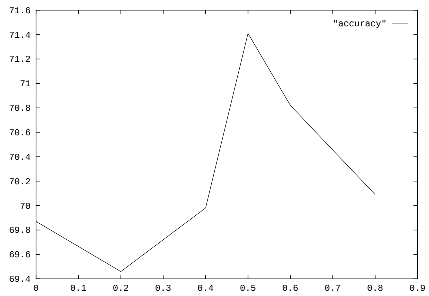


Figure 4.4. Accuracy (y-axis) as a function of parameter α (x-axis) ls-content.

a lot of examples in the training data and no noise so its a good idea to "overfit" the data: ie: it is a good idea to use an α value of 0.0. The graphs in Figure 4.1 show that the default value of α of 0.5 does reasonably well except in the Tic-Tac-Toe data set (also see Table 4.4 which shows that ls-content does relatively poorly on this data set).

On the molecular-biology data sets (Figure 4.2), Splice presents an interesting result. The similarity to the LED graphs suggests that the 200 training examples used are not enough for this data set. Three other data sets (Soybean, Lymphography and Audiology) present graphs similar to those of LED8i and LED17i suggesting that more training examples are needed in those domains.

Graphs for four data sets (Hypothyroid, Iris, B.Cancer.Wisconsin and Voting) are misleading because of the small vertical scale used. In all four cases, the extent of the vertical variation is less than 1%. So the conclusion must be that for these data sets, accuracy does not vary very much with α. Hepatitis and Primary-tumor present graphs similar to Tic-Tac-Toe suggesting that the sample size is sufficient for these data sets. The peak accuracies, however, are only 85% for Hepatitis and 41.5% for Primary-tumor suggesting that these accuracies are the best that can be obtained using the given attributes - learning more general clauses with greater values of α will lead to a reduction in the overall accuracy. (The 85% accuracy for Hepatitis is well above that achieved by any other algorithm that we have used.)

This subsection has shown how the different data sets behave as purity is traded off for coverage. The graphs support the conjecture that maximum accuracy is achieved neither when one attempts to maximize purity nor when one attempts to maximize coverage but at an intermediate point. The graphs show that the default value of $\alpha = 0.5$ does quite well. The other (not surprising) conclusion is that aiming to learn very pure rules works well for noise-free data sets with sufficient sample size (e.g. Tic-Tac-Toe). The surprise, however, is that the shape of the graph can give a clue as to whether more training examples are needed. If the graph is shaped like that of Tic-Tac-Toe, it suggests that the given number of training examples is in the asymptote of the learning curve. If the graph is shaped like that of LED8i, it suggests that the given number of examples is in the region of the learning curve where accuracy increases rapidly if a few more training examples are added.

4.11 Discussion

This chapter has presented HYDRA, the system that is used to explore the problems addressed in this thesis. The chapter also presented how varying the solutions to five key issues affects classification accuracy. These decisions are:

1. (Classification) What to do if more than one rule from a class is satisfied for a test example? How should evidence from all satisfied rules be compared? Two approaches to this problem were compared: the maximizing and product approaches and it was shown that there is very little difference between the two.

2. (Knowledge Representation) What kind of reliability weight to associate with a rule? It was shown that associating the Laplace accuracy of a rule yields better accuracies than using the degree of logical sufficiency (LS) of a rule; specially in domains in which much of the data set consists of rare classes. It was also noted that using LS with data-estimated priors is equivalent to using the Laplace accuracy. Therefore the conclusion to prefer Laplace accuracy over LS (with uniform priors) is equivalent to preferring LS with data-estimated priors over LS with uniform priors.

3. (Classification, Knowledge Representation) What to do if no rules of any class are satisfied? Two approaches were compared: i) guess the most frequent class in the training data, ii) try to find clauses (rules) that were almost satisfied. It was shown that this latter approach, called the *partial clauses* approach, yields a significant increase in accuracy, particularly on noisy, real-world data sets. It is not a good idea on large-sample, noise-free data sets.

4. (Learning) What kind of gain metric to use? The gain metrics compared were: information gain, gain in posterior probability (Bayes) and ls-content. Most of the differences were less than 4 percentage points but there were a number of data sets on which differences were statistically significant. It was shown that Bayes is best overall and that ls-content is best for noisy, real-world data sets. It was also shown that the information gain metric usually does badly because it learns many rules, each covering few examples (i.e. overfits the data) and that this leads to poor accuracy.

5. (Learning) How does trading-off purity and coverage affect accuracy? A general form of ls-content which accepts a parameter, α, that allows for a trade-off of purity and coverage was investigated. The results supported the notion that maximal accuracy is obtained neither by looking for pure rules neither by ignoring purity but by trading-off purity and coverage (a hold-back set could be used to estimate the best value of α). Surprisingly, the resulting coverage-purity trade-off graphs can suggest when more training examples are needed in a data set (low accuracy, rising alpha-variation curve as in the case of LED8i) as opposed to when the set of attributes needs to be changed (low accuracy, declining alpha-variation curve as in the case of Hepatitis and Primary Tumor).

Chapter 5
The small disjuncts problem

This chapter presents a description of the small disjuncts problem, evidence for its existence and an algorithm for reducing the errors due to small disjuncts and the overall error rate in the presence of small disjuncts. Although the small disjuncts problem has been shown to exist for algorithms other than rule-learning algorithms (decision trees - Quinlan, (1991)) because the focus of this thesis is on rule-learning algorithms, the discussion in this chapter will assume the context of a rule-learning algorithm. Many but not all of the statements also apply to the small disjuncts problem for non-rule-learning algorithms.

Next, the chapter presents the method of cumulative distributions as a measure of the *degree* of the small disjuncts problem. Then it is shown that making classifications using rules with reliability estimates in a one-per-class model reduces the degree of the small disjuncts problem. That is, HYDRA has less of a small disjuncts problem than FOIL. Finally, the chapter concludes with a comparison of the rule reliability method to other methods for reducing the small disjuncts problem.

5.1 Motivation

It is the aim of rule-learning algorithms to learn rules that are *all* error-free. That is, if the rule is of the form:

$$A_1\&...\&A_n \rightarrow C$$

(which is read as "If A_1 and ... and A_n then conclude that the example belongs to class C") then the error rate of the rule on some set of examples is defined as the proportion of occasions on which the body of the rule was satisfied by examples of classes other than C. Formally, if S_C denotes the number of times that the body of the rule was satisfied for examples of class C and $S_{\neg C}$ denotes the number of times that the body of the rule was satisfied for examples of other classes, then the error rate of the rule is defined as $e = \frac{S_{\neg C}}{S_{\neg C}+S_C}$. (The accuracy of the rule is defined as $1-e$.) It is the aim of rule-learning methods to learn rules that all have zero error rates.

However, as Holte *et al.* (1989) have observed, in practice, some learned rules have quite high error rates. Furthermore, as shown in Muggleton *et al.* (1992), the error rates of rules as measured on any set of examples distinct from the training set (called the *off-sample error rate* or *generalization error rate*) do not correlate well with the rules' error rates on the training set. This is because most rule-learning algorithms overfit the training data to varying extents. Muggleton *et al.* noted, instead, that the off-sample error rate correlates better with the training coverage of the rule; i.e. the number of training examples that it covers.[1]

This is just another way of corroborating the observation of Holte *et al*: that the rules that cover few examples tend to have higher error rates. As may be inferred from this discussion, there is no hard and fast criterion to differentiate "small" disjuncts from "large" disjuncts. Accordingly, to date, no one has formulated a rigorous definition for what constitutes a small disjunct. In practice, if some significant number of the learned rules have high error rates and those rules each cover a few training examples, then the learning algorithm is said to have "a small disjuncts" problem on that data set.

[1]Section 5.4 discusses a new way (Quinlan, 1991) of estimating off-sample error-rate from the training data that factors in rule coverage.

Why does the small disjuncts problem arise? Two major reasons are:

1. Rare subclasses.
2. The nature of "covering" or "separate and conquer" algorithms.

One reason for the small disjuncts problem is the presence of rare subclasses in a class. Collectively, the rare classes may constitute a substantial proportion of the class so just ignoring subclasses that, individually, are represented by fewer than 5 training examples, say, will not work. Such an approach would lead to too many errors of omission. If the rare subclasses do *not* constitute a significant proportion of the class, it may be profitable to use a simple policy such as "do not learn rules that cover fewer than N training examples" to combat the small disjuncts problem. Therefore, the small disjuncts problem is only a problem when the rare subclasses, collectively constitute a substantial proportion of the class (this qualification was first noted in (Danyluk & Provost, 1993)). An important distinction needs to be made here: the subclasses exist in the example space, independent of any learning algorithm. Small disjuncts, however, are a product of the learning algorithm. Conceivably, some kinds of rare subclasses may lead to a small disjuncts problem for one rule-learning algorithm but not for another so the presence of many rare subclasses *per se* does not imply a small disjuncts problem. This is especially true if the alphabet (set of attributes) is changed or if the form of the learned model is changed (from decision tree to rules, for example).

A second reason for the small disjuncts problem is that "covering" or "separate and conquer" learn rules for a class in an iterative fashion, learning a rule for some subset of the class and removing examples of that subset to learn rules for the remainder of the class. As more rules are learned, fewer training examples remain. Generally, the probability of learning a rule exactly correctly is an increasing function of the number of training examples. Consider the following scenario. A separate-and-conquer algorithm has learned many rules and examples belonging to two rare subclasses remain. There is no single attribute-value combination that holds true for examples of both subclasses as measured over the *entire* example space. But there is noise in the data so it happens that some attribute-value combination holds for *training* examples of both subclasses and holds for no training examples of other classes. Then, the learning algorithm will learn an overly general rule (consisting of that attribute-value combination) that will cause errors of commission on test examples.

Alternatively, small disjuncts may be learned even if there are no rare subclasses in the class because the rules learned earlier may have split up a subclass in a way such that now two (or more) rules need to be learned to cover that subclass. There are many reasons why an algorithm may overlook the correct rule for a large subclass:

Table 5.1: Evidence for small disjuncts problem for CN2 on the Chess endgame domain (KPa7KR). Coverage refers to number of test examples covered by the rule. Rules (disjuncts) that cover few test examples typically have much higher error rates than rules that cover many test examples.

Training Coverage	Error Rate	Cumulative % Matched	Cumulative % Errors
4	16%	3	6
5	16%	11	25
6	20%	19	45
...			
11-13	4%	45	97
14-30	2%	56	100
31-40	0%	76	100
>40	0%	100	100

1. If the sample size is small, other rules that are nearly as good in the overall example space may appear to be better on the sample. However, this sub-optimal rule may cover examples of more than one subclass. Then, it may be difficult to cover the remaining examples of those subclasses. By difficult, we mean that with respect to the given set of attributes, an error-free description of the remainder of the subclass is syntactically complex and unlikely to be correctly learned.
2. One or more of the correct attributes required to describe that large subclass may not be available to the learning algorithm.
3. Noise may lead to the selection of a rule that appears to be better (on the training set) but is more error prone off the training set (and causes the major subclass to be split in a way that is difficult to repair).

5.2 Evidence for the small disjuncts problem

Holte *et al.* (1989) were the first to characterize the small disjuncts problem. They showed (Table 5.1) that rules learned by CN2 (Clark & Niblett, 1989) that covered a relatively small proportion of the training examples (45% in the figure) were responsible for a disproportionate amount (97% in the figure) of the errors. However, as Table 5.1 shows, although rules that cover few training examples tend to have higher error rates, error rate is not a strictly monotonically decreasing function of coverage - rules that covered 6 examples had a *higher* error rate than rules that covered fewer than 6 examples. This may be due to sampling variation or due to some domain-dependent peculiarity. A more robust way of demonstrating that the small disjuncts problem exists is to plot *cumulative* coverage against *cumulative* errors (Figure 5.1). The cumulative error distribution graph for CN2 is indicated in the middle figure in Figure 5.1 by small open circles. Because the graph for CN2 is above the diagonal it indicates that rules that cumulatively cover some small percentage (e.g. 45%) of the test examples are responsible for a much larger percentage (e.g. 97%) of the classification errors. Therefore, by being above the diagonal, the graph indicates that CN2 has a small disjuncts problem on this domain. The diagonal line indicates the distribution of errors that would be obtained if there was no small disjuncts problem; i.e. if rules made errors in proportion to their coverage. This kind of cumulative graph provides a more robust way of detecting the small disjuncts problem than using error rates alone. If the graph for an algorithm is "significantly" above the diagonal line, that indicates that algorithm has a small disjuncts problem on that data set.

A similar (above the diagonal) graph is obtained for FOIL on the King-Rook-King domain (top) and on the DNA promoters data set (bottom) indicating that FOIL has a small disjuncts problem on those data sets. Since FOIL only learns a concept description for the class nominated as the "positive" class it is necessary to make a choice of which class to nominate as the positive class. To allow comparability to previous research, we nominated the `illegal` class in the King-Rook-King domain as the positive class and the `promoters` class in the DNA domain as the positive class. If a test example of class `legal` matches more than one rule of class `illegal`, that is only counted as one error of commission.

The errors plotted on the y-axis for CN2 constitute all the errors that CN2 can make. Errors of omission, however, cannot be attributed to particular rules so the errors plotted on the y-axis for FOIL correspond only to the errors of commission made by FOIL rules. CN2, however, cannot make errors of omission because it learns *decision lists* (Clark & Niblett, 1989). A decision list is an *ordered* sequence of rules concluding for different classes. CN2 cannot make errors of omission because by convention, the last rule in the CN2 sequence (called the *default rule*) has a rule-body which always succeeds (i.e. matches any test example). The class associated with the default rule is the majority class of the training examples that failed to match any of the preceding rules. Any test example failing to match the ordinary CN2 rules will always match this rule so there will be no error of omission in CN2.

The situation in HYDRA is different again. In HYDRA it is possible that a test example of class `legal` matches a rule of class `illegal` *and* a rule of class `legal`. Furthermore, it can happen that the rule of class `legal` has a higher reliability weight. In this case, the example will be correctly classified even though it was matched by a rule of the wrong class. However, because an error did not result, we do not count this as an error of commission. Therefore, an error of commission (for a test example of class C) in HYDRA occurs when a rule of class other than C matches the test example *and* beats out any competition from class C. The y-axis for HYDRA plots only the errors of commission made by rules of

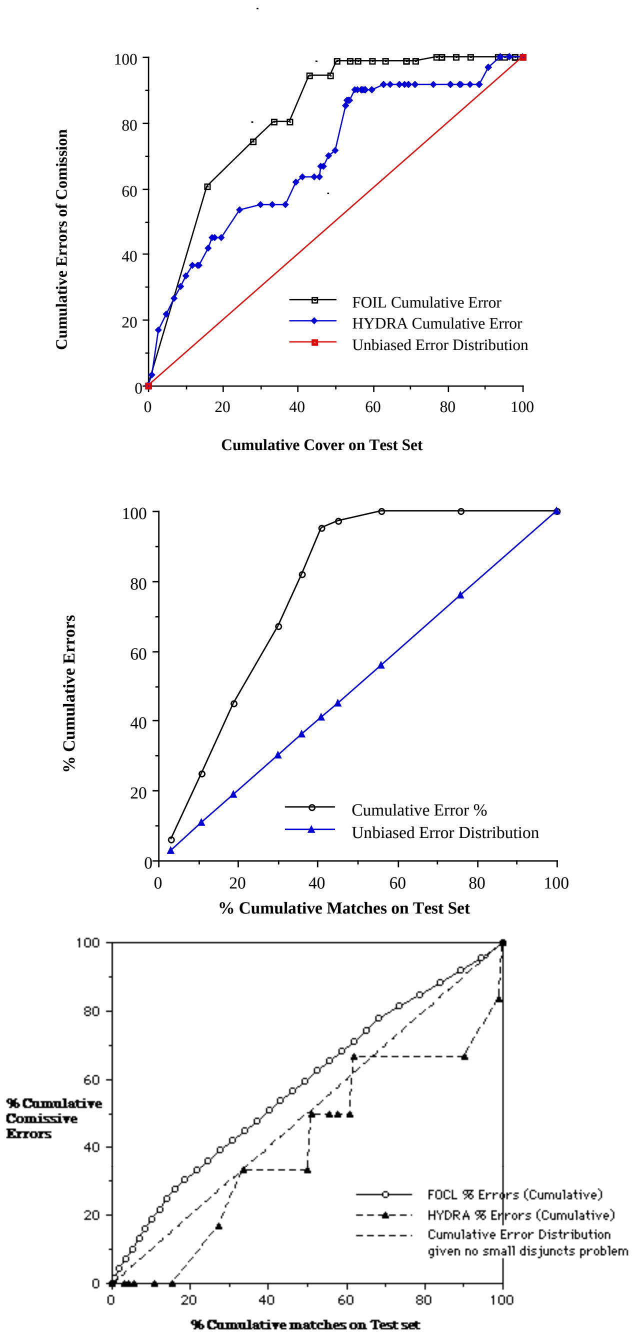


Figure 5.1: Top: FOIL and HYDRA (to a lesser extent) have the small disjuncts problem on the King-Rook-King data set with 160 training examples and 5% attribute noise. Middle: Small disjuncts problem for CN2 on KPa7KR. Bottom: FOIL and HYDRA on the DNA domain.

class `illegal`. This allows greater comparability with the FOIL graph which by definition can only learn rules for one class.

5.3 Reliability weights: a partial solution

The question that is asked in this section is:

> Does making classifications using a one-per-class model (a model in which a description is learned for each class in the domain) with reliability weights attached to the rules decrease the small disjuncts problem?

In order to answer this question, a way of measuring the degree of the small disjuncts problem is needed. A reasonable way to measure whether algorithm A has a greater small disjuncts problem than algorithm B is the following: Algorithm A is considered to have a greater small disjuncts problem than algorithm B if the cumulative error distribution graph for A has a higher value that that of B for *every* value of the x-axis (coverage). Note that this definition is conservative in that if the cumulative error distribution graphs intersect, then one cannot say which algorithm has the greater small disjuncts problem. It is important to be able to measure the degree of the small disjuncts problem for an algorithm since we want to see if we can reduce that degree.

It is known that rules covering fewer examples are more error-prone (e.g. Muggleton *et al.*, 1992) so if such rules are assigned small reliabilities, it is our hope that they will lose in a conflict situation with a less error-prone rule. The assumption is that the less error-prone rule will have greater coverage and hence, will have a greater reliability weight.[2] Of course, this is only a partial solution to the small disjuncts problem because the small disjunct can still make an error on a test example if there is no satisfied rule of the correct class that can contend for that example.

Using this definition, we adapted the procedure of Holte *et al.* (1989) for HYDRA and recorded an error distribution for rules of class `illegal` on the King-Rook-King domain similar to that shown in the illustration in Table 5.2. The actual graph obtained for HYDRA is shown in the top sub-figure of Figure 5.1. This figure provides evidence for the main conclusion of this chapter:

> Using reliability weights (LS values) and the one-per-class model allows the small disjuncts problem to be reduced when compared to that of FOIL.

[2] This line of reasoning works only if both rules cover no training examples of other classes. If they do, then their LS weights are no longer monotonically related to positive training coverage.

Table 5.2: Evidence for small disjuncts problem for HYDRA on Chess endgame domain (King-Rook-King) for class `illegal` (160 training examples, 5% added attribute noise). Coverage refers to number of test examples (of any class) covered by the rule of class `illegal`. Error rate (column 4) is derived by dividing number in column 3 by number in column 2. This is an illustration of a small disjuncts problem for HYDRA since rules that cumulatively match a small proportion of the test examples (e.g. 11%) are responsible for a much greater proportion of the errors (e.g. 31%).

Training coverage	# (of any class) matched	# of wrong class matched	Error Rate	% Matched (cumulative)	% Errors (cumulative)
4	60	12	20%	6%	14%
5	50	14	28%	11%	31%
...					
31-40	220	11	5%	70%	93%
>40	300	6	2%	100%	100%

The bottom graph in the figure shows a similar reduction on the DNA promoters domain. LS was used as a reliability measure in all these experiments and ls-content was used as a gain metric.

Figure 5.1 shows how the *distribution* in error changes as a result of using reliability estimates and the one-per-class model. It does not indicate whether the overall error rate decreased. However, the overall error rate does decrease for this data set (and many others) when using HYDRA instead of FOIL. On the King-Rook-King data set with 160 training examples and 5% attribute noise, the overall accuracy increases from 90.6% (5.0% standard deviation) for FOIL to 93.6% (3.4% s.d.) for HYDRA - a statistically significant increase (paired, two-tailed t-test at 95% confidence). On the DNA promoters data set, the accuracy increases from 73.6% for FOIL to 81.1% for HYDRA.

5.4 Comparison with previous work

There has only been one previous method that we are aware of that tries to reduce the small disjuncts problem and that is the work of Holte *et al.* (1989) which uses a maximum specificity bias. There have been three other studies (Quinlan, 1991; Danyluk & Provost, 1993; Weiss, 1995) related to the small disjuncts problem although the studies did not demonstrate a method for reducing the error due to small disjuncts. In (Quinlan, 1991) it is shown that a modified form of the Laplace error estimate of a disjunct more closely models the "true" error of that disjunct (as estimated on a test set) than the Laplace error estimate. Danyluk & Provost (1993) show that in the domain of diagnosing errors in a telephone network, deleting rules that covered fewer examples than a specified percentage of the class leads to an increase in the overall error rate of the RL rule-learning algorithm (Clearwater & Provost, 1990). The third study was an extensive one done by Weiss (1995) that examined the effect that factors such as level of class noise and number of irrelevant attributes have on learning small disjuncts. These methods and studies will now be reviewed and compared to the approach of using reliability estimates as described in the previous sections.

Holte *et al.* (1989) explore a maximum specificity bias for small disjuncts. Given that some set of examples has been selected as belonging to the same disjunct, the maximally specific rule covering that set is the set of all conditions that are true for all the examples in that set. Normally, covering algorithms (CN2 (Clark & Niblett, 1989), FOIL (Quinlan, 1990), HYDRA (Ali & Pazzani, 1993), AQ15 (Michalski, 1986)) choose to describe such a set with a small set of conditions that differentiates that set from examples of other classes. If there is another condition that is also true for all members of that set, the maximum specificity bias will add it to the rule but CN2 etc. will not normally add it to the rule. Holte *et al.* show that this bias reduces the error rate for the small disjuncts but increases the error rate for large disjuncts so that overall error rate increases. They then explore a third type of bias, called "selective specificity" which they show is able to reduce the small disjuncts error rate and only leads to a small increase in error rate for the large disjuncts. However, they are unable to reduce the overall error rate. The main goal of the research described in this chapter is to reduce error rate of small disjuncts in a way that the overall error rate will decrease. The method of Holte *et al.* is unable to do this whereas the method of reliability estimates presented in this thesis both leads to a reduction in the error rate of small disjuncts and in the overall error rate.

Quinlan (1991) compares the error estimate of a disjunct as estimated from the training set to the "true" error of the disjunct as evaluated on a test set of examples. He shows that the discrepancy, the absolute difference of these two estimates, is lower for his contextualized Laplace error estimate than it is for the usual Laplace error estimate. The usual Laplace error estimate of a disjunct (as estimated on the training data) for class C that covers n training examples, e of which are from classes other than C is $\frac{e+1}{n+2}$ for a 2-class data-set.

In order to understand his contextualized Laplace error estimate, it is necessary to use a more general form of the Laplace estimate called the m-estimate (Smyth & Goodman, 1992):

$$P(E) = \frac{e + mp_0}{n + m} \tag{5.1}$$

where the event E has been observed to occur in e trials out of n and the prior probability of E is p_0. m is the "effective sample size" to associate with the prior probability of p_0. One can think of m as the number of examples with which one wishes to weight the prior probability.

Quinlan adapts this to compute the error rate of small disjuncts as follows. Let $\mathcal{C}$ denote the majority class of the disjunct - therefore, if the disjunct were a rule, it would be a rule for class $\mathcal{C}$. He first computes the set, S^+, of training examples of class $\mathcal{C}$ which satisfy *all but one* of the conditions in the disjunct. A similar set, S^-, is computed for the training examples of other classes. The error rate of the disjunct over this "context" (sets S^+ and S^-) is then $D = \frac{|S^-|}{|S^+|+|S^-|}$. Quinlan uses this for the p_0, the "prior" error rate of the disjunct. In order to use the m-estimate, however, one must also supply a value for m. Quinlan uses the ad-hoc value of $m = \frac{1}{1-D}$.

The resulting error estimates made from the training sample are shown to be closer to the "true" error estimates (on test examples) than the Laplace error estimate on five domains with lopsided class distributions in which the minority class is a small percentage of the overall training set. This work differs from that described in this thesis in that HYDRA is able to reduce the error due to the small disjuncts problem whereas Quinlan is concerned with predicting that amount of error. However, one could incorporate his disjunct error estimates into the gain metric and doing so should lead to a decrease in overall error rate.

Danyluk & Provost (1993) show that excluding rules that cover below a certain percentage of the training examples of that class causes the overall error rate to increase on a set of test examples (experiments on one domain with the RL (Clearwater & Provost, 1990) a rule-based learner). However, their conclusions depend on one domain and on the critical fact that the *cumulative* coverage of the small disjuncts in that domain was quite large. In early experiments with HYDRA in the KRK domain, deleting such rules also led to increased error rates because deleting such rules led to a significant increase in omissions. For some other learning algorithm that may be more prone to the small disjuncts problem, keeping such rules might not be worth it. However, the one-per-class approach coupled with reliability measures is able to enjoy the reduction in omission rates that comes with keeping such rules whilst also managing to reduce the small disjuncts problem that such rules can cause.

Weiss (1995) studies the effect of the following factors on error rate in artificial domains (parity function and voting function) in which it is possible to systematically vary the factors below.

- **Attribute noise** - it is shown that small disjuncts are a relatively greater problem for low levels of noise. As noise increases, large disjuncts also become significantly error prone. If there is no noise in the testing data, a relatively greater proportion of the error is due to small disjuncts than when there is noise in the test data (as there is for natural data sets).
- **Missing relevant attributes** - detailed experiments were not done. Weiss concludes "the results are expected to be consistent with those found for learning with attribute noise."

- **Class noise** - using results not presented in (Weiss, 1995), it is concluded that class noise is much *less* harmful to small disjunct error rates than attribute noise.
- **Training set size** - as the training set increases in size, the large disjuncts' error rates decline first so that the small disjuncts come to dominate the error rate. Finally, at asymptotic levels, the error rates of the small disjuncts also decline.

However, Weiss' study is not prescriptive - unlike the results of the one-per-class approach with reliability weights, it does not suggest how one might go about reducing the error contribution of small disjuncts.

5.5 Discussion

The results presented in this chapter show that unlike some previous research, the method of reliability estimates reduces the contribution to error from small disjuncts *and* reduces the overall error rate. In particular, it is shown, that FOIL's small disjuncts problem on DNA and King-Rook-King is reduced by HYDRA which uses reliability estimates and a one-per-class model. It is our hypothesis that because rules of differing classes have to compete to claim test examples in HYDRA, test examples which correspond to small disjunct errors in FOIL will mostly be correctly classified in HYDRA because most of the time there is a rule from the correct class that will also cover that example. This is demonstrated on the cumulative error distribution graph that shows that FOIL small disjuncts that cumulatively cover 20% of the test examples are responsible for 60% of the errors on the King-Rook-King data set with 160 training examples and 5% attribute noise. The small disjuncts learned by HYDRA, on the other hand, that cover 20% of the test examples are only responsible for 40% of the overall error rate.

Chapter 6
Learning probabilistic rules from noisy relational data

This chapter shows how HYDRA is better able to learn from noisy data than FOIL. The chapter starts with a review of Section 2.5 in which classification and attribute noise and the task of learning from noisy data are defined. The links between noise, overfitting and poor classification accuracy are explained. Then an empirical evaluation is presented showing that HYDRA is able to obtain significantly higher accuracies than FOIL on noisy data sets.

HYDRA differs from FOIL in three ways: (1) it learns a concept description per class, (2) it attaches reliability estimates to rules and (3) it uses a different gain metric (ls-content). In Section 6.3 the contribution of each of these changes towards accuracy improvement is discussed. Finally, the chapter concludes with a comparison of HYDRA with other methods for noise-tolerant learning of relational rules: MDL (Quinlan, 1990), selecting a more noise-tolerant gain metric (Dzeroski & Bratko, 1992) and cross-validation (Brunk & Pazzani, 1991).

Now a review of Section 2.5 is presented. Class noise is present in a set of examples if for at least one training example, the class label associated with that example as presented to the learning algorithm is not the true class label of that example. Attribute noise is present if for some example, the value for some component (or attribute) of the example as included in the training set is not equal to the true value of that component. The attribute noise dealt with in this thesis is transcription attribute noise in which the probability of large perturbations of component values is just as likely as smaller perturbations.

Noise leads to poor classification accuracy if the learning algorithm only allows the learning of pure rules (rules that cover training examples of only one class). Consider a simple two-class learning situation (classes `a` and `b`) and assume that a concept description for class `a` is being learned. Now due to class noise it may happen that some of the examples labeled as class `b` are really class `a` and hence share the same structure as other examples of class `a`. The phrase "share the same structure" means that the examples mislabeled as belonging to class `b` typically satisfy rules satisfied by many of the true examples of class `a` whereas very few of the true examples of class `b` satisfy those rules. The learning algorithm will have difficulty finding rules that cover many examples of class `a` but exclude these mislabeled examples. So, instead it will end up learning many rules, each of which covers a few training examples of class `a`. The same will happen when it learns a description for class `b`. Experiments (Muggleton et al., 1992) have shown that for most problems, rules that cover fewer training examples tend to make more classification mistakes on test examples and so lead to an overall reduction in classification accuracy.

Previous work in noise-tolerant learning of relational rules includes the minimum description length method (MDL, e.g. (Quinlan, 1990), (Muggleton et al., 1992)), statistical tests and the role of the gain metric (e.g. Dzeroski & Bratko (1992)) and cross-validation (Brunk & Pazzani, 1991). The next section presents the novel approach used in HYDRA to avoid overfitting when learning from noisy data sets. Section 6.4 compares this approach to previous work in noise-tolerant learning of relational concept descriptions.

6.1 HYDRA's approach to noise-tolerance

HYDRA differs from FOIL in three ways: (1) in learning a separate class description for each class, (2) in attaching reliability estimates to the learned rules and (3) in using a gain metric more suitable for learning in noisy domains. These decisions were made to develop a learning algorithm that is specifically better than FOIL on noisy domains even if that means suffering a slight loss in accuracy on noise-free domains. For the experiments described in this chapter, unless otherwise indicated, LS was used as the reliability weight. It is our belief that the results would not be significantly different if some other "reasonable" form of reliability weight, such as Laplace accuracy (c.f. Section 4.7) were to be used instead.

6.2 Comparison with FOIL

This section begins with a comparison of HYDRA to FOIL. This is a "horse-race" type comparison since the algorithms differ in more than one way (they differ in three ways). Horse-race comparisons can be contrasted to "scientific" comparisons in which two algorithms only differ in one factor. This latter type of comparison is useful for inferring how that factor affects classification accuracy and other performance measures. The "horse-race" comparisons are more useful for answering application-oriented questions such as "what algorithm should I use on noisy, relational data?"

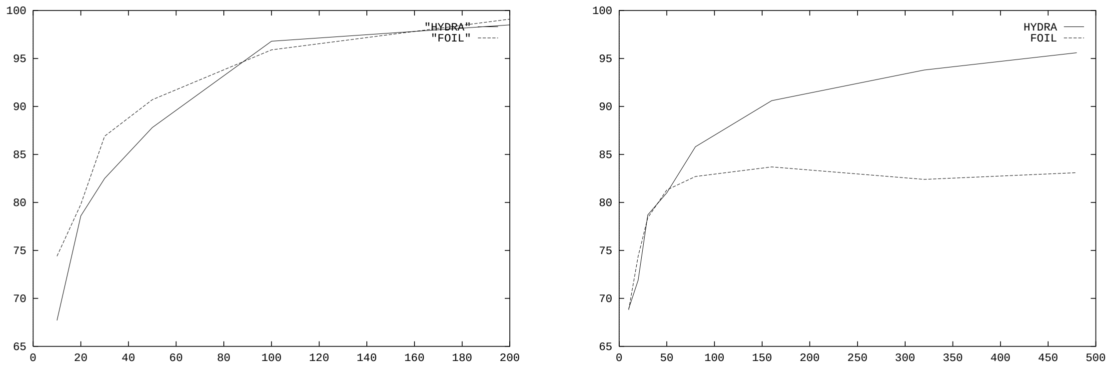


Figure 6.1: Comparison of FOIL and HYDRA on noise-free KRK domain (left) and 20% class-noisy KRK domain (right). The x-axis corresponds to number of training examples and the y-axis to accuracy. The graph shows that both algorithms are approximately equal on the noise-free data but that HYDRA has a significant advantage on the noisy data (as planned).

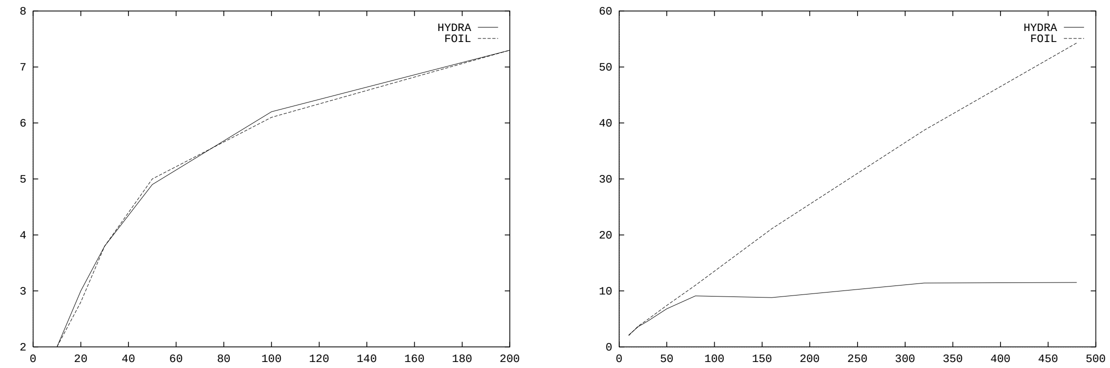


Figure 6.2: Comparison of FOIL and HYDRA on noise-free KRK domain (left) and 20% class-noisy KRK domain (right). The x-axis corresponds to number of training examples and the y-axis to number of learned clauses for class `illegal`. The graph shows that FOIL learns many more clauses than HYDRA on noisy data suggesting that its low classification accuracy on such data sets is due to overfitting the noise.

Table 6.1: Comparison of FOIL and HYDRA on various data sets. A "+" indicates a statistically significant increase in accuracy according to the paired, 2-tailed t-test at 95% confidence level. Standard deviation is indicated in parentheses for the accuracies columns. Standard deviation is not indicated for the promoters domain because leave-one-out testing was used on that domain. Entries in the last column are vectors because HYDRA builds a set of clauses for each class. LS-content was used for learning and LS weights were associated with learned rules for HYDRA.

Domain	Number of training examples	FOIL accuracy	FOIL # of clauses	HYDRA accuracy	HYDRA # of clauses
Without noise					
KRK	10	+ 74.4 (9.8)	2.0	67.7 (11.1)	(2.0,1.3)
KRK	20	79.8 (10.6)	2.8	78.6 (9.0)	(3.0,2.5)
KRK	30	+ 86.9 (5.6)	3.8	82.5 (6.7)	(3.8,3.2)
KRK	50	+ 90.7 (3.9)	5.0	87.8 (4.5)	(4.9,4.4)
KRK	100	95.9 (3.6)	6.1	96.8 (2.2)	(6.2,5.8)
KRK	200	99.1 (1.2)	7.3	98.5 (1.3)	(7.3,7.5)
With 20% class noise					
KRK	10	68.8 (11.1)	2.0	68.9 (14.6)	(2.1,1.8)
KRK	20	74.4 (9.9)	3.7	71.9 (11.2)	(3.6,3.1)
KRK	30	78.4 (8.3)	4.9	78.7 (7.1)	(4.6,4.0)
KRK	50	81.3 (6.9)	7.4	81.0 (6.7)	(6.8,5.8)
KRK	80	82.7 (5.8)	11.0	+ 85.8 (4.3)	(9.1,7.4)
KRK	160	83.7 (5.0)	21.1	+ 90.6 (3.5)	(8.8,8.1)
KRK	320	82.4 (2.5)	38.7	+ 93.8 (2.8)	(11.4,9.9)
KRK	480	83.1 (2.7)	54.3	+ 95.6 (2.6)	(11.5,13.4)
Natural data sets					
Cancer	190	63.5 (4.3)	29.5	66.6 (5.2)	(5.3,7.8)
KPa7KR	200	90.3 (2.5)	7.6	+ 94.7 (1.1)	(7.6,8.3)
Promoters	105	73.6	4.0	76.4	(5.0,6.1)

Table 6.2: Comparison of FOIL and HYDRA on 17 data sets. "+" indicates HYDRA was statistically significantly more accurate, "–" indicates it was statistically significantly less accurate.

Data set	FOIL Accuracy	HYDRA Accuracy
Tic-Tac-Toe	99.4	– 98.4
KRKP	92.7	+ 94.7
KRK100	96.4	– 95.3
KRK200	98.8	98.3
Average	96.8	96.7
KRK160e.5a	91.5	+ 93.5
KRK320e.5a	92.0	+ 96.6
KRK160e.20c	85.2	+ 92.1
KRK320e.20c	83.7	+ 95.5
Average	88.1	+ 94.4
DNA	72.6	65.1
Average	72.6	65.1
Mushroom	96.4	+ 97.6
Hypothyroid	97.4	– 95.2
B.Canc.Wisc	92.6	+ 94.0
Voting	93.2	93.3
Horse-colic	81.6	+ 84.6
Hepatitis	70.6	+ 77.5
Diabetes	67.4	+ 72.9
B.Cancer	62.7	+ 70.4
Average	82.7	85.7
Overall Average	86.7	89.1

Table 6.1 and Figure 6.2 compare FOIL's accuracy to that of HYDRA, first on noise-free KRK examples and then on noisy data sets. The natural data sets are considered noisy as their target concepts may not be expressible in DNF form using the available attributes or relations. FOIL is able to attain lower error rates only for noise-free data of the concept `illegal` from the KRK domain because that concept is easily expressible using first-order Horn clauses. On the noise-free KRK data set, FOIL is assured of an accuracy equal to the base accuracy of approximately 70% even if it builds a null concept description (consisting of no clauses). FOIL only builds concept descriptions for the positive class (illegal for the KRK domain). As the class legal occurs with probability approximately equal to 0.7, FOIL can attain approximately 70% accuracy by not building any clauses and just guessing the most frequent class. For noisy data from the KRK domain, FOIL's accuracy quickly climbs to approximately 82% as the number of training examples are increased but fails to increase substantially above that figure. The number of clauses built increases linearly with the number of noisy training examples, indicating FOIL is overfitting the data. Experiments we conducted with other noise levels for the KRK domain reaffirm the basic result that HYDRA performs significantly better than FOIL on noisy data. The reduction of the small disjuncts problem (Figure 5.1) is another reason why HYDRA is able to attain higher accuracies on noisy data sets than FOIL.

Table 6.2 shows another set of results comparing FOIL and HYDRA - this time on a greater number of 2-class problems. Information gain was used as the metric for FOIL and HYDRA. The results show that for this distribution of data sets, HYDRA out-performs FOIL by an average of 2.4% - usually a statistically significant margin. However, if we examine the two algorithms on the noisy data sets (the group from KRK160e.5a to KRK320e.20c), HYDRA wins by a large margin of 6.3%. In terms of error, the average error of HYDRA on those data sets at 5.6% is less than half that of FOIL (11.9%). On the natural data sets (Mushroom to Breast Cancer) HYDRA gets an average of 85.7% to FOIL's 82.7%. HYDRA is more accurate on all but one of those noisy, real-world data sets.

6.3 Study of the differences between HYDRA and FOIL

The remainder of this section examines a progression of algorithms from FOIL to HYDRA that allows one to see the impact of each of the three differences between FOIL and HYDRA. Figure 6.3 shows the comparisons that will be made. Note that the comparison between Algorithm 3 and HYDRA (Algorithm 4) is a "lesion" study, because exactly one component of HYDRA (the conflict resolution method used to determine the class when clauses from more than one class are

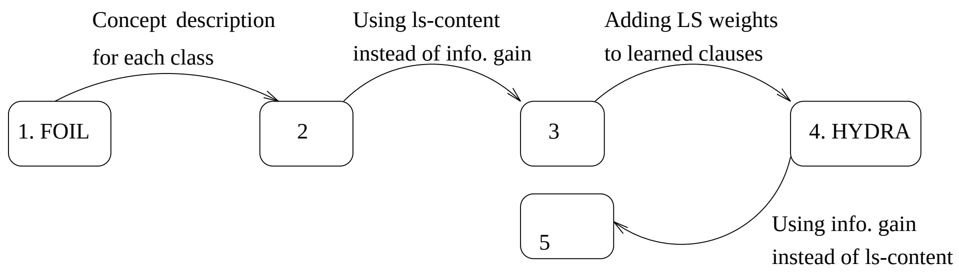


Figure 6.3: A progression of algorithm transformations that converts FOIL into HYDRA. This progression allows one to see the impact of each of the three differences between FOIL and HYDRA.

satisfied) is changed to form Algorithm 3. Similarly, the comparison between HYDRA and Algorithm 5 is a lesion study looking at the impact of the gain metric.

First, FOIL (Algorithm 1 in Figure 6.3) is compared to a version of FOIL (Algorithm 2 in Figure 6.3) that only differs from FOIL in that it learns a concept description for each class. When classifying an example, if an example satisfies a clause of more than one concept description, Algorithm 2 needs to estimate the reliability of the contending clauses in order to resolve this conflict. HYDRA resolves such conflicts by selecting the class with the highest posterior odds (product of prior odds and LS of the most reliable satisfied clause). Algorithm 2 resolves conflicts by selecting the class corresponding to the clause with the greatest coverage (number of positive training examples). These approaches only differ if the learned clauses cover negative training examples. When a test example satisfies no clause of any class, Algorithm 2 guesses the most frequent class.

As Table 6.3 indicates, learning a description for each class (Algorithm 2) yields significant increases in accuracy on noisy data. This is because it makes fewer errors of commission than FOIL. Consider the KRK domain. FOIL only learns a definition for the concept `illegal` so an error of commission occurs when it classifies an example of class `legal` as belonging to `illegal`. Algorithm 2 may not make such an error in the same situation if that example also satisfies a clause of `legal` that is more reliable than the contending clause from `illegal`. Thus, Algorithm 2's conflict resolution strategy would correctly award that example to `legal`. For the noisy KRK, cancer and lymphography data sets, such conflicts occur on approximately 20% of the test examples. On approximately 1 to 2% of the test examples in these domains, HYDRA has to guess the most frequent class. However, in the promoters domain, Algorithm 2 has to guess the most frequent class on about 30% of the test examples. Since both classes are approximately equally frequent in this data set, guessing the most frequent class is a very bad strategy in this domain. This underlines the need for reliability weights when making classifications in a one-per-class situation.

Table 6.3: Progression from FOIL to HYDRA. A "+" indicates a statistically significant improvement (using a paired t-test at 95% confidence) as measured with respect to the column immediately to the left. A "-" indicates a significant decline. All figures represent accuracies averaged over 20 trials except for DNA for which leave-one-out testing was used.

Domain	Number of eg.s	FOIL 1	Concept desc. per class (2)	Using ls-content (3)	Adding LS weights (HYDRA) (4)	Replacing ls-content w. info. gain (5)
Noise-free						
KRK	100	97.0	95.9	95.0	96.8+	97.1
KRK	200	99.1	98.2+	97.9	98.5	98.6
With 20% class noise						
KRK	160	83.9	87.3+	85.9	90.6+	88.0–
KRK	320	83.8	90.9+	81.5–	93.8+	91.4–
With 5% tuple noise						
KRK	160	90.6	91.0	89.6	93.3+	92.4
KRK	320	90.7	94.0+	89.0–	96.1+	94.9
Natural data sets						
Cancer	190	63.5	66.4+	62.5–	66.6+	63.8–
Lymph.	99	–	79.4	80.5	81.4	78.8
Kpa7KR	200	90.3	94.1+	94.7	94.7	94.2
DNA	105	73.6	61.3+	76.4+	76.4	70.8–

Now consider the effect of replacing information gain by ls-content (the transition from Algorithm 2 to Algorithm 3). Algorithm 3 leads to a decrease in accuracy on all the variants of the KRK domain and the cancer domain. The drop in accuracy on the noise-free KRK data is due to the fact that ls-content weighs coverage more highly than purity, leading it to build fewer clauses, each of which may cover some negative examples. That is, ls-content causes a greater number of conflicts so it needs a good way of resolving conflicts: coverage is not as good a way as LS. That is why Algorithm 3 does significantly worse than Algorithm 2 on the very noisy data sets (KRK 320,20% class noise; KRK 320,5% tuple noise and Breast Cancer).

Algorithm 3 does not do poorly on the DNA and lymphography data sets because those are relatively noise-free and the learned clauses do not cover examples of other classes so the conflict rate is low. Another reason Algorithm 3 does better than Algorithm 2 is that on these domains, Algorithm 2 suffers from many errors of omission. On such occasions, Algorithm 2 has to guess the most frequent class. Algorithm 3, using ls-content builds clauses with greater coverage so its has to guess less frequently and consequently has a higher accuracy on these data sets.

Next, consider the impact of adding LS weights to the concept description learned by Algorithm 3. This is Algorithm 4, which we call HYDRA. Table 6.3 shows that using LS to resolve conflicts rather than coverage helps for the KRK and some other domains and does not hurt on the other domains. This increase in accuracy is due not to building a different concept description but simply to using a better conflict resolution strategy - one based on LS rather than coverage.

These two methods only differ when comparing clauses that cover negative training examples (which happens on the noisy KRK data sets). Table 6.3 shows that HYDRA (using LS's to resolve conflicts) yields significantly higher accuracies on all four of the noisy KRK data sets than using positive training coverage to resolve conflicts.

Finally, consider a lesion study in which HYDRA is run using information gain rather than ls-content (transition from Algorithm 4 to Algorithm 5). This transition studies the effect of the gain metric and should be considered in conjunction with Table 4.4 which studies the effect of the gain metric on a larger number of domains. The accuracies obtained on the ten data sets that are in common between the two tables are slightly different because a different set of thirty training/testing partitions was used in each study.

On the ten data sets presented in Table 6.3, ls-content did significantly better than information gain on the 20% class-noisy KRK data sets, Breast Cancer and Promoters. On different partitionings from these same domains as presented in Table 4.4, ls-content was *not* significantly better than information gain. So conservatively, one would have to conclude that there is no significant difference between information gain and ls-content on 20% class-noisy KRK data sets. However, ls-content is still superior to information gain on the highly disjunctive DNA data set. On the Breast Cancer data set, although ls-content is not superior at the 95% level, it is superior at the 90% level thus agreeing with Table 4.4. One reason HYDRA with information gain is less accurate than HYDRA with ls-content is that the information-gain version learns over-specific clauses leading it to make many errors of omission. In summary (as Table 4.4 shows) ls-content is primarily superior to information gain on the noisy natural data sets. There is not any significant difference on noisy artificial data sets.

6.4 Previous approaches

Previous work in the area of noise-tolerant learning of relational rules includes the minimum description length method (MDL, e.g. (Quinlan, 1990), (Muggleton et al., 1992)), statistical tests (e.g. Dzeroski & Bratko (1992)) and cross-validation (Brunk & Pazzani, 1991). These will discussed in turn and compared to the approach of reliability estimates used in HYDRA.

6.4.1 MDL

The Minimum Description Length Principle (MDLP, Rissanen, (1978) ; Quinlan & Rivest (1989)) is an indirect estimator of the generalization (off-sample) error. According to the principle, one should choose the model whose encoding length is minimal with respect to the given encoding scheme. Therefore, the method requires a coding scheme as a parameter. Ideally (Shannon & Weaver, 1949) it should take $-lg(P(Model|Data))$ bits to encode a model whose posterior probability is $P(Model|Data)$.[1] Now we consider the application of the MDLP to learning first-order Horn clauses.

The MDL paradigm views learning a model analogously to compressing data for transmission. In this paradigm, there is a receiver of information and a sender of information. Both parties have access to the training data in the same order and have agreed upon a scheme for encoding data and for encoding concept descriptions. It is up to the sender to send the classifications of the training data in a manner which will minimize the length of the transmission in bits. For example, if the data is highly regular with respect to the alphabet and concept descriptions agreed upon by the two parties, the length of the transmission will be short. One important thing to note here is that information theory (Hamming, 1986; Chaitin, 1987; Solomonoff, 1964) dictates that the expected length of messages will be minimal when the coding scheme takes $-lg(P)$ bits to encode a model whose posterior probability given data is P. In this theory it is analytically shown that a coding scheme that does not satisfy this property has a sub-optimal (longer) expected coding length. In practice, the selection of a coding scheme that approaches this ideal is a significant issue. Equivalently, the choice of a coding scheme can be seen as choosing a prior probability distribution over the model space. Typically, a function for the prior is chosen that assigns lower prior probabilities for more complex models. The reasoning is that more complex models are less likely *a priori* as good explanations for data. However, there are many probability distributions that assign smaller *a priori* probabilities to more complex models and that sum up to 1 over the model space.

In MDL applications to learning, two kinds of encoding need to be specified: one to encode the learned model, another to encode the exceptions to that model. Hopefully, the sum of the lengths of the encodings of the model and exceptions will be much less than the encoding length of the extensional form of the training set. In this way, compression will be achieved. MDLP is an estimator of generalization-error and it is based on the assumption that models with greater posterior probability (and hence, greater compression) will have smaller generalization error. *So compression is equivalent to generalization in this view.*

[1] See Section 4.9 for how the posterior probability of a model consisting of class descriptions consisting of rules is computed.

Quinlan's MDL modification to FOIL involves a method for encoding examples and a method for encoding a literal. This method is used in learning as follows. It only conjoins literals to the clause being learned as long as the sum of the encoding lengths (in bits) of the literals in the clause is less than the encoding length of the examples covered by the clause; i.e. as long as there is positive compression. The encoding length of the examples is defined to be

$$lg(T) + lg\begin{pmatrix} T \\ p \end{pmatrix} \tag{6.1}$$

for p positive examples out of a total of T examples. The encoding length of the clause is defined to be

$$(\sum_i 1 + lg(R) + lg(V_i)) - lg(n!) \tag{6.2}$$

where the sum is taken over the literals in the clause. R is the number of background relations, n is the length of the clause so far and V_i is the number of variabilizations[2] of the i-th literal with respect to variables found in the head of the clause or introduced by the previous $i-1$ literals (the literals are indexed in the order they were learned, "left to right"). See Quinlan (1990) for a justification of these formulae. These encoding schemes are incorporated into an algorithm that stops conjoining literals to the clause if the encoding length of the clause exceeds the encoding length of the remaining training data. In this manner, it is hoped that by avoiding overly complex descriptions, overfitting is avoided.

However, Brunk and Pazzani (1991) experimentally showed that the encoding length approach applied to FOIL is ineffective because it results in less accurate concepts on noisy data sets. They also show that another algorithm for avoiding overfitting, Reduced Error Pruning (REP; Quinlan (1987)), yields more accurate concept descriptions. Since REP has been empirically shown to be more accurate than Quinlan's MDL formulation, HYDRA will only be compared to REP (Section 6.4.2).

One reason that the MDL approach may not do so well is that the two different encoding schemes (one for data, the other for intensional descriptions) need to be calibrated: a bit in one is not equivalent to a bit in the other (this trade-off factor will be denoted as f). Actually, this is just the problem of choosing a suitable prior distribution in another guise.

The limitations of the MDL approach are tied to the choices that must be made regarding the encoding scheme and the functional form of the prior distribution. Unfortunately, experiments (Quinlan & Rivest, 1989), have shown that the size of the model and off-sample accuracy are somewhat sensitive to settings

[2] A variabilization is a list of atoms (such as (X, Y)) which together with the name of a predicate (such as a) is used to syntactically construct a literal ($a(X, Y)$).

Table 6.4: Comparison of HYDRA and REP. Figures in parentheses for accuracies columns indicate standard deviations. "+" indicates statistically significantly higher accuracy. The table shows that for small sample sizes REP obtains lower classification accuracies than HYDRA. For large sample sizes, REP is prohibitively expensive, its running time is $\Omega(n^4)$ where n denotes the number of training examples. Although CPU times were only measured for the class noisy KRK data sets, REP was similarly inefficient on the other data sets. LS-content was used for learning and LS weights were associated with learned rules for HYDRA.

Domain and # of training examples	HYDRA accuracy	HYDRA # of clauses	HYDRA cpu seconds	REP accuracy	REP # of clauses	REP cpu seconds
With 20% class noise						
KRK10	+ 68.9 (14.6)	(2.1,1.8)	2	63.0 (13.4)	0.3	1
KRK20	71.9 (11.2)	(3.6,3.1)	5	69.6 (8.9)	0.9	3
KRK30	+ 78.7 (7.1)	(4.6,4.0)	9	75.4 (8.5)	1.2	6
KRK50	+ 81.0 (6.7)	(6.8,5.8)	27	78.0 (8.7)	2.2	21
KRK80	+ 85.8 (4.3)	(9.1,7.4)	53	82.2 (7.0)	2.8	68
KRK160	90.6 (3.5)	(8.8,8.1)	111	91.2 (4.4)	4.3	160
KRK320	93.8 (2.8)	(11.4,9.9)	252	92.7 (3.3)	6.5	1705
KRK480	95.6 (2.6)	(11.5,13.4)	340	96.3 (2.3)	6.9	7950
With 5% attribute noise						
KRK10	71.3 (11.6)	(1.9,1.7)	-	67.5 (10.0)	0.4	-
KRK20	+ 78.1 (7.9)	(3.1,2.6)	-	74.3 (8.1)	1.0	-
KRK30	+ 83.6 (7.1)	(4.3,3.9)	-	74.9 (7.9)	1.1	-
KRK50	+ 86.8 (4.8)	(5.5,5.2)	-	81.7 (7.9)	1.9	-
KRK80	91.5 (3.4)	(8.0,6.7)	-	85.5 (7.3)	2.9	-
KRK160	93.3 (3.3)	(10.1,8.9)	-	91.7 (3.5)	3.9	-
KRK320	96.1 (1.2)	(13.3,11.5)	-	95.7 (2.1)	5.7	-
KRK480	97.5 (1.2)	(17.6,16.3)	-	97.1 (1.9)	7.0	-
Natural data sets						
B.cancer190	66.6 (5.2)	(5.3,7.8)	-	67.9 (4.5)	3.4	-
Lymph.99	81.4 (4.2)	(1.6,8.4,7.6,1.0)	-	-	-	-
KPa7KR200	+ 94.7 (1.1)	(7.6,8.3)	-	91.1 (2.3)	2.9	-
Promoters105	76.4	(5.0,6.1)	-	78.0	1.9	-

of the trade-off factor f. More work on the factors causing sensitivity to f and the optimal setting of f is needed. More work is also needed on exploring the sensitivity to the form of the coding scheme and the functional form of the prior distribution.

6.4.2 Cross-validation

REP (Reduced-Error Pruning, Quinlan (1987)) splits the training data into two sets which we will call the learning set and the pruning set. The pruning set is sometimes called the *hold-back set* or the *validation set*. Although REP was first used in the context of post-pruning of decision trees[3] we will explain it here in the context of post-pruning a class description consisting of a set of rules. This work was done by Brunk & Pazzani (1991). This method uses the idea behind

[3]Post-pruning means the pruning occurs after learning.

cross-validation in which a separate set of examples (the pruning set) is used to validate the description learned on the learning set of examples.

FOIL is used to create a concept description using only the learning data. REP takes the concept description and applies operators (delete-clause and delete-last-literal) to make small syntactic transformations to the concept description. It measures the merit of each operator application by comparing the numbers of correct classifications (on the pruning data) before and after application of the operator. The transformations resulting from the best operator application are kept and the process is repeated until the application of any operator would result in a decrease in accuracy. The idea behind REP is that noise is unlikely to affect two training sets in the same manner. Hence, a clause that is accurate on the learning set due to being overfitted for that set will probably not be accurate on the pruning set. However, a "good" clause that belongs in the target description will probably be accurate on the learning set *and* the pruning set.

Table 6.4 compares HYDRA and REP. For these experiments, the pruning set size was 30% of the training set. HYDRA is significantly more accurate than REP when there are a limited number of training examples because REP over-prunes when the pruning set is too small. To see this, assume that each learned clause corresponds to a subclass and that because the pruning set is so small, it is missing examples from several subclasses. In this situation, REP will use its `delete-clause` operator to prune away these clauses, probably leading to a lower accuracy on the much larger test set which does contain examples form those subclasses. The table illustrates that for small numbers of examples from the KRK domain, very few clauses remain after REP is done (e.g. on average, 0.3 clauses remain when the training set has 10 examples; by contrast, FOIL learns 2 clauses with 10 examples). REP prunes away almost all the concept description (column 6) so its reasonable accuracy is a result of the fact that even the null concept description attains approximately 70% accuracy in this domain. More detailed experiments show that REP is essentially guessing the globally most frequent class - it is highly accurate on test examples of the majority class and highly inaccurate on test examples of the minority class. Asymptotically, both algorithms approach 100% accuracy on the noisy data sets from the KRK domain. However, in the range where both algorithms are approximately equal in accuracy, REP is much more expensive in terms of learning cost. For example, at 320 examples with 20% class noise in the KRK domain, REP takes 7 times as long as HYDRA and is still 01.1% less accurate. Although Table 6.4 measures complexity in cpu seconds, the differences in complexity are large enough such that a first order approximation is sufficient here. This is substantiated by Cohen (1992) who demonstrates that REP has an asymptotic time complexity of $\Omega(n^4)$. HYDRA by comparison, only needs to evaluate each learned clause over the training set once, in order to estimate the LS values. In summary, HYDRA is more accurate than REP on small training sets and equally accurate but much more computationally efficient on larger training sets.

6.4.3 mFOIL

In this section, experiments comparing HYDRA to mFOIL (Dzeroski & Bratko, 1992) are presented. mFOIL is an algorithm that uses beam search and a different gain metric to make a noise-tolerant version of FOIL. Although Dzeroski & Bratko also describe the use of the Likelihood Ratio statistical test in mFOIL, their experiments were done with this option turned off.

mFOIL (Dzeroski & Bratko, 1992) is an extension to FOIL 2.1[4] which differs from FOIL 2.1 in its use of the Laplace estimate of rule accuracy as a gain metric (FOIL 2.1 uses information content). Other significant differences include the use of beam search (with a beam width of 5) instead of the greedy search used by FOIL 2.1. Both FOIL 2.1 and mFOIL use Quinlan's MDL-based method (Section 6.4.1) to prevent overfitting.

In their experiments, the Dzeroski & Bratko compare FOIL 2.1, mFOIL using Laplace estimate of rule accuracy ($p(Class_i|rule_{ij}(\tau) = true)$) as a gain metric and mFOIL with various settings of m for the m-estimate as a gain metric. The m-estimate of rule accuracy is:

$$\text{m-estimate} = \frac{r+m\times p_0}{n+m}$$

where the rule for class C covers n total examples out of which r are of class C and p_0 denotes the prior probability of that class. The m-estimate is a generalization of the Laplace estimate $\frac{r+1}{n+2}$ with $m = 2$ and $p_0 = 0.5$. m can be thought of (Smyth *et al.* (1990)) as the "effective number of examples" to be used as a weight for the prior probability estimate of $p(Class_i|rule_{ij}(\tau) = true)$.

The experiments of Dzeroski & Bratko show that mFOIL using the Laplace estimate is as accurate as FOIL 2.1. However, using $m = 0.01$ (the best overall value of m), mFOIL is more accurate than FOIL 2.1. As expected, the experiments indicate that as the level of class noise is increased, the value of m at which the algorithm performs best increases. This is expected since at high levels of noise, greater weight should be put on the prior estimate (0.5) of rule accuracy.

Our experiments compare HYDRA to mFOIL using the Laplace gain metric and also to mFOIL using the best value of m - 0.01. Since the learning algorithms

[4]FOIL 2.1 differs from FOIL in its use of determinate literals - literals that have no information gain, but introduce a new variable which subsequent literals may use to discriminate negative from positive examples.

Table 6.5: Comparison of mFOIL to HYDRA on the KRK domain. All results are for 100 training examples from the KRK domain. Accuracies for mFOIL are taken from (Dzeroski, 1992) (rather than being run on the same training and testing sets) so these comparisons with HYDRA are not paired and cannot use significance tests. A "+L" indicates that HYDRA's accuracy was substantially greater (more than 2% greater) than that of mFOIL using Laplace. "+M" indicates HYDRA was substantially more accurate than mFOIL using its best m value.

Level of noise	mFOIL using Laplace	mFOIL m = 0.01	HYDRA accuracy
Class noise			
0%	95.1	95.6	96.8
5%	92.7	94.3+	91.9
10%	88.9	92.0	91.0+L
15%	86.4	90.0	88.9+L
20%	84.6	88.1	88.9+L
Tuple noise			
5%	90.1	91.9	90.0
10%	80.4	84.6	86.8+L+M
15%	78.3	80.1	83.0+L+M
20%	73.7	75.0	79.7+L+M

were run on different learning sets, significance tests cannot be done here so a difference of 2.0% is considered a "substantial" difference. This value comes from experience with t-tests at 95% confidence levels. The conclusions are of the comparisons between HYDRA, mFOIL using Laplace and m-FOIL using m-estimate with $m = 0.01$ are:

1. At low levels of noise there is no significant difference between any of the algorithms.
2. At high levels of class noise, HYDRA is significantly more accurate than mFOIL using Laplace and equally accurate to the best m value for mFOIL. However, since the best m value is not known ahead of time, HYDRA would be easier to use for such data sets.
3. For tuple noise, HYDRA is significantly more accurate even more often than for class noise - it is more accurate than even the best fitted value of m. It is not clear why the relative performance of the systems varies between class noise and tuple noise. Unfortunately, since there are so many differences between mFOIL and HYDRA (mFOIL uses a beam width of 5 and a different gain metric and does not build a concept description for each class and does not use LS as the weight associated with a learned rule) it is not clear why HYDRA is able to do so much better on tuple noise.

6.5 Discussion

The results of the comparison between HYDRA and FOIL can be summarized as follows:

1. HYDRA is significantly more accurate than FOIL on artificially and naturally noisy data sets. FOIL overfits the data and the number of clauses it learns grows linearly in the number of training examples (evidence: noisy KRK data, Figure 6.2).
2. FOIL fails to asymptote to 100% accuracy on noisy KRK data no matter how many training examples are used.
3. FOIL is more accurate than HYDRA on noise-free data sets. HYDRA is designed specifically to do well on noisy data sets even if that means sacrificing some performance on noise-free data sets.

Next, the chapter presented a progression of algorithms from FOIL to HYDRA. This allowed the impact of each of the three differences between FOIL and HYDRA to be studied in turn. The results of this study are as follows:

1. The first comparison is between FOIL and Algorithm 2 which learns a description for each class. This led to significant improvements in accuracy on 7 of the 10 data sets by reducing the commission error rates.
2. The second step involves changing the gain metric from information gain to ls-content. LS-content learns fewer clauses, each of which covers more examples and it also tends to learn clauses that cover a few examples of other classes. This change led to a significant decline in accuracy on noisy KRK data sets because LS-content learned clauses covering negative examples and positive training coverage is not a good way to resolve conflicts when the conflicting clauses cover negative examples.
3. Changing from information gain to LS-content helped on the highly disjunctive DNA domain since LS-content learns clauses that cover more examples. Thus it was able to reduce the high omission error rate on DNA.
4. Changing from resolving conflicts by positive training coverage to resolving by LS helped a lot on noisy data sets where the learned clauses covered examples of other classes. Thus, the accuracy increased substantially for noisy KRK data sets (comparing Algorithm 3 and Algorithm 4 - HYDRA).
5. Changing from using HYDRA with ls-content to using HYDRA with information gain: there was no difference on artificially-noisy data sets and ls-content was superior on some naturally noisy data sets.

HYDRA was compared to two systems (REP and mFOIL) for noise-tolerant learning from relational data. HYDRA was not compared to Quinlan's MDL approach since Brunk & Pazzani had already showed that REP was better than that MDL approach. The comparison of HYDRA and REP showed that for small training set sizes HYDRA was significantly more accurate since REP tended to over-prune. For large training set sizes, both algorithms were equally accurate but REP was much more computationally expensive.

In the comparison of HYDRA and mFOIL, at most class noise levels, HYDRA was more accurate than mFOIL using Laplace, and equally accurate as mFOIL using the m-estimate. For attribute noise, HYDRA was significantly more accurate than either version of mFOIL.

These results can also be seen as a validation of the one-per-class approach. This approach has been criticized by Dietterich & Bakiri (1995) as yielding inferior results as compared to the multi-class approach. However, the difference between our results arises due to a small but critical difference - in the experiments of Dietterich and Bakiri, conflicts were resolved randomly. In our approach, a more informed way of resolving conflicts leads instead to a validation of the one-per-class approach, at least as compared to Quinlan's implementation of MDL (Quinlan, 1990), Reduced-Error Pruning (Brunk & Pazzani, 1991) and mFOIL (Dzeroski & Bratko, 1992).

In summary, HYDRA does better than both REP and the mFOIL approach on the domains presented here. Most of the benefit of HYDRA comes from learning a description per class and then resolving any conflicts with the LS measure. This approach reduces the commission error rate. The benefit of the gain metric seems more marginal - as Section 4.9 shows the Bayes metric would do better than the ls-content metric. However both these metrics are better than information gain which tends to fit the noise more than these metrics.

Chapter 7
Multiple Models

This chapter presents approaches for addressing the problems caused by limited lookahead and the problem of generalizing from a limited sample. These approaches are all variants of the multiple models approach in which many models are learned from the same training set. The chapter also presents results that show that the multiple models method produces greater classification accuracy when compared to the accuracy of a single-model learned on the same training data.

The multiple model approach is also useful for describing concepts that need evidence combination. This is as opposed to concepts that can be accurately described using a succinct DNF form. For this latter type of concept, satisfaction of a single rule is sufficient to conclude membership in the class. However, for concepts that need evidence combination, many rules must be satisfied in order to conclude class membership. The multiple models approach facilitates evidence combination.

Two major issues in using the multiple models approach which are explored in this chapter are:

- How to produce multiple models of a single training set?
- How to combine the classifications (or evidence) from multiple models?

The chapter examines alternative approaches for these questions and identifies the kinds of data sets in which each approach is better than the other approaches. Since one of the motivations for using the multiple models approach is to improve the error rates of small disjuncts (Section 1.5), Section 7.6 will present results on how effective the multiple models approach is for dealing with small disjuncts. Finally, the chapter concludes with a review of previous comparisons of model generation methods and evidence combination methods and shows how the results found here supplement these earlier results.

Table 7.1. Top two candidate literals as given by a 1-step lookahead.

Rule	Coverage	Info. gain
$Class_1(X,Y) \leftarrow a(X,Y)$	(30,1)	20.7
$Class_1(X,Y) \leftarrow b(Y,X)$	(30,4)	16.7

7.1 The problem with the single model approach

Consider the problem of induction when there is no noise in the data. For any given training set and model language it is almost certain that there will be more than one model in that model space that is consistent with the training data. The problem is to find the model from this set of consistent models that generalizes best on examples not in the training set.

Information gain (or some similar metric) in conjunction with greedy search is often used to search through the space of models for a model that is both consistent with the training data and syntactically compact. However, for some data sets, during learning it may happen that the candidate (e.g. a literal in a rule or a node in a decision tree) with the highest information gain is flanked by other candidates that have "almost as much" information gain. It may be that due to the inclusion or exclusion of a few examples in the training set, the candidate that is truly the best appears to be second best. Such a candidate will be completely ignored by the single model approach in favor of the candidate that appears to be the best as judged from the training data. This, then is the first problem with the single model approach.

One way to solve this problem is to increase the amount of lookahead. Assume that information gain is being used as the gain metric and that the training set contains 2 classes, the first with 60 training examples, the second with 40 training examples. Table 7.1 shows the information gain for the top two literals: A coverage of $(20, 0)$ means that the rule covers 20 positive training examples (i.e. 20 training examples of the class - $Class_1$ - for which the rule is being learned) and 0 negative training examples. In this situation, FOIL will prefer the first rule since it has higher information gain. However, assume that if we go with the first choice, then the best additional literal that we can add to the rule to exclude the single negative example only covers 1 positive example (Table 7.2). Had we gone with the second choice in Table 7.1, we would have found that an additional literal can be found that covers 28 positive examples. In most domains, rules that cover just 1 training example tend to be error prone on the test set. And, had we evaluated the information gain of *pairs* of literals as in Table 7.2, even the information gain metric would have preferred the second choice in Table 7.1 over the first choice. This situation illustrates the problem of *limited lookahead.* Specifically, it illustrates the

Table 7.2. Top two candidate literals as given by a 2-step lookahead.

Rule	Coverage	Info. gain
$Class_1(X,Y) \leftarrow a(X,Y), c(X)$	(1,0)	0.74
$Class_1(X,Y) \leftarrow b(Y,X), d(Y)$	(28,0)	20.64

limitations of single-step lookahead. If 2-step lookahead had been used, the second choice in Table 7.1 would have been preferred.

However, the traditional approach of increasing the amount of search (lookahead) and then producing a single model still has limitations. Assume, for example, that the model space only consists of models that are conjunctive rules. Assume that the training set consists of 60 examples of the first class and 40 examples of the second class and that noise is present in the training set. Assume that the best two rules as found by some lookahead algorithm for this model space are as shown in Table 7.3. Even though there is no limitation due to search in this situation, one still cannot rule out the possibility that the second rule may have higher off-sample accuracy. The single model approach (even with unlimited lookahead) is limited to using one of the models in Table 7.3 whereas the multiple models approach can make classifications based upon combining the classifications of both models. Having thus argued for the general multiple models approach, next we examine different ways of generating multiple models and ways of combining evidence and classifications from those models to make an overall classification.

7.2 The multiple models approach

Recent years have seen much work in learning multiple models for the purpose of reducing classification error.[1] Studies involving the use of multiple models

[1] Some examples are:
Decision trees: Kwok & Carter (1990); Buntine (1990); Oliver & Hand (1995); Breiman (1994).
Rules: Kononenko & Kovacic (1992); Kovacic (1994); Smyth *et al.* (1990).

Table 7.3: Even arbitrary lookahead can only produce a single model. The model that appears second best on the training set may outperform the apparently best model when compared on the test set.

Rule	Coverage	Info. gain
$Class_1(X,Y) \leftarrow a(X,Y), c(X), d(X).$	(60,0)	44.2
$Class_1(X,Y) \leftarrow a(X,Y), d(X), f(X).$	(59,1)	42.1

typically learn a set of models from one set of training examples. This *ensemble* makes classifications by combining the classifications of its models or by combining evidence from its models. The error rate of this ensemble, called the *ensemble error rate*, is usually compared to the error rate of the single model produced by using a single-model learning algorithm on the same training data.

Research in multiple models falls in to two broad categories: in the first, called *meta-learning* (Wolpert, 1992; Schaffer, 1994) each model is of a different form (decision tree, rule, ...). This is also called the *heterogeneous multiple models* approach. In the second category, several models in the same hypothesis space are constructed from the same training examples. In this approach, the form of the model has been chosen by the user ahead of time and the program simply learns a set of models of that form. This approach is called the *homogeneous multiple models* approach. In this thesis we will be concerned with this kind of approach.

The following methods have been used in previous research to generate multiple models from one training set; these are listed in approximate chronological order:

1. **Boosting** (Schapire, 1990): Boosting is used for incremental or on-line learning - in this framework, one can think of a stream of examples from which training examples are sampled. The first model is constructed on a training set of size N fixed by the user. Next, examples are sampled from the stream and the first model is tested on these examples. The first $N/2$ examples that are correctly classified by the model are retained and the first $N/2$ examples incorrectly classified are retained. A second training set of size N is thus constructed. A model is then learned on this training set. This process can continue indefintely to learn many models but a very large number of examples is required to be available on-line.
2. **Partition sampling** (Gams (1989); Ali & Pazzani, (submitted)): The training set is divided into N equal-sized *folds* and on the i-th iteration, all but the i-th fold is used for training. Thus, N training sets are produced and learning is performed on each training set. This then produces N models. N is a parameter set by the user.
3. **User-intervention** (Kwok & Carter (1990)): The user is presented with a list of the best candidates (decision tree splits) ranked by information gain. Each time the user is presented with the same list, s/he can pick a different candidate. After the user has picked a candidate, learning continues in an automated fashion to complete the decision tree.

Rule sets: Gams (1989); Ali & Pazzani (1995b).
Neural networks: Hansen & Salamon (1990); Perrone & Cooper (1993); and many others.
Bayesian networks: Madigan & York (1993).
Regression: Perrone & Cooper (1993); Breiman (in press).

4. **Beam search** (Buntine (1990); Smyth *et al.* (1990); Oliver & Hand (1995)): Beam search is used to collect all candidates whose gain is within some fraction of the gain of the best candidate. Beam search is used to expand these best options and the process is recursively repeated until each beam element produces a complete model.
5. **All pruned models** (Buntine (1990)): A single model (decision tree in Buntine's case) is learned and then is augmented by all bottom-up pruned sub-models of that model.
6. **Random initialization** (Hansen & Salamon, (1990); Kononenko & Kovacic (1992); Perrone & Cooper, (1993)): This method has mainly been applied to neural networks although Kononenko & Kovacic (1992) have used it to initialize models consisting of single rules. Each neural network is initialized with a slightly different set of weights and then learning proceeds deterministically from that point on. In the work of Kononenko & Kovacic, a random attribute-value pair is used to initialize each rule. Greedy search is then used to complete the rule.
7. **Stochastic search** (Ripley, 1987; Kononenko & Kovacic, 1992; Ali & Pazzani, 1995c; Ali & Pazzani, 1995b) Instead of always preferring the candidate that maximizes the gain metric, this method stochastically chooses from a bucket of candidates. The bucket either comprises the top N candidates, or consists of all candidates whose gain was within some fraction β of that of the top candidate.
8. **Bootstrap sampling** (Breiman (1994); Ali & Pazzani (submitted)): N training sets are constructed by sampling *with replacement* from the original training set; N is a user-supplied parameter. N models, one per training set, are constructed. Since examples are being sampled with replacement, it often happens that the same example may appear many times in a training set. This method is also called *Bagging* (Breiman, 1994).

Having learned several models, the next question is how to combine the models' classifications in order to produce an overall classification by the ensemble. One way is to combine the models' classifications. A more powerful scheme is to allow each model to output a vector of "degrees of belief" together with its classification. The i-th component in the vector denotes the degree to which the model "believes" that the current test example should be assigned to the i-th class. This vector encodes the *evidence* from the i-th model. Then for each class $Class_i$, the degrees of belief for $Class_i$ are combined to yield an overall degree of belief for $Class_i$. Finally, the output of the ensemble classification procedure is the class label which has the highest degree of belief.

Some combination functions that have been used are:

1. **Uniform Voting** (Kwok & Carter (1990); Breiman (in press); and others): In this scheme, each model only outputs a class label. The class that is

predicted by the greatest number of models becomes the prediction of the ensemble. This method is also called *Simple Voting* or *Plurality*.

2. **Naive Bayesian Combination** (Kononenko & Kovacic, 1992): In this scheme, the following equation is used to compute the posterior probability of class C:
$$p(C|M_1,\ldots,M_k) = P(C) \times \prod_{i=1}^{k} \frac{P(C|M_i)}{P(C)}$$
Here, $p(C)$ denotes the prior probability of class C, and $\{M_i\}$ denotes the set of learned models. If some model M_i was not satisfied by the test example, 1 is used in the product instead of $\frac{p(C|M_i)}{p(C)}$ for that model. The output of the ensemble is the class with the maximum posterior probability $p(C|M_1,\ldots,M_k)$.

3. **Likelihood Combination** (Ali & Pazzani, (1995b)): This method multiplies the prior odds of a class with a product of the LS's of the representative rules of all the class descriptions of that class. The class that maximizes this overall product is output as the classification of the ensemble. As Likelihood Combination is the default evidence combination method used in HYDRA, it is described in detail in the following section.

4. **Bayesian Combination** (Buntine, (Buntine, 1990); Ali & Pazzani (1995c), Oliver & Hand, (1995)): In Bayesian Combination, the voting weight of a model is proportional to its posterior probability given the training data. Section 7.4 gives details of this method. Briefly, the posterior probability of a model is higher if the model fits the training data better. It is also higher if the model is syntactically simple. The output of the ensemble is the class that has highest expected posterior probability.

5. **Distribution Summation** (Ali & Pazzani, submitted; Clark & Boswell, 1991) This method is also explained in detail in Section 7.4; only an outline is given here. Roughly speaking, this method sums up the training coverages of satisfied rules. The class with the highest component in the resulting summed training coverage vector is used as the classification of the ensemble.

6. **(Un)Weighted average** (Perrone & Cooper, 1993; Breiman, in press) Perrone & Cooper apply the multiple models method to regression. They build an ensemble of regression models and then use the Lagrange multiplier method to find the best (with respect to a validation set) set of weights for those models. Breiman uses a simple method in which a straight average of the values predicted by the regression models is taken. Note that for regression, the output of each model is a continuous value denoting not the amount of evidence, but the predicted value. Perrone & Cooper also apply the Lagrange method to find an optimal set of weights with which to combine the activation levels of neural networks. Thus, they use weighted averaging for classification and regression.

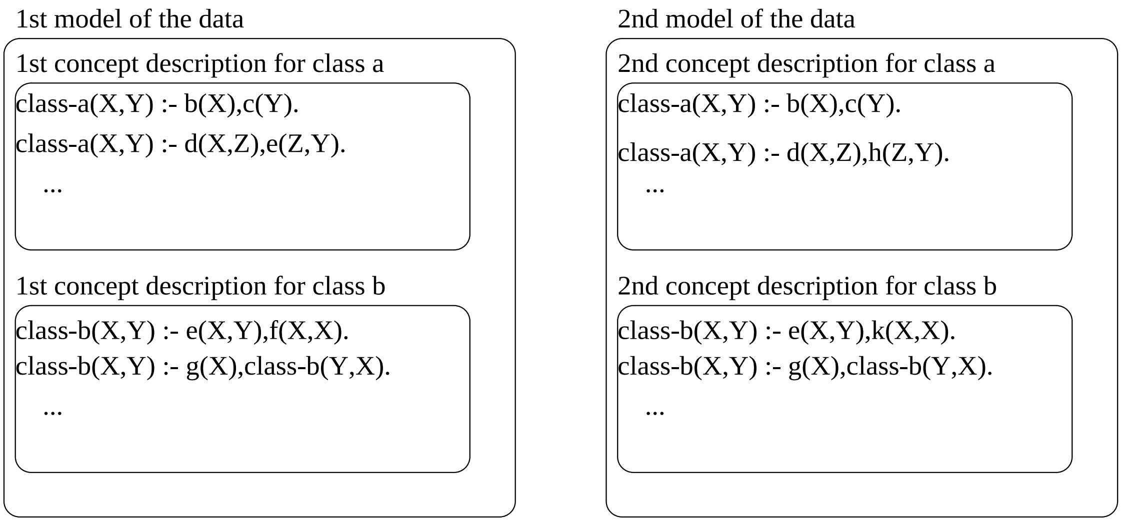


Figure 7.1: HYDRA-MM learns several models thus it also learns several descriptions for each class.

7.3 Multiple models using HYDRA

HYDRA produces a single model which consists of a set of class descriptions. Each class description consists of a set of rules for that class. In this section, we present HYDRA-MM which learns several models as shown in Figure 7.1.

Classification - HYDRA-MM combines evidence from models using the method of Likelihood Combination. Let $M_1, ..., M_k$ denote the ensemble of models that have been learned. Then, the posterior odds of class C_i for the test example x given the ensemble are given by the odds form of Bayes rule (Duda et al., 1979):

$$O(C_i|M_1, \ldots, M_k) \propto O(C_i) \times O(M_1, \ldots, M_k|C_i) \tag{7.1}$$

Using independence assumptions we can transform this into

$$O(C_i|M_1, \ldots, M_k) \propto O(C_i) \times \prod_j O(M_j|C_i) \tag{7.2}$$

where $O(C_i)$ denotes the prior odds of class C_i as in Equation 4.4. For the remaining terms, $\{O(M_j|C_i)\}$, the LS of the representative rule of class C_i, $r(M_j(C_i), x)$, in model M_j for x is used. $M_j(C_i)$ denotes the class description for class C_i in model M_j. If there is no rule in $M_j(C_i)$ satisfied by x, a value of 1 is used for $O(M_j|C_i)$. In summary then, the posterior odds of a class are calculated by multiplying the prior odds of that class with a product of LS's of representative rules. The class that maximizes this equation is output as the class predicted by the ensemble.

Learning - By default, HYDRA-MM uses stochastic hill-climbing to generate multiple models from one training set (however because experiments were conducted comparing various model generation methods, all those methods are now incorporated into HYDRA-MM). Stochastic hill-climbing only involves modifying HYDRA's procedure for selecting which literal to add to the clause currently being learned. Instead of picking the best literal (ranked according to some measure such as information gain) stochastic hill-climbing stores all literals whose gain is within some fraction, β, of the best (we will refer to this set as the "β-bucket") and then picks non-deterministically from among that set. The probability of a literal being picked is proportional to its gain. For the experiments described in this thesis, 0.8 was used as the value for β.

7.3.1 Comparison to single-model HYDRA

In this section we present results of an experiment designed to answer the following question:

> What effect does the use of multiple models have on classification error as compared to the error produced by learning a single model?

For this experiment eleven models were chosen. We chose a odd number to prevent ties from occurring for the Uniform Voting combination method for two-class domains (of which there are many). Although most of the results in the remainder of the thesis are given for eleven models, we also performed experiments using one, two and five models. Figure 7.2 shows the effect of varying the number of models on classification accuracy. For most of the data sets used in this study, increasing the number of models beyond eleven does not yield significant increases in accuracy.

Table 7.4 compares results using eleven stochastically-generated models combined using Likelihood Combination to that of a single model using LS weights and data-derived (non-uniform) priors. Information gain was used as the gain metric. In the table, the first column lists the data set identifier. The second column indicates the accuracy that would be attained by guessing the most frequent class in the training set. An asterisk signifies that the accuracy of the single model method was not significantly better than guessing the most frequent class. The third column indicates the accuracy obtained by using HYDRA (using information gain) and the fourth column indicates the accuracies obtained by HYDRA-MM. The next column indicates the ratio ($E_r = E_e/E_s$) between the error obtained by learning multiple models (E_e) over the error obtained by learning a single model (E_s). A "+" indicates a significant (using the paired 2-tailed t-test at the 95% confidence level) reduction in error, a "–" indicates a significant increase. An asterisk indicates that the accuracy of the single model version was not significantly better than guessing the most frequent class. If A_s denotes the accuracy of the single model version and A_e denotes the accuracy of the ensemble, the error ratio

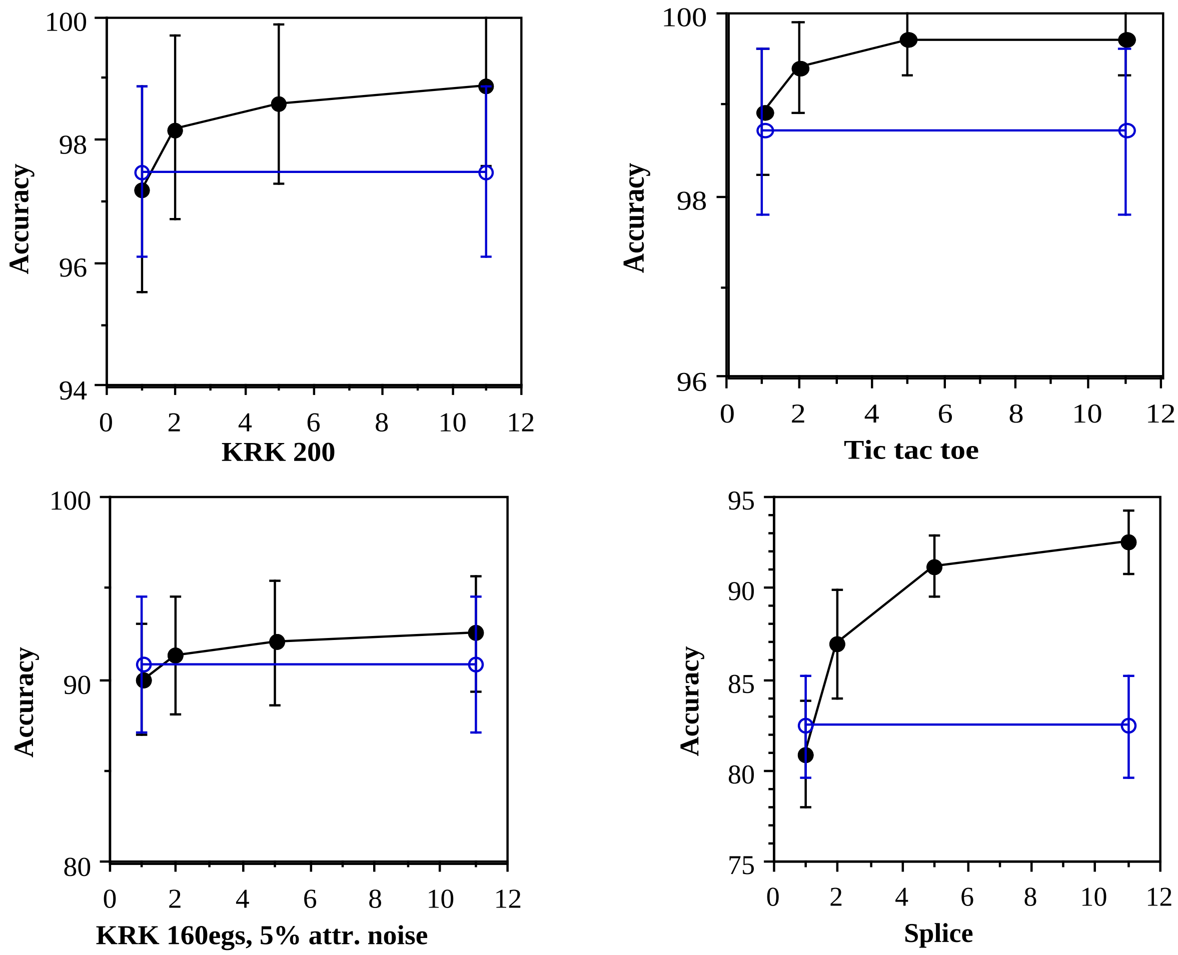


Figure 7.2: The figures above illustrate the effect of varying the number of stochastically-learned models (combined with Uniform-Voting). Open circles represent accuracies obtained by HYDRA, closed circles represent the multiple models method, HYDRA-MM. Standard-deviation bars show 1 standard deviation. At 11 models, HYDRA-MM is significantly more accurate than HYDRA on all but the KRK 160e, 5a data set. At least one data set from each of the major types of data sets (artificial concepts, artificial concepts with added noise, Molecular-biology, other "real-world") used in this thesis is represented.

Table 7.4: Comparison of accuracies produced by a single model versus accuracies produced by an ensemble of eleven stochastically-learned models (combined with Likelihood Combination).

Domain	Default Accuracy	Single Model Accuracy	11 Stochastic Models Accuracy	11 Stochastic Models Error Ratio	Number of training examples
Led 8i	10.0%	87.2%	+ 96.4%	+ .28	30
Led 17i	10.0%	83.7%	+ 94.6%	+ .33	30
Tic-tac-toe	65.3%	99.0%	+ 99.8%	+ .22	670
Krkp	52.0%	94.5%	+ 95.5%	+ .82	200
Krk 100e	66.7%	95.1%	95.6%	.90	100
Krk 200e	66.7%	98.3%	98.9%	.66	200
Krk 160e 5a	66.7%	91.9%	93.2%	.84	160
Krk 320e 5a	66.7%	94.8%	95.8%	.80	320
Krk 160e 20c	66.7%	89.6%	91.1%	.86	160
Krk 320e 20c	66.7%	92.5%	93.4%	.88	320
Led 20a	10.0%	94.3%	94.6%	.94	50
Led 40a	10.0%	85.0%	87.7%	.82	50
DNA	50.0%	67.9%	+ 86.8%	+ .41	105
Splice	53.4%	85.3%	+ 92.5%	+ .51	200
Mushroom	50.0%	97.4%	96.8%	1.24	100
Hypothyroid	90.0%	95.3%	+ 97.8%	+ .47	200
BC-Wisconsin	65.5%	93.5%	+ 96.1%	+ .60	200
Voting	62.0%	93.5%	+ 94.9%	+ .78	100
Wine	39.8%	93.3%	+ 98.9%	+ .16	118
Iris	33.3%	91.4%	92.4%	.88	50
Soybean	14.6%	88.5%	+ 92.3%	+ .67	288
Horse-colic	63.4%	83.2%	+ 87.1%	+ .77	245
Hepatitis	*79.6%	78.8%	79.2%	.98	103
Lymph.	54.7%	77.9%	+ 83.9%	+ .73	110
Audiology	25.3%	72.1%	+ 80.5%	+ .70	150
Diabetes	65.1%	71.9%	+ 73.9%	+ .93	200
B.Cancer	*70.2%	69.9%	– 67.2%	– 1.09	190
Heart	*54.1%	54.3%	55.2%	.98	200
Primary-tumor	24.7%	38.8%	38.2%	1.01	225
Average	–	84.8	87.9	0.80	–

is equal to $\frac{100-A_e}{100-A_s}$. For the DNA domain, the t-test is not applicable because we used leave-one-out testing. For this domain, we used a sign-test (DeGroot, 1986).

Table 7.4 shows that stochastic search using Likelihood Combination is able to statistically significantly (95% confidence) reduce or maintain error on all domains except the (Ljubljana) breast-cancer domain.[2] On that breast cancer data set few learning methods have been able to get an accuracy significantly higher than that obtained by guessing the most frequent class suggesting it lacks the attributes relevant for discriminating the classes. The table shows that for approximately half the data sets, error is reduced by a statistically significant margin when using models learned by stochastic search and combined with Likelihood Combination. Tables 7.8 and 7.9 shows that the other evidence combination methods and learning methods also lead to statistically significant error reductions for a large fraction of the data sets. There is no significant change in error for most of the other data sets - on very few occasions does the multiple models approach lead to a significant increase in error.

In Table 7.8 accuracies for all four evidence combination methods for the single model algorithm and all three multiple model generation methods are presented. Within each model-generation method, evidence-combination algorithms are in the following order: Uniform Voting (UV), Bayesian Combination (BC), Distribution Summation (DS) and Likelihood Combination (LC). A "+" indicates a statistically significant improvement in accuracy over the corresponding single-model algorithm run with the same evidence-combination method. So, for instance, the accuracy of 97.0% for LED8i (Stochastic, Uniform Voting) indicates that it was statistically significantly better than the single model algorithm using Uniform Voting. A "-" indicates a significant decrease in accuracy. At the end of each group of data sets is a row of ranks. The rank is an average for that learning/evidence-combination algorithms for that group of data sets. So, for instance, the table shows that on the noise-free artificial concepts, the best model-generation/evidence-combination-algorithm combination was Stochastic and Likelihood Combination since it had the lowest average rank over that group of data sets.

Another striking aspect of the results presented in Table 7.4 is that the error is reduced by a factor of 6 for the wine data set (representing an increase in accuracy from 93.3% to 98.9%!) and by factors of 3 or 4 for LED and Tic-tac-toe. Accuracy improvements such as the improvement from 93.3% to 98.9% are rare in Machine Learning in which most learning algorithms typically differ by a few (typically less than 3% or 4%) percentage points in accuracy on most domains.

[2]Note that even though the error ratio for Mushroom (1.24) is worse than it is for Ljubljana Breast Cancer (1.09), it does not correspond to a statistically significant decrease in accuracy because the t-test only deems two algorithms to be different if their differences are much greater than the within-algorithm standard deviations. The within-algorithm standard deviation of accuracy for the multiple models algorithm on Mushroom was quite large so the t-test did not find a significant difference on Mushroom even though the error ratio is quite large.

The molecular biology data sets also experienced significant reduction with the error being halved (for DNA this represented an increase in accuracy from 67.9% to 86.8%). The error reduction is smallest for the noisy KRK and LED data sets and for the (presumably noisy) medical diagnosis data sets. Most of the data sets which scored relatively poor error ratios (above 0.8) were noisy data sets. This finding is further explored in Chapter 8 in which the effect of class noise on error ratios is explored. The fact that the best error ratios were obtained on the noise-free and molecular biology data sets holds for all four of the evidence combinations schemes used and three model generation methods (see Sections 7.4 and 7.5).

The LED domain, in particular, gives us some insight into the effect of irrelevant attributes and class noise on error ratios. As the table shows, learning multiple models helps a lot in reducing errors of the LED data sets with irrelevant attributes and no noise. For eight irrelevant attributes, the error is reduced from 12.8% to just 3.6%. This suggests that when irrelevant attributes are present, using multiple models provides a substantial benefit. Backing up this hypothesis are the DNA and Splice domains which have many attributes, many of which are probably irrelevant. Impressive (low) error ratios were obtained on these domains. These observations led us to more carefully investigate the effect of irrelevant attributes on error ratio. The results of those investigations are presented in Chapter 8.

Although error ratios for the noisy data sets represent a statistically significant reduction in error, the reductions are not as impressive as they are for noise-free domains containing irrelevant attributes. Again, the LED data sets provide some insight since the two types of LED data sets only vary in that one type has irrelevant attributes and no noise, and the other has noise and no irrelevant attributes. All other factors are unchanged. The LED results suggest that noise is the main factor preventing the multiple models method from obtaining impressive error ratios. This issue is explored in detail in Chapter 8.

In summary, the answer to the question for this section - "What effect does the use of multiple models have on classification error?" - is that the use of multiple models leads to significant reductions in classification error for about half of the data sets tested here. For most of the other data sets, the error does not change significantly. It only increases significantly on one of the 29 data sets.

7.4 Comparison of combination methods

This section presents a comparison of evidence combination methods and identifies the domain characteristics under which each method is superior. Our experiments compare four evidence combination methods: Uniform Voting, weighted combination according to Bayesian probability theory (Bayesian Combination;

1st model of data (posterior prob. = 0.02)

1st description of class a

a(X,Y) :- j(X), g(Y). Covers (7,0) LS = 5.0, Accu. = 0.89

a(X,Y) :- c(X,Z), i(Y). Covers (4,1), LS = 1.6, Accu. = 0.71

a(X,Y) :- k(Y,X), d(X). Covers (3,1), LS = 1.3, Accu. = 0.67

1st description of class b

b(X,Y) :- h(X,Y), f(Y). Covers (6,0), LS=11.2, Accu. = 0.88

b(X,Y) :- n(X,Y), f(Y). Covers (2,1), LS=2.4, Accu. = 0.60

2nd model of data (posterior prob. = 0.015)

2nd description of class a

a(X,Y) :- j(X), d(Y). Covers (8,0). LS=5.6, Accu. = 0.90

a(X,Y) :- k(Y,X), i(Y). Covers (6,0), LS=4.4, Accu. = 0.88

2nd description of class b

b(X,Y) :- m(X,Y), f(Y). Covers (6,1), LS=5.2, Accu. = 0.78

b(X,Y) :- n(X,Y), i(Y). Covers (2,0), LS=4.8, Accu=0.75

Figure 7.3: Comparison of evidence combination methods. "Covers (6,1)" for a rule for class 'b' indicates that the body of that rule is true for 6 training examples of class 'b' and for 1 training example of other class(es). The given accuracy is a Laplace estimate of the accuracy of the rule as estimated from the training data.

Buntine (1990)), weighted combination according to Distribution Summation (Clark & Boswell, 1991) and Likelihood Combination (Duda et al., 1979). Results using four evidence combination methods compared in this section and three model generation methods compared in Section 7.5 are listed in Table 7.8.

Figure 7.3 illustrates the task for evidence combination. The rules in bold typeface indicate the rules that have been satisfied for the current test example. The figure indicates that the test example satisfies the preconditions for two rules of class `a` in the first description for class `a`. The first of these rules covers four training examples of class `a` and a training example of the other class, class `b`. The figure shows that each evidence combination method requires its own type of reliability estimate. Bayesian Combination, for instance, requires an estimate of the accuracy of the rule to be attached to each learned rule. In the figure, the accuracy of each rule is calculated by using a Laplace estimate of the accuracy over the training data. Although the figure shows three kinds of reliability measure associated with each rule, each evidence combination method only uses one kind of reliability measure.

Table 7.5: Degrees of belief for four evidence combination methods. The composite degree of belief for a class is obtained by summing the degrees of belief for that class over descriptions of that class.

Evidence comb. combination method	Descr. 1 class a	Descr. 2 class a	Composite class a	Descr. 1 class b	Descr. 2 class b	Composite class b
Uniform	1	1	2	0	1	1
Bayesian	0.02 * 0.71 = 0.0142	0.015 * 0.90 = 0.0135	0.0277	0.02 * 0 = 0	0.015 * 0.78 = 0.0117	0.0117
Distribution	(4,1) + (3,1) = (7,2)	(8,0)	(15,2)	(0,0)	(6,1)	(6,1)
Likelihood	1.6	5.6	$1.6 \times 5.6 \times 1.75$ = 15.68	1	5.2	$1 \times 5.2 \times 0.57$ = 2.96

When a single model has been learned, evidence combination proceeds as follows.

- **Uniform Voting:** If some rule of a class description is satisfied, that class gets a score of 1; otherwise it gets a score of 0. Since Uniform Voting uses the "single, most reliable rule" bias, satisfaction of additional rules is not taken as greater evidence for the class. See Brazdil & Torgo (1990) for empirical support for this type of bias. If there is more than one class that ends up with a score of 1, a random choice is made.
- **Bayesian Combination:** If some rule of a class description is satisfied, the Laplace accuracy of that rule is used as the score for that class. Bayesian Combination also uses the "single, most reliable rule" bias so it only looks at the satisfied rule with the highest accuracy. Since only one model is being used, the posterior probability of the model does not influence classification - the score of the class is simply the Laplace accuracy of the most reliable satisfied rule.
- **Distribution Summation:** This is the only evidence combination method that does not use the most-reliable rule bias. It sums up the distributions of all satisfied rules to produce a composite distribution. The class with the greatest component in that distribution is the winner.
- **Likelihood Combination:** For each class, its score is computed by multiplying the prior odds of that class by the LS of the most reliable satisfied rule (or 1.0 if there is no satisfied rule).

When more than one model has been learned, for each class, the evidence combination method proceeds as follows. It calculates a degree of belief for that class from each of the descriptions for that class. Then, these degrees of belief are combined to yield the overall degree of belief that the test example belongs to the current class. This process is repeated for each class. The class with the highest overall degree of belief is output as the class predicted by the ensemble.

The four evidence combination methods compared in this section are:

- Uniform Voting - Each description votes with an equal weight to classify new examples. For example, in Figure 7.3, class `a` receives two votes since both descriptions of class `a` have at least one rule matching the test example. As class `b` only receives one vote, this evidence combination method assigns the test example to class a.

- Bayesian Combination (Buntine, 1990) - In Bayesian Combination, there are weights associated with models (the posterior probability of the model) and weights associated with rules (the accuracy of the rule). In the general form of Bayesian Combination, the test example, x, should be assigned to the class, c, with the maximal expected posterior probability:

$$E(p(c|x, \mathcal{T}, \vec{x}, \vec{c})) = \sum_{t \in \mathcal{T}} p(c|x, t)p(t|\vec{x}, \vec{c}) \tag{7.3}$$

where $\mathcal{T}$ is the model (hypothesis) space of all possible models, t is an arbitrary model in that space, $\vec{x}$ denotes the training examples and $\vec{c}$ denotes the class labels of those training examples. $p(c|x, T)$ is the probability of class c given a test example and a particular learned model t. $p(c|x, T)$ can be thought of as the degree to which t "endorses" class c for example x. Since we are using a "single, most reliable rule" bias, we use the Laplace estimate of the training accuracy of the representative rule as an estimate for $p(c|x, T)$. $p(T|\vec{x}, \vec{c})$ denotes the posterior probability of the model. See (Ali & Pazzani, 1995c) for details on how Buntine's (1990) form for the posterior probability of a decision tree is adapted to compute the probability for a model consisting of a class description for each class. Briefly, models whose class descriptions are syntactically-compact and are well able to separate the training examples of differently classes end up with higher posterior probabilities.

The general Bayesian method is used in our "single, most reliable rule" framework as follows. As Figure 7.3 indicates, the first model has posterior probability 0.02 and the representative rule for the given test example and class description for class `a` has accuracy 0.71. The Laplace accuracy of the representative rule in the second description for class `a` is 0.90. This gives a degree of belief for class `a` of 0.0277 ($0.02 * 0.71 + 0.015 * 0.90$). Doing the same for class `b` yields a degree of belief of 0.0117 for class `b`. Since class `a` has a higher expected posterior probability, the test example is assigned to class `a`.

- Distribution Summation (Clark & Boswell, 1991) - This method associates a k-component vector (the distribution) with each rule. The vector consists of the numbers of training examples from all k classes covered by that rule.[3] A component-wise sum is formed over all satisfied rules of all class descriptions

[3]Note that this method differs from other evidence combination methods in domains where the number of classes is greater than two because in such domains, "negative examples" can

of all classes to produce a combined distribution. This method is the only one of the four evidence combination methods compared in this section that uses more than one satisfied rule per class description (hence, it does not use the "single, most reliable" bias). So, in Figure 7.3, the distributions associated with the satisfied rules (of all classes) are added together to yield the combined distribution: $(21, 3)$ which is the sum of all the distributions of the satisfied rules: $(21,3) = (4,1) + (3,1) + (8,0) + (6,1)$ The class corresponding to the highest value in the combined distribution is chosen as the winning class. In this case, the highest value is attained by class `a` so the test example is assigned to class `a`.

- Likelihood combination - This method is described in Section 7.3. Here we use Table 7.5 to illustrate its usage. In Figure 7.3, 14 of the 22 training examples are from class `a` so class `a` has a "prior" probability of 0.63 and prior odds of 1.75. Class `b` has 8 of the 22 examples for a "prior" probability of 0.36 and prior odds of 0.57. As per Equation 4.5 the posterior odds of class `a` are obtained by multiplying the prior odds of class `a` by the LS of the most reliable matching rule in the first description of class `a` and by the LS of the most reliable matching rule in the second description of class `a`. This yields posterior odds of 15.68 for class `a` and posterior odds of 2.96 for class `b`. Therefore, this evidence combination method will assign the example to class a.

A few words on why these evidence combination methods were chosen for the comparison. We chose Uniform Voting as a "straw man" method which the other methods should be able to beat in terms of accuracy. We chose Bayesian Combination because it is an approximation to the optimal Bayes approach. In particular, if some other method did better than this method that would imply that quantities in that other method could be more reliably estimated from training data rather than because of some fundamental theoretical flaw in the Bayesian approach. Distribution Summation was chosen since it can utilize more than one satisfied rule per class description. This encodes a bias which may be suitable for a certain class of domains. We are interested in characterizing that class. Finally, we chose Likelihood Combination because the logical sufficiency measure has the flavor of measuring both coverage and accuracy.

Now we present comparative results for the four evidence combination methods on 29 data sets from 21 domains. In particular, we investigate the hypothesis that the Distribution Summation combination method should do relatively poorly on domains where one or more of the classes are near-disjoint DNF concepts. Examples of such concepts in our suite are KRK and Tic-Tac-Toe. It is our hypothesis that the Distribution Summation method should do relatively poorly on

be from two different classes: the Distribution Summation method differentiates these classes whereas the other evidence combination methods lump examples of all such classes together as "negative examples."

such domains because in a situation where more than one rule of a class is satisfied by a test example, the Distribution Summation method looks at all such satisfied rules. For near-disjoint DNF classes, all test examples should satisfy exactly one rule each. Satisfaction of more than one rule means the class definition was poorly learned and hence the rules are probably more error-prone. Hence, it is our hypothesis that the Distribution Summation method encodes a bias that is unsuitable for near-disjoint DNF classes and so should do relatively poorly on domains containing such classes.

To compare the four combination methods, we compared their accuracies (for eleven stochastically generated models learned using information gain) using a paired 2-tailed t-test at the 95% confidence level. Table 7.6 illustrates differences in behavior among the four evidence combination methods. Uniform priors were used for the Bayes method.

To read the table, an entry such as -sum in the column comparing "Sum" (Distribution Summation) to Bayes indicates that the Distribution Summation method was significantly weaker than Bayes. To determine whether to enter "-Sum" or "+Bayes" for such a comparison, both the Distribution Summation and Bayes methods are compared to all the evidence combination methods. If, for example, the accuracy for Bayes is in line with the other methods but Distribution Summation is weaker than the other methods, then we attribute the significant difference between Bayes and Distribution Summation to the weakness of Distribution Summation rather than to the strength of Bayes. Blank entries indicate there was no significant difference between the methods. The table indicates that typically, when significant differences arise, they arise between a method and all the others. So, for instance, the table shows that the Distribution Summation method is significantly stronger than all other methods on the Primary Tumor problem. For Diabetes, significant differences in accuracy are attributable to two effects: Bayes is significantly or numerically weaker than the other methods and Distribution Summation is significantly or numerically stronger than the other methods.

The main point to draw from this table is that the Distribution Summation method is more different to the other methods than they are from each other. We hypothesize that this is because the Distribution Summation method does not use the "single, most reliable rule" bias. The table indicates that not using this bias is an advantageous strategy for very noisy domains. On the other hand, the table shows that the Distribution Summation method is at a disadvantage for near-disjoint DNF concepts (such as Tic-Tac-Toe) and the less noisy natural data sets such as Mushroom and Breast Cancer (Wisconsin). On the Tic-Tac-Toe domain with eleven stochastically learned models, the LS method used 0.998 rules (on average) for test examples from the first class and 0.977 rules for test examples from the other class (values less than 1 indicate that some test examples did not satisfy any rules - in such cases, HYDRA has to guess the most frequent class). The

Table 7.6: Comparison of evidence combination methods using eleven stochastically generated models.

Domain	Unif vs. Bayes	Unif vs. Sum	Unif vs LS	Bayes vs. Sum	Bayes vs. LS	Sum vs. LS
LED8i						
LED17i						
Tic-tac-toe				-SUM		
KRKP						
KRK100e						
KRK200e						
KRK160e,5a						
KRK320e,5a						
KRK160e,20c						
KRK320e,20c						
LED20a						
LED40a						
DNA						
Splice	-BAYES			-BAYES	-BAYES	
Mushroom		-SUM		-SUM		
Hypothyroid					+LS	+LS
B.Cancer (Wisc.)		-SUM		-SUM		-SUM
Voting						
Wine						
Iris						
Soybean		-SUM	+LS		+LS	+LS
Horse-colic				-BAYES	-BAYES	
Hepatitis	+UNIF					
Lymphography						
Audiology						
Diabetes		+SUM		+SUM		+SUM
				-BAYES	-BAYES	
B.Cancer						-LS
Heart		+SUM		+SUM		+SUM
Primary-tumor		+SUM		+SUM		+SUM

Distribution Summation method used, on average, 1.51 rules for test examples of the first class and 1.19 rules for the other class. We hypothesize that Distribution Summation does poorly on Tic-Tac-Toe simply because it can consider more than one satisfied rule per description. Similar trends hold for the KRK domain: the Distribution Summation method was inferior at the 90% confidence level. The Distribution Summation method strengthens the belief in a class when more than one rule from the description of that class matches the test example. But when learning from a large number of examples from a nearly disjoint DNF concept, the extensions of the learned clauses are nearly disjoint so the rules should be learned such that test examples do not match more than one example in a class description. By considering additional satisfied rules, the Distribution Summation method over-strengthens the degree of belief in that class. These results are reinforced by Table 7.8 which shows that the average rank for the Distribution Summation method on the noise-free artificial data sets is higher (i.e. worse) than the other methods under all three multiple-model generation methods.

Both tables (7.6 and 7.8) also indicate weakness in the Bayesian Combination method. We hypothesize that this occurs when one of the eleven models has a much higher probability (voting weight) than any of the other models. As a consequence, the classifications made by that model greatly dominate those of the others and the advantages of evidence combination are lost. We have noticed that the posterior probabilities associated by the Bayes method with models are quite sensitive to the syntactic size of the model. Less complex models are assigned much higher posterior probabilities. It is not clear whether this is an implementation problem or arises because we are using such a small number of models. To be in accord with Bayesian theory we should be using all the models in the model space.

Table 7.6 also indicates, somewhat surprisingly, that the Uniform and LS only have a significant difference on one data set. These are also the methods that are less variable from domain to domain. However, the average rank of LS over all 29 data sets (Table 7.8) at 5.0 is better than that of Uniform Voting at 6.1 suggesting that overall LS is better than the simple Uniform Voting method.

Table 7.8 also shows that the Uniform Voting method compares quite well to the other methods suggesting that the more complex methods that depend on rule-reliability estimates are not making particularly good estimates. This in turn suggests the need for some kind of hold-back set in order to make better estimates. The Uniform Voting method gets the second best rank overall beating out Distribution Summation and Bayesian Combination.

The differences between the evidence combination methods can be summarized as follows. For all model-generation methods, LS had a lower rank than the other evidence-combination methods (Table 7.8) suggesting it is best to use overall. Surprisingly, Uniform Voting came second and did better than the other two more “sophisticated” combination methods. These two methods also varied less in their performance than the other two methods. Distribution Summation did relatively poorly on near-disjoint DNF concepts but was best on very noisy natural data sets. The Bayes method did poorly on domains where the posterior probability of one model dominates those of the other models and overall it did not do well.

7.5 Comparison of model-generation methods

Table 7.7 compares three methods for generating multiple models. The entries are for eleven descriptions combined using the best overall evidence combination method - the LS method. Results for all three generation methods crossed with four evidence combination methods are presented in Table 7.8. The table shows that Partition is clearly inferior to the other two model-generation methods (this is corroborated by Table 7.8). There is no data set for which Partition is significantly better than the other two methods.

Table 7.7: Comparison of description generation methods (all using the LS method).

Domain	Stochastic vs. Partition	Stochastic vs. Bagging	Partition vs. Bagging	Number training examples
LED8i		-BAG	-BAG	30
LED17i		-BAG	-BAG	30
Tic-tac-toe		-BAG		670
KRKP				200
KRK100e				100
KRK200e			+BAG	200
KRK160e,5a				160
KRK320e,5a		+BAG		320
KRK160e,20c				160
KRK320e,20c		+STOCH		320
LED20a		-BAG	-BAG	50
LED40a		-BAG	-BAG	50
DNA				105
Splice	-PARTITION		-PARTITION	200
Mushroom				100
Hypothyroid	-STOCH	-STOCH		200
B.Canc. (Wisc.)	+STOCH	+STOCH		200
Voting			+BAG	100
Wine	-PARTITION	+STOCH	-PARTITION	118
Iris			+BAG	50
Soybean	+STOCH			290
Horse-colic				245
Hepatitis				103
Lymph.	+STOCH	+STOCH		110
Audiology				145
Diabetes		+BAG	+BAG	200
B.Cancer		+BAG	+BAG	190
Heart		+BAG	+BAG	200
P. Tumor	+STOCH	+BAG	+BAG	225

Table 7.8: Accuracies for all evidence combination methods (next page). See Subsection 7.3.1 for a detailed explanation of the table.

Task	Single Model				Stochastic				k-fold Partition				Bagging			
	UV	BC	DS	LC	UV	BC	DS	LC	UV	BC	DS	LC	UV	BC	DS	LC
LED8i	85.2	91.7	91.0	89.2	+ 97.0	+ 97.9	+ 97.7	+ 98.0	+ 97.2	+ 97.8	+ 98.0	+ 97.9	+ 91.5	92.0	92.3	+ 92.4
LED17i	76.5	86.0	85.0	83.5	+ 95.4	+ 96.3	+ 96.2	+ 96.4	+ 94.9	+ 95.6	+ 95.1	+ 95.5	+ 87.0	87.2	+ 87.7	+ 87.5
TTT	98.7	99.0	98.9	99.0	+ 99.7	+ 99.8	+ 99.5	+ 99.8	+ 99.7	+ 99.8	+ 99.5	+ 99.6	+ 99.5	+ 99.6	99.0	+ 99.4
KRKP	92.5	94.5	94.5	94.5	+ 95.3	95.2	+ 95.4	+ 95.5	+ 94.9	95.0	94.9	+ 95.8	+ 95.1	94.5	94.9	+ 95.4
100	94.7	95.2	95.2	95.1	95.5	95.6	94.6	95.6	95.5	95.9	94.3	95.7	95.5	94.8	− 91.7	95.8
200	97.6	98.3	98.3	98.3	+ 98.8	98.9	98.2	98.9	− 93.7	− 94.3	− 93.7	− 94.7	+ 98.6	98.1	− 96.2	98.4
160.5	90.8	91.9	92.0	91.9	92.5	92.5	91.8	93.2	+ 93.0	92.4	92.3	93.5	+ 93.2	92.6	90.8	+ 94.5
320.5	93.4	94.8	94.8	94.8	+ 94.9	95.0	94.9	95.8	+ 95.8	94.9	95.4	+ 96.6	+ 95.9	95.0	95.0	+ 97.1
160.20	88.6	89.6	89.6	89.6	90.3	90.5	90.7	91.1	+ 90.9	90.2	+ 91.4	91.3	+ 91.3	88.8	90.8	+ 91.9
320.20	91.7	92.5	92.6	92.5	+ 95.5	+ 95.9	+ 94.3	+ 95.7	+ 93.5	92.8	93.6	+ 94.1	+ 93.9	92.9	+ 93.9	+ 94.3
LED20a	92.7	94.3	93.0	94.3	93.7	94.3	93.7	94.7	96.0	95.0	93.7	95.3	89.3	− 88.0	− 83.0	− 88.7
LED40a	81.0	85.7	82.0	85.0	84.7	87.0	85.3	87.7	+ 87.0	86.3	86.3	86.7	76.0	− 75.3	− 74.7	− 76.0
DNA	59.4	67.9	67.9	67.9	+ 86.8	+ 90.6	+ 87.7	+ 86.8	+ 84.0	+ 80.2	+ 85.8	+ 85.8	+ 87.7	+ 82.1	+ 89.6	+ 90.6
Splice	82.4	85.3	85.3	85.3	+ 92.5	+ 91.1	+ 92.3	+ 92.5	+ 91.0	+ 90.6	+ 91.0	+ 90.9	+ 92.3	+ 89.9	+ 92.0	+ 92.3
Mush.	97.4	97.5	97.5	97.5	98.0	97.6	96.3	97.3	98.1	97.4	− 95.5	97.5	97.6	− 94.5	− 94.5	96.4
Hypo.	97.6	97.6	97.6	95.2	− 95.8	− 94.9	− 95.4	+ 96.6	97.8	97.8	97.7	+ 97.5	97.9	97.8	− 96.9	+ 97.8
Voting	93.4	93.6	93.6	93.6	+ 94.6	+ 94.4	+ 94.9	+ 94.8	+ 94.4	94.1	+ 94.6	+ 94.5	+ 94.9	+ 94.7	+ 95.1	+ 95.1
BC.Wisc	92.5	93.5	93.6	93.5	+ 95.8	+ 95.5	+ 94.9	+ 96.1	+ 95.1	+ 95.1	93.7	+ 95.5	+ 95.5	+ 94.6	+ 94.6	+ 95.7
Wine	91.5	92.9	93.3	92.9	+ 98.9	+ 98.7	+ 98.9	+ 99.0	+ 97.0	+ 97.1	+ 97.5	+ 96.5	+ 98.4	+ 97.7	+ 98.4	+ 97.7
Iris	90.2	91.4	91.1	91.4	+ 92.8	92.6	92.2	92.4	90.8	91.5	90.1	91.7	+ 93.6	+ 93.5	+ 93.5	+ 93.9
Soybean	84.6	88.5	88.5	88.5	+ 91.6	+ 91.6	+ 91.0	+ 92.2	+ 90.8	+ 90.8	+ 90.3	+ 91.5	+ 91.3	+ 90.8	+ 89.4	+ 91.8
H.colic	82.3	83.2	83.3	83.2	+ 86.7	+ 86.0	+ 87.7	+ 87.0	+ 87.2	+ 86.1	+ 86.4	+ 87.8	+ 88.3	+ 87.2	+ 87.8	+ 88.0
Hep.	80.4	80.5	80.4	80.4	81.2	79.5	80.2	79.6	79.8	78.8	− 78.0	79.1	80.7	78.9	79.6	79.9
Lymph.	76.6	78.1	78.5	77.9	+ 83.8	+ 82.6	+ 82.4	+ 83.8	+ 78.9	80.4	80.3	80.1	+ 81.2	79.5	80.1	+ 80.9
Audio.	71.5	72.2	72.1	72.0	+ 80.5	+ 80.3	+ 78.5	+ 80.3	+ 78.3	+ 77.9	+ 77.0	+ 78.7	+ 78.8	+ 77.6	+ 76.5	+ 78.8
Diab.	70.4	71.9	72.1	71.9	+ 73.5	+ 72.9	+ 74.7	+ 73.9	+ 73.4	72.4	+ 74.5	+ 73.8	+ 74.6	+ 73.2	+ 75.2	+ 74.7
B.Canc	69.9	69.9	69.8	69.9	68.6	68.1	70.2	− 67.3	69.4	68.5	+ 72.0	68.5	+ 72.2	70.4	+ 72.4	70.5
Heart	54.2	54.3	54.2	54.3	56.0	55.7	+ 57.6	55.1	+ 56.3	54.9	+ 57.4	55.3	+ 57.8	+ 56.5	+ 58.2	+ 57.4
P.tumor	37.5	38.8	39.2	38.8	38.7	40.3	+ 42.8	38.3	37.3	39.1	41.0	− 36.1	+ 41.5	39.6	+ 44.3	40.6

Table 7.9: Table of rankings of various model-generation method and evidence combination method pairs on 29 data sets. Ranks are averages over the various groups of the 29 data sets.

Group	Single Model				Stochastic				k-fold Partition				Bagging			
	UV	BC	DS	LC	UV	BC	DS	LC	UV	BC	DS	LC	UV	BC	DS	LC
Noise-free artificial	14.7	11.2	12.0	12.2	5.5	3.3	7.3	2.0	8.7	5.5	9.7	5.3	8.8	11.0	11.3	6.7
Noisy artificial	16.0	12.7	12.7	12.7	8.0	6.2	9.5	3.5	6.5	10.5	6.7	3.2	4.7	10.2	9.2	1.7
Small sample noisy Artificial	12.0	6.0	11.0	7.0	9.0	3.5	8.0	2.5	1.5	4.0	7.0	3.0	13.0	15.0	16.0	13.5
Molecular Biology	16.0	13.5	13.0	13.5	4.0	4.0	3.5	3.5	9.5	11.5	8.0	9.0	4.0	11.5	4.5	3.0
Real-world	13.4	11.3	11.3	12.1	5.8	7.8	6.3	7.1	9.1	9.9	9.1	9.1	3.3	8.3	6.6	4.5
Overall	14.1	11.3	11.7	12.0	6.1	6.1	6.9	5.0	8.1	8.8	8.7	7.1	5.4	9.8	8.4	5.1

From the poor showing of Bagging on the LED data sets, one can infer that Bagging does poorly when there are few training examples (30 and 50 for the LED data sets). This is plausible since, on average, each training set constructed by bootstrap sampling leaves out about 37% of the original training examples - this is clearly a bad idea if there are only 30 training examples to begin with. Note however, that on these small sample-size LED data sets, Bagging still does better than the single-model approach. It is the combination of *noise with small-sample size* that causes a lot of problems for Bagging (LED20a and LED40a).

Since Partition does relatively poorly, the remainder of the discussion will focus on Stochastic and Bagging. Also, since LS generally provides more accurate classifications, the discussion will focus on the LS method. Using Table 7.8, one can see that on noisy data sets (artificial or "real"), Bagging does better than Stochastic. In fact, this is the main difference between the two methods. On artificial noisy data sets, for most evidence combination methods, the average rank of Bagging is less than that of Stochastic. Bagging is also better on the natural, noisy data sets (Bagging rank: 4.5, Stochastic rank: 7.1).

Stochastic does significantly better than Bagging on the Breast Cancer data set (Wisconsin) which is probably relatively noise-free since very high accuracies are obtained on that data set. However, for the more noisy data-sets (B.Cancer, Heart, Primary Tumor, Diabetes) Bagging does significantly better than Stochastic. In addition, on a number of data sets, Bagging does better than Stochastic but the difference fails to reach the 95% significance level. On Iris, for example, Bagging does better than Stochastic at the 90% confidence level. Bagging is better than Stochastic on the noisy data sets with the small-sample LED data sets being notable exceptions.

Another way to look at the results is to consider the number of data sets on which each method was not significantly different to the algorithm that achieved the highest accuracy on that data set. Table 7.10 presents these results. So, for example, looking at Tables 7.8 and 7.7 for the LED8i data set, we see that all multiple models methods were significantly better than the single model method but that Bagging was worse than the Stochastic and Partition. There was no significant difference between Partition and Stochastic. So, in Table 7.10 we assign a "x" for Stochastic and another for Partition. The table breaks up the "noisy artificial data sets" group into two groups: those with sufficient sample size (relatively many training examples) and those with few training examples: LED20a and LED40a. This helps bring out the difference between the multiple model methods.

The results of this section can be summarized as follows. Partition was clearly inferior to the other algorithms. Stochastic and Bagging are quite comparable with two exceptions: i) on small sample data sets, Bagging degrades quickly, and ii) on noisy data sets, Bagging does significantly better than Stochastic. For the distribution of data sets that we used in this study, Table 7.8 shows that the

Table 7.10: Distribution of data sets for which each method did equally as good as the best method.

Method	Noise-free Artificial Data Sets	Noisy Artificial (Small Sample)	Noisy Artificial (Sufficient Sample)	Molecular-Biology Data Sets	Real-world Data Sets	TOTAL (29 data sets)
Single-model	xx		xx		xxx	7
Stochastic	xxxxx		xx	xx	xxxxxxxxxx	19
Partition	xxxxx	x	xx	x	xxxxx	14
Bagging	xx	xxxx		xx	xxxxxxxxxxxxx	21

best combination of model generation and evidence-combination algorithms are the Bagging and Stochastic algorithms using Likelihood Combination.

7.6 The small disjuncts problem

Here we test the hypothesis that the multiple models approach is able to lower error because it is able to lower the error rates of small disjuncts. In order to test this hypothesis, we need a canonical representation of the target concept. The target representation of class `illegal` in the King-Rook-King domain (Muggleton et al., 1989) consists of five disjuncts named in Table 7.11. Two other domains we studied for which we knew one the disjuncts of one of the classes were the student-loan domain (7 disjuncts, Pazzani & Brunk, (1991)) and the tic-tac-toe domain (8 disjuncts). These domains are two class domains. Unfortunately, HYDRA-MM using stochastic search achieves a lower error on the student-loan and tic-tac-toe domains by reducing the error rate of the complementary class; error increases slightly for the class whose canonical representation is known to be in near-disjoint DNF form. Accordingly we only use the King-Rook-Kind domain for this study.

The two classes in the King-Rook-King domain are `illegal` and `legal`. There are three pieces on the board: the white king, the black king and the white rook. To represent the state of the board, a 6-tuple over the set $\{1 \ldots 8\}$ is used to store the 3 pairs of coordinates. The board belongs to class `illegal` if the black king is in check (white's move) or if more than one piece occupies the same square. The background predicates used to learn on this domain are *equal-row(X,Y)* (true when X and Y are bound to the same integer in $\{1..8\}$), *between-row(X,Y,Z) (true when* $X < Y < Z$ *or when* $X > Y > Z$*) and* near-row(X,Y) (true when $X = Y - 1$ or when $X = Y + 1$).

The error rate of each disjunct is calculated as follows. Each test example is tagged with the disjuncts that are satisfied by that example. If an error is made when that example is a test example, the error count for all its associated

Table 7.11: Effect of the Stochastic multiple models approach on small disjuncts of KRK (200 training examples, 11 models, evidence combination: LS, $\beta = 0.8$).

Disjunct name	Frequency	Single model % of error	Multiple model % of error	Single model error rate	Multiple model error rate
Rook attack King via column	32.8%	15.2%	0.0%	2.0%	0.0%
Rook attack King via row	32.0%	12.6%	3.1%	1.7%	0.2%
King attack King	27.5%	30.4%	18.2%	4.8%	1.4%
King on King	4.0%	13.1%	25.6%	14.6%	13.6%
W.King on Rook	3.7%	28.6%	53.1%	34.3%	30.4%

disjuncts is increased by $1/d$, where d is the number of disjuncts associated with that example.

Table 7.11 presents the results of this experiment. It shows the frequency of each disjunct and what proportion of overall error on the `illegal` class was caused by errors on test examples of that disjunct. The small disjuncts problem is evident: the last two disjuncts which are small (matching 4.0% and 3.7% of the examples of their class) are responsible for much higher proportions of the errors of commission: 13.1% and 28.6%. The table shows that Stochastic multiple models does not solve the small disjuncts problem. Surprisingly, the multiple models approach is able to do better than the single model approach on this data set primarily by reducing the error rates of *large* disjuncts!

7.7 Previous work

First, we will review previous work on comparing model generation methods and then we will review previous work that compares evidence combination methods.

Buntine's thesis (1990) includes a comparison between two tree generation methods: beam search and the "all pruned models" approach (Section 7.2). His work also includes a method of combining the learned decision trees into a compact form: an option tree. One option tree can typically represent hundreds of individual decision trees. The main result of this work is that different prunings of a single decision tree yields trees are all quite functionally similar and hence do not do as well as the more functionally diverse trees generated by beam search. Buntine also compares the effect of gain metric on decision-tree ensembles, showing that the Bayes gain metric does better than information gain. The best combination is provided by using the Bayes gain metric for learning and using option trees with non-uniform priors (the prior is proportional to $exp(-V)$ where V is the number of nodes in the decision tree).

Kononenko & Kovacic (1992) compare different model generation methods produced by varying the amount of search. They compare random-start hill-climbing, stochastic-search hill-climbing and simulated annealing. Their results show that the more intensive search methods yield ensembles that have lower error rates.

Kwok & Carter (1990) compare two ways of generating decision trees. In one, the user can vary the decision node at the root of a decision tree and in the second, the user can vary decision nodes further down the tree but not at the root. They show that varying the root leads to syntactically more dissimilar trees (which presumably are functionally more diverse) that produce ensembles with lower error rates. However, none of this work identifies domain characteristics under which each method is superior - we find that Bagging is superior for very noisy data sets unless the sample size is very small. Stochastic-search is superior to Bagging for relatively noise-free data sets.

There has been little comparison of evidence combination methods in prior work. The work of Buntine compares Uniform Voting with Bayesian Combination for option trees and shows that most of the time Bayesian Combination does better. Kononenko & Kovacic (1992) briefly mention that in internal experiments they compared Distribution Summation, Uniform Voting and Naive Bayesian Combination (similar to the Likelihood Combination method) and found that Naive Bayesian Combination does best. This result is consistent with our result that Likelihood Combination does best overall.

Drucker, Schapire & Simard (1993) and Drucker *et al.* (1994) compare two ways of combining evidence from neural networks. The first is uniform voting. In the second method, each network produces a vector of activation levels: one per class. Then, for each class, the sum of activations for that class is formed. The class with the highest aggregate activation level is the class predicted by the ensemble. This method is similar in spirit to Distribution Summation. The authors show that this method does better than Uniform Voting. This result is consistent with our result that Distribution Summation is better than Uniform Voting on very noisy data sets.

7.8 Discussion

This chapter has presented the approach of learning more than one model of the training data and combining classifications or evidence from those models in order to make more accurate classifications than done by the single model approach. HYDRA-MM, the multiple models version of HYDRA was shown to be *statistically significantly* more accurate than HYDRA on approximately 50% of the 29 data sets used in the comparison. For all but one of the remaining data sets, HYDRA and HYDRA-MM had no significant difference in accuracy.

The chapter also outlined some of the problems of producing a single model. This problem is that there may be many models that are close to the model that maximizes the gain metric with respect to the training set and these models may make significantly different classifications on the test set. Yet the single model approach ignores these models. Increasing lookahead does not ameliorate this problem so apparently, the only solution is to retain multiple models. The only situation in which one would want to retain just a single model is when that model has a significantly (using Fisher's exact test (Finney et al., 1963) for example) greater gain than the other models. We hypothesize that on the breast-cancer data set, using 0.8 as a value for β allows significantly sub-optimal models into the ensemble and this is why HYDRA-MM did poorly on this data set. Future work should concentrate on testing a better method for deciding which models are allowed into the ensemble. A statistical test such as Fisher's exact test coupled with beam search might be a better way of constructing an ensemble.

The comparison of the evidence combination methods showed that for all model-generation methods, Likelihood Combination did best overall (lowest rank in Table 7.8). It should be noted, however, that these kinds of aggregate results are dependent on the distribution of data sets used in the study. The Uniform Voting method was inferior to Likelihood Combination but not by much. Distribution Summation did relatively poorly on domains containing near-disjoint DNF classes and Bayes did poorly on domains in which the posterior probability (voting weight) of one model dominated those of others. However, the successful results of Buntine (1990) and Oliver & Hand (1995) using Bayesian Combination suggest that with many more models (hundreds) the Bayesian method does quite well. Distribution Summation was significantly better than the other combination methods on very noisy natural data sets.

The chapter also presents results comparing three multiple-model generation methods. Overall the Partition method did worst and there was little overall difference between Stochastic and Bagging although there were groups of data sets where one was clearly better than the other. Bagging was better than Stochastic on very noisy data sets (Diabetes, Breast Cancer, Heart and Primary Tumor). However if the sample size is small and there is noise (LED20a, LED40a), Bagging degrades quickly. Stochastic did better than Bagging on on artificial, noise-free data sets.

Chapter 8
The link between error correlation and error reduction

Learning multiple models has been shown to reduce generalization error (references in Section 7.2) but the amount of error reduction (ensemble error rate divided by single model error rate) varies greatly from data set to data set. In particular, we examine the hypothesis that:

> H1: The amount of error reduction is negatively correlated with the degree to which the models in the ensemble make errors in a correlated manner.

That is, if the models make errors in a correlated manner, then the amount of error reduction will be small. On the other hand, if the models make errors in an uncorrelated manner, the amount of error reduction will be large. This hypothesis stems from earlier theoretical results (Hansen & Salamon, 1990; Perrone & Cooper, 1993) that show that when the models make errors in a *completely* independent manner, then error can be reduced arbitrarily down to the Bayes rate by increasing the number of models. This is also echoed in Kong & Dietterich (1995) in which they hypothesize that their error-correcting output codes approach is able to reduce error because it relies on learning several models that make errors in an uncorrelated manner. However this earlier work does not experimentally verify whether the models are making errors in an uncorrelated manner nor does it model the *degree* of error reduction. This chapter presents a precise measure, ϕ_e, of the degree of correlatedness of errors among models in an ensemble and uses this to examine the H1 hypothesis for four classes of models: rule-set models as learned by HYDRA, unpruned decision trees, pruned decision trees and artificial neural networks.

Note that ϕ_e provides a post-hoc way of understanding the amount of error reduction since it is based on the error patterns of the models on the *test* examples. Section 8.2 examines the possibility that the amount of error reduction can be *predicted* from statistics gathered during learning. In particular, the section examines the role of *gain ties* on the amount of error reduction. The role of gain ties also explains why on some data sets it is possible to learn models that make uncorrelated errors whereas on other data sets this is not possible.

Sections 8.3 and 8.4 examine other factors that may explain the variation in amount of error reduction. These factors are the amount of class noise and the number of irrelevant attributes.

The chapter concludes with the presentation of an argument that the goal of ensemble learning should not be to learn ensembles whose constituents make errors in a completely indepenndent, uncorrelated manner. Rather, the learning method should aim to learn models whose errors are dependent but negatively correlated. Individual model errors thus arranged will lead to fewer overall ensemble errors than if the individual models made errors in an independent manner. One could even envisage models that are specialists - they may have a high overall error rate but may be accurate on a part of the example space that has not been well modeled by previously learned models.

8.1 Error Correlation and Error Reduction

This section begins by presenting precise definitions of error correlation and error reduction. Next, it is shown that the degree of error reduction is negatively correlated with the degree of error correlatedness, ϕ_e, for rule-set models. Ensembles whose models make errors in an uncorrelated manner will tend to reduce error by a greater amount. The following sections (8.1.3 and 8.1.4) show that amount of error reduction is also negatively correlated with error correlatedness for decision tree ensembles and neural network ensembles. Therefore, the section empirically confirms the H1 hypothesis for a wide variety of models.

8.1.1 Definition of error correlatedness

Now we present precise definitions of the degree of error reduction (E_r) and the degree of error correlatedness (ϕ_e). Two obvious measures comparing the error of the ensemble (E_e) to the error of the single model (E_s) are error difference ($E_s - E_e$) and error ratio ($E_r = E_e/E_s$). We use error ratio because it reflects the fact that it becomes increasingly difficult to obtain reductions in error as the error of the single model approaches zero. Error ratios less than 1 indicate that the multiple models approach was able to obtain a lower error rate than the single model approach. The lower the error ratio, the greater the error reduction and the better the situation. The error rates, E_e and E_s, are averages over several trials. Therefore, the error ratio is the ratio of these averages.

Let the ensemble $\mathcal{F}$ consist of the learned models $\{\hat{f}_1 ... \hat{f}_T\}$ and let the true, target function be denoted by f. Therefore, $f(x) = y$ means that example x belongs to class y. In order to define "the degree of error correlatedness," let

Table 8.1. Relationship between correlatedness and statistical independence.

$\phi_e > \phi^*$	Positively correlated	Dependent errors
$\phi_e = \phi^*$	Uncorrelated	Independent errors
$\phi_e < \phi^*$	Negatively correlated	Dependent errors

$p(\hat{f}_i(x) = \hat{f}_j(x), \hat{f}_i(x) \neq f(x))$ denote the probability that models $\hat{f}_i$ and $\hat{f}_j$ make the same kind of error. ϕ_e is then just the pairwise average of the probability of making the same kind of error. That is,

$$\phi_e(\mathcal{F}) = \frac{1}{T(T-1)} \sum_{i=1}^{T} \sum_{j \neq i}^{T} p(\hat{f}_i(x) = \hat{f}_j(x), \hat{f}_i(x) \neq f(x)) \tag{8.1}$$

The higher the values of ϕ_e ($\phi_e \in [0, 1]$) the more correlated the errors made by members of the ensemble.

The values of ϕ_e for the data sets presented in Table 8.3 were estimated on the *test* set of examples. Therefore, they cannot be used by the learning algorithm but provide us with an understanding of why error is reduced more in some data sets. In the table, an "i" suffix indicates number of irrelevant attributes, an "e" indicates number of examples, an "a" represents level of attribute noise and a "c" indicates level of class noise. '–' indicates a significant reduction in error. $1 - \phi_e$ can be used to measure the degree of *un*-correlatedness of errors. This will be denoted $\overline{\phi}_e$. In terms of $\overline{\phi}_e$, the hypothesis is that there is a positive correlation between the amount of error reduction, E_r, and the degree to which the models make errors in an uncorrelated manner, $\overline{\phi}_e$.

There is an intimate link between making uncorrelated errors and making errors in a statistically independent manner. Let C denote the number of classes and let ϕ^* denote the following special value of ϕ_e:

$$\begin{aligned}\phi^*(\mathcal{F}) = \quad & \textstyle\sum_{i=1}^{T} \sum_{j \neq i}^{T} \sum_{k=1}^{C} p(f(x) = k) \times \\ & \left[\textstyle\sum_{l \neq k} p(\hat{f}_i(x) = l | f(x) = k) \times p(\hat{f}_j(x) = l | f(x) = k)\right]\end{aligned} \tag{8.2}$$

This is the value of ϕ_e that would be obtained if all the members of the ensemble made errors in a pairwise, statistically independent manner for each class in the data. This value is used to define the meaning of "negatively correlated" as shown in Table 8.1.

Some authors (Hansen & Salamon, 1990; Perrone, 1993) have analytically demonstrated that making errors in an uncorrelated (independent) manner leads to a lower error rate for the ensemble and produces some desirable results relating to error reduction. Kong & Dietterich (1995) attribute the success of their error-correcting output code method to its ability to learn functions that make uncorrelated errors. However, our analysis above suggests that because ϕ^* is not the lowest possible value obtainable, one should aim to learn ensembles whose models make errors in a negatively-correlated manner. In Section 8.5 we present further arguments for the hypothesis that ensembles whose members make errors in a dependent but negatively correlated manner will have lower ensemble error rates than ensembles whose members make errors in an uncorrelated (independent) manner.

8.1.2 Results for rule-set models

For the following experiments, unless noted, we ran HYDRA using information-gain versus HYDRA-MM using information gain and stochastic learning with $\beta = 0.8$. Ensembles of eleven models were learned. For each data set, the average error rates (over thirty trials) of HYDRA and HYDRA-MM were determined and their ratio was used to calculate E_r. ϕ_e values that were averaged over thirty trials were also calculated for HYDRA-MM.

If the error ratios and ϕ_e values for the data sets for which there was significant error reduction are plotted in a scatter-plot and a least mean-squares linear fit is done (Figure 8.1), it can be determined that the tendency to make correlated errors explains 56% of the variance in the error ratio variable ($r^2_{\phi_e,E_r} = 0.56$).[1] We restrict our analysis to those domains for which error declined significantly because we are proposing ϕ_e as a model of error *reduction*. Of the 29 data sets used in this study, significant error reduction was obtained (when using stochastic learning and Uniform Voting) on 15 data sets. Error did not increase significantly for any of the remaining 14 data sets. For the other evidence combination methods, the values were 56% (Bayesian combination), 43% (Distribution Summation) and 41% (Likelihood combination). When Partition Sampling was used, the values were 60% (Uniform Voting), 40% (Bayesian combination), 35% (Distribution Summation) and 41% (Likelihood combination). These high values of r^2 are quite encouraging given that the data sets vary widely in type of class description, optimal Bayes error level, numbers of training examples and numbers of attributes. Another point to note is that ϕ_e is a pairwise measure, whereas what the error rate measures (under Uniform Voting) is the proportion of the test examples on which at least half of the members in the ensemble make an error.

How stable are these estimates of r^2? In particular, is it possible that we are able to get such high values of r^2 simply because of one point luckily appearing near the line of best fit? In order to measure the stability of these estimates of r^2, for each of the eight combinations of learning method and evidence combination

[1] These results are for 26 data sets rather than 29 since ϕ_e cannot be reliably estimated for the DNA data set since per trial, there is only one test example. Similarly, ϕ_e cannot be reliably be estimated for the LED20a and LED40a data sets since per trial, there is only one test example per class in those data sets.

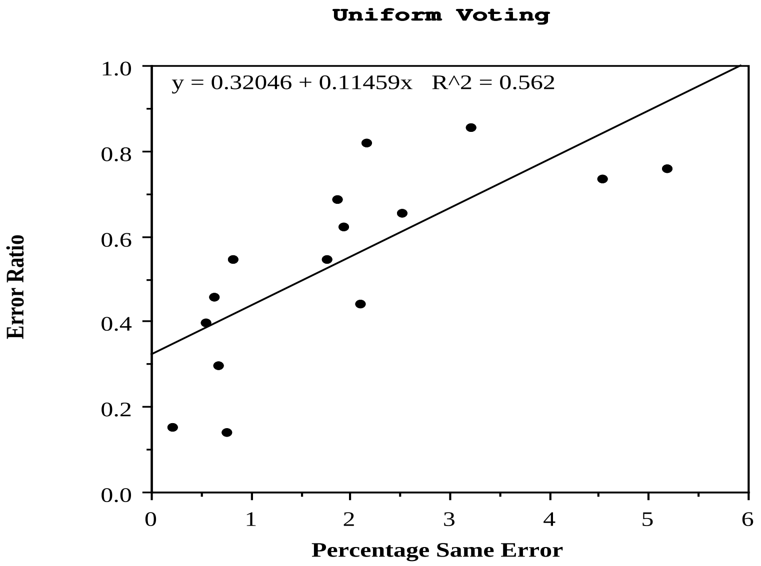


Figure 8.1: Plot of error ratio as a function of $100 \times \phi_e$ for rule-set models. One point represents one data set. Learning method: stochastic hill-climbing, evidence combination method: Uniform Voting). The figure shows that for rule-set models, the tendency to make correlated errors explains much of the variance in E_r, the error reduction variable.

method we calculated twenty-nine r^2 values - each time calculating what the r^2 would be if one of the 29 data sets were left out. This analysis (Table 8.2) shows that the r^2 values presented above do not depend critically on any single data set. We also performed significance tests to compute the likelihood of the observed results under the null hypothesis (that the population correlation, ρ, between E_r and ϕ_e equals 0). The tests showed that the likelihood of our data given H_0 was less than 0.01 for each of the eight combinations of learning method and evidence combination method. Therefore, we can conclude that

> There is a significant linear correlation between error ratio and the tendency to make correlated errors for all three rule-set model ensemble learning methods and four evidence combination methods used in this thesis.

In order to gain insight into why ϕ_e explains so much of the variance in error ratio consider the simpler problem of modeling variation in error *within* a given data set. This removes possibly confounding variables such as optimal Bayes error rate that vary from one data set to another. Assume that N trials have been conducted to yield N ensemble error values. Assume that the simplest evidence combination method (Uniform Voting) is used and that the data set contains two classes and that the ensemble contains just two models. In this situation, an ensemble error occurs if both the models make an error or if the models disagree and the tie is broken so as to cause an error. Assume that a tie will occur for

Table 8.2: Ranges of leave-one-out estimates of r^2 between error ratio and ϕ_e (the tendency of models to make correlated errors).

	Uniform Voting	Bayesian combination	Distribution Summation	Likelihood combination
Stochastic Hill-climbing	[0.54,0.61]	[0.47,0.71]	[0.37, 0.52]	[0.38,0.48]
Partition Sampling	[0.55,0.66]	[0.27,0.56]	[0.28,0.46]	[0.29,0.56]

a negligible proportion of the test examples. Under these assumptions, ϕ_e is an exact measure of ensemble error (E_e). In practice, ϕ_e is not a perfect measure of E_e because the ensemble usually consists of more than two models. In modeling across domains, E_r is a more suitable dependent variable. In such situations, it is much more difficult for error-correlatedness (ϕ_e) to model error reduction (E_r).

As ϕ_e is a pairwise measure, how well it models within-dataset ensemble error depends on the size of the ensemble. It is a better model of ensemble errors for ensembles of smaller size. The evidence combination method also affects the ability to model ensemble error using ϕ_e. ϕ_e is a better model of ensemble error obtained by Uniform Voting than it is for evidence combination methods in which different models are given different "voting" weights.

8.1.3 Results for decision trees

This subsection compares an implementation of ID3 for learning decision trees to a stochastic version of ID3 for learning ensembles of decision trees. It shows that there is a relationship between error reduction and error correlatedness for decision trees similar to that relationship for rule-set models. The second part of this subsection presents results showing error reduction and correlatedness of errors are also negatively correlated for *pruned* decision trees.

Learning decision tree ensembles: We use the method of top-down induction of decision trees (ID3 (Quinlan, 1986) with 1-step lookahead with respect to entropy minimization to learn a single tree. Unknown attribute values are handled by the method of token averaging (Quinlan, 1986).

Stochastic search is used to generate multiple trees. We consider all decision tree splits whose resultant entropy (Quinlan, 1986) is within some factor $1/\beta$ of the entropy of the split with the lowest entropy. For our experiments, we set this factor to 0.8. The probability of choosing a split from this set is proportional to

Table 8.3: Comparison of errors made by single decision tree and an ensemble consisting of eleven stochastically-learned decision trees (Uniform Voting).

Domain	Base Error Rate	Num. Training Eg.s	1 Dec. Tree Error	11 Dec. Trees (Uniform) Error	1 Dec. Tree Accuracy	11 Dec. Trees (Uniform) Accuracy
Led 8i	90.0%	30	13.1%	– 9.4%	86.9	90.6
Led 17i	90.0%	30	20.9%	– 12.3%	79.1	87.7
Tic-tac-toe	34.7%	670	15.9%	– 5.2%	84.1	94.8
Krkp	48.0%	200	5.8%	5.2%	94.2	94.8
Krk 100e	33.4%	100	3.8%	4.4%	96.2	95.6
Krk 200e	33.4%	200	1.8%	1.7%	98.2	98.3
Krk 160e 5a	33.4%	160	8.6%	8.6%	91.4	91.4
Krk 320e 5a	33.4%	320	5.7%	5.7%	94.3	94.3
Krk 160e 20c	33.4%	160	12.9%	11.8%	87.1	88.2
Krk 320e 20c	33.4%	320	9.4%	9.6%	90.6	90.4
Led 20a	90.0%	30	10.0%	10.0%	90.0	90.0
Led 40a	90.0%	30	26.0%	– 21.7%	74.0	78.3
DNA	50.0%	105	17.0%	– 6.4%	83.0	93.6
Splice	46.6%	200	24.6%	– 12.2%	75.4	87.8
Mushroom	50.0%	100	1.6%	1.2%	98.4	98.8
Hypothyroid	5.0%	200	2.3%	1.9%	97.7	98.1
BC-Wisconsin	34.5%	200	6.5%	– 4.4%	93.5	95.6
Voting	38.0%	100	6.5%	6.4%	93.5	93.6
Wine	60.2%	118	6.5%	– 2.8%	93.5	97.2
Iris	66.7%	50	5.4%	5.3%	94.6	94.7
Soybean	85.4%	290	13.9%	– 11.9%	86.1	88.1
Horse-colic	36.6%	245	17.0%	– 14.0%	83.0	86.0
Hepatitis	20.4%	103	25.2%	– 20.4%	74.8	79.6
Lymph.	45.3%	110	25.0%	26.5%	75.0	73.5
Audiology	74.7%	145	21.6%	22.3%	78.4	77.7
Diabetes	34.9%	200	31.5%	– 27.0%	68.5	73.0
B.Cancer	29.8%	190	36.6%	35.6%	63.4	64.4
Heart	45.9%	200	49.9%	– 45.3%	50.1	54.7
Primary-tumor	75.3%	225	64.0%	– 59.8%	36.0	40.2

1/Entropy.[2] We have not experimented with other values of $1/\beta$ - we hypothesize that our results will hold for other values of $1/\beta$ in $(0, 1)$.

Table 8.3 presents results using 29 data sets from 21 domains. For approximately half the data sets in Table 8.3 there is a statistically significant reduction in error rate when classifications are made using an ensemble of eleven trees (combined using Uniform Voting). No significant change in error occurs on the remaining data sets. Hence, on all the data sets, there is either a significant reduction or no change in error rate. The values of E_r and ϕ_e used in this experiment represent averages over thirty trials for each data set presented in Table 8.3.

[2]To prevent zero values for Entropy, we used the Laplace approximation for the probabilities involved in the Entropy expression.

However, the main point of this section is not to demonstrate that error is reduced due to the multiple models approach. Rather, we seek to explain the amount of error reduction, E_r, as a function of the tendency to make correlated errors, ϕ_e. The linear correlation coefficient (r_{ϕ_e,E_r}) between error correlation and error ratio can be used to measure how well error correlatedness models error reduction. The results show that for decision trees the tendency to make correlated errors explains 60% of the variance in the error ratio variable ($r^2_{\phi_e,E_r} = 0.60$) when using Uniform Voting.

This is empirical evidence for the hypothesis that there is a relationship between the degree to which error is reduced and the degree to which individual model errors in the ensemble are made in an correlated manner. Specifically, that relationship is of the "negative correlation" form - the greater the correlatedness, the smaller the error reduction.

However, it is better to conduct several trials to estimate the statistic $r^2_{\phi_e,E_r}$ so we conducted 10 trials.[3] Within each of these "meta-trials," 30 trials per data set were run. For the i-th meta-trial we used $i \times 10\%$ of the training data. So, for example, on the Tic-Tac-Toe domain, the original training set size was 670 training examples so this was augmented by 30 trials at 67 examples, 30 trials at 134 training examples etc.

Of the resulting 260 data sets (from 20 domains)[4] significant error reduction occurred on 163 occasions when using Uniform Voting. The figure shows that under the Uniform Voting combination function (top left of Figure 8.2), 53% of the variance in *amount* of error reduction is explained by the tendency to make the same correlated errors. This value is more reliable than the 60% value for $r^2_{\phi_e,E_r}$ mentioned earlier because it uses approximately ten times as much data. The figure also shows a similar relationship using the three other evidence combination functions: the degree to which error is reduced is negatively correlated with the degree to which constituents in the ensemble make individual errors in an correlated manner.

Because r is distributed normally for samples of large (greater than 30) size we can apply a significance test to see what the probability of achieving a r of 0.73 ($r^2 = 0.53$) under the null hypothesis, H_0, would be for 162 degrees of freedom. In this case, the null hypothesis would be that the population correlation, ρ_{ϕ_e,E_r}, between E_r and ϕ_e is 0. For each of the four combination functions, the probability of attaining the observed r values under H_0, is less than 0.0005 (120 degrees of freedom were used). Therefore, we can confidently say that the perceived linear correlation between ϕ_e and E_r is very unlikely to arise by chance. The 95%

[3]Note that the r^2 is being estimated from 30 pairs of values of E_r and ϕ_e; each value of E_r and ϕ_e is an average of 30 trials within each of those data sets.

[4]See first footnote in this chapter for why there are only 20 domains and only 26 data sets (rather than 29) from which to construct the amplified set of 260.

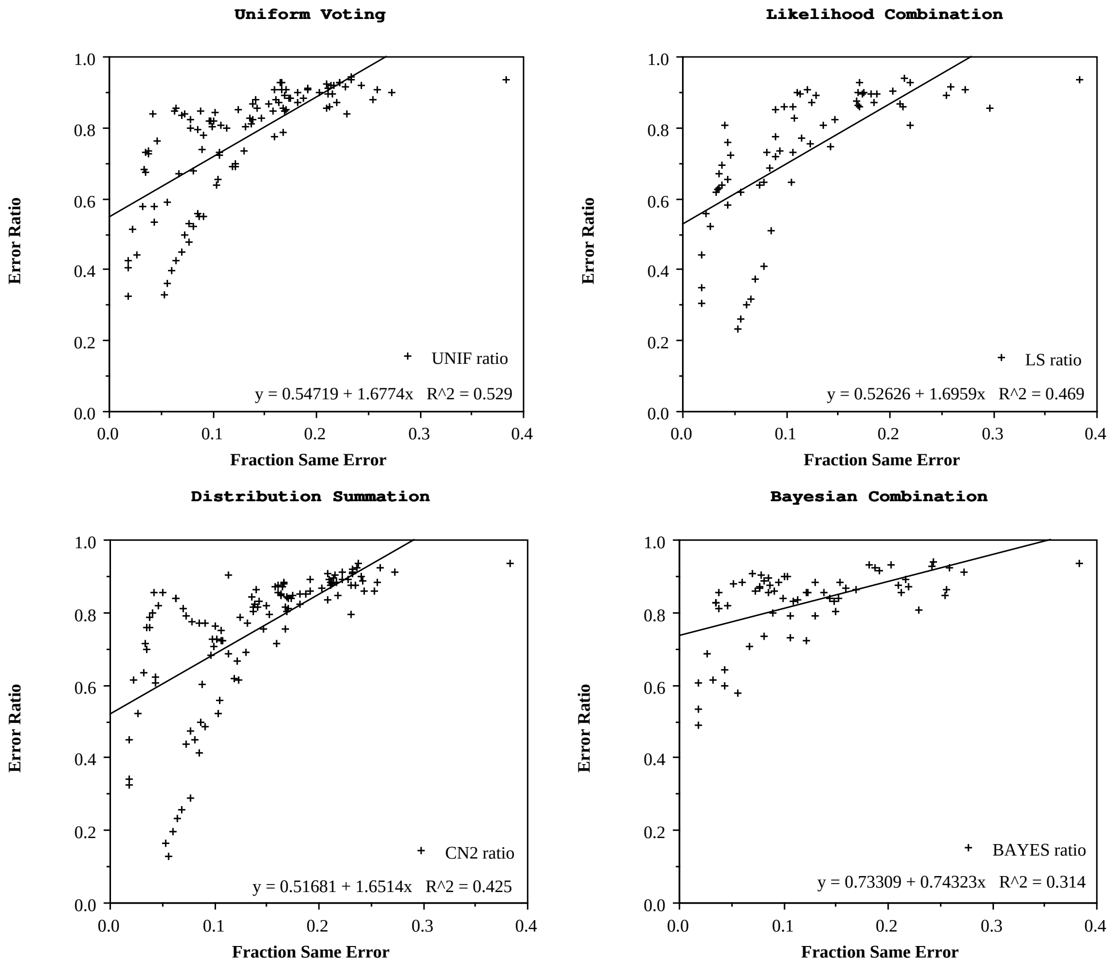


Figure 8.2: The figures above illustrate that greatest error reduction in decision tree ensembles is obtained for ensembles which make less correlated errors (have lower values of ϕ_e). One point represents one data set - a combination of a domain and a specific training set size.

confidence intervals around r^2 are $[38\%, 60\%]$ for Uniform Voting, $[20\%, 44\%]$ for Bayesian combination, $[28\%, 49\%]$ for Distribution Summation and $[32\%, 56\%]$ for Likelihood Combination.

In other experiments, we calculated $r^2_{\phi_e,E_e}$ *within* each domain. Note that this is between ϕ_e and error *rate*, E_e, not error *ratio*. This within-domain experiment factors out the influence of optimal Bayes error rate and other variables which may vary from domain to domain. For the within-domain experiments, a separate value for ϕ_e is calculated per trial, rather than averaging over 30 trials. In these experiments, we obtained very high values for $r^2_{\phi_e,E_e}$ for most domains; up to 96.8% for tic-tac-toe.

It may be argued that the multiple models approach offers such great error reductions for decision trees simply since we are not using any method to prevent overfitting for the noisy data sets. To correct for this, we ran experiments comparing pruned decision trees to ensembles of pruned decision trees. χ^2 pre-pruning with a 99% confidence level was used. The results show that under Uniform Voting, 56% of the variance in error reduction in ensembles of pruned decision trees can be explained by the tendency to make correlated errors. For the other combination methods, the results are: $r^2_{\phi_e,E_r} = 0.56$ for Bayesian Combination, $r^2_{\phi_e,E_r} = 0.43$ for Distribution Summation and $r^2_{\phi_e,E_r} = 0.41$ for Likelihood Combination.

To summarize: our results provide an explanation of why the multiple models approach leads to great error reduction in some domains but hardly any in other domains. The results show that there is a negative correlation between the amount of error reduction and the amount of correlatedness of errors - the less correlated the individual model errors, the greater the error reduction as compared to the error of the single model learned on the same data.

8.1.4 Results for neural networks

We also ran experiments with ensembles of 1 hidden-layer, feed-forward artificial neural networks (Rumelhart & McClelland, 1986) to determine whether a similar relationship between error correlation, ϕ_e, and error reduction, E_r, holds for neural networks. The parameters for the neural network learning algorithm were set as follows:

- Learning Rate, $\eta = 0.1$.
- Momentum scale factor, $\alpha = 0.9$.
- Initial weights uniformly distributed in range $[-0.3, +0.3]$.
- Maximum number of epochs = 1000.
- Number of hidden units = $0.1\times$(# input units + # output units).

For multi-class classification tasks, an output node per class was created. For test-time classification, the example was assigned to the class with the highest activation level.

In order to generate multiple neural networks from the same training data, different settings for the initial weights of the links in the neural network were used. This was repeated eleven times to yield an ensemble of networks.

Two evidence combination methods were used since it is not apparent how to convert the Likelihood Combination and Bayesian Combination methods to work with neural networks.[5] In Uniform Voting, the output of each network is a class label. The votes for each class label are simply tallied and the class with the highest number of votes is output as the prediction made by the ensemble of networks. In Activation Summation (the neural network analogue of Distribution Summation), the activation levels of the neural networks are also taken into consideration. Each networks outputs a vector of activation levels, one level per class. These vectors are then added up and the class with the greatest composite activation level is then output as the prediction made by then ensemble.

As Table 8.4 shows, we did not run experiments involving the 0-irrelevant attribute LED data sets (LED20a, LED40a) or the DNA promoters data set because ϕ_e cannot reliably be estimated for these data sets. Of the remaining 26 data sets, significant error reduction was obtained for 7 data sets when using Uniform Voting. When using Activation Summation, significant reductions were obtained on 9 data sets. Figure 8.3 plots error reduction as a function of correlatedness of errors for these data sets. The figure shows that there the tendency to make correlated errors explains nearly half the variance in the amount of error reduction when using Activation Summation. When using Uniform Voting, correlatedness of errors explains 40% of the variance in amount of error reduction.

In summary, these experiments provide further evidence for the claim that for many classifiers, a significant amount of the *degree* of error reduction obtained through the use of multiple models is explainable by the tendency to make correlated errors.

8.2 Gain ties and error reduction

The amount of error correlation provides a post-hoc way of understanding why the multiple models approach reduces error more for some data sets than

[5]This is not to say that posterior probabilities for the Bayesian method cannot be computed for neural networks. Evidence combination among neural networks is simply not a focus of this research. Buntine presents a method for calculating the posterior probability of a neural network in (Buntine, 1991).

Table 8.4: Error rates comparing a single neural network to ensembles of eleven neural networks. The training and testing sets used here were the same as those used for the rule-set model and decision tree experiments. A "–" indicates a statistically significant decrease in error rate. The neural network trials tended to have higher variance than the rule-set and decision tree trials. Hence, fewer of the differences between the ensemble and single model are significant according to the 2-tailed, paired t-test.

Domain	Single Network Error Rate	11 Net.s (Unif. Vote) Error	11 Net.s (Actvn. Sum.) Error	Unif. Error Ratio	Actvn. Sum Error Ratio	Single Network Accu.	11 Net.s (Uniform Vote) Accuracy	11 Net.s (Activation Sum.) Accuracy
LED8i	26.0	– 12.5	– 10.9	0.5	0.4	74.0	87.5	89.1
LED17i	53.8	– 43.7	– 41.3	0.8	0.8	46.2	56.3	58.7
Tic-Tac-Toe	9.1	5.9	– 2.7	0.7	0.3	90.9	94.1	97.3
KRKP	6.7	5.8	5.8	0.9	0.9	93.3	94.2	94.2
KRK100e	4.9	4.9	4.9	1.0	1.0	95.1	95.1	95.1
KRK200e	2.0	1.7	1.7	0.9	0.9	98.0	98.3	98.3
KRK160e,5a	10.2	– 7.3	– 7.3	0.7	0.7	89.8	92.7	92.7
KRK320e,5a	12.0	6.7	– 6.3	0.6	0.5	88.0	93.3	93.7
KRK160e,20c	17.4	16.4	15.7	0.9	0.9	82.6	83.6	84.3
KRK320e,20c	39.8	40.6	38.2	1.0	1.0	60.2	59.4	61.8
Splice	15.4	– 12.0	– 11.8	0.8	0.8	84.6	88.0	88.2
Mushroom	3.4	3.2	3.2	0.9	0.9	96.6	96.8	96.8
Hypothyroid	3.1	2.8	2.9	0.9	0.9	96.9	97.2	97.1
Iris	6.0	6.0	6.0	1.0	1.0	94.0	94.0	94.0
Wine	0.8	0.7	0.7	0.9	0.9	99.2	99.3	99.3
BC-Wisconsin	4.0	4.0	4.0	1.0	1.0	96.0	96.0	96.0
Voting	5.5	5.2	5.4	0.9	1.0	94.5	94.8	94.6
Soybean	11.6	– 9.2	– 9.2	0.8	0.8	88.4	90.8	90.8
Horse-colic	15.4	14.4	14.2	0.9	0.9	84.6	85.6	85.8
Hepatitis	22.5	21.3	21.8	0.9	1.0	77.5	78.7	78.2
Lymph.	18.7	18.9	18.7	1.0	1.0	81.3	81.1	81.3
Audio	22.9	– 20.6	– 20.2	0.9	0.9	77.1	79.4	79.8
Diabetes	42.5	42.5	42.5	1.0	1.0	57.5	57.5	57.5
Primary-tumor	63.9	– 56.9	– 56.8	0.9	0.9	36.1	43.1	43.2
Heart	42.5	42.1	42.2	1.0	1.0	57.5	57.9	57.8
B.Cancer	36.8	35.5	36.0	1.0	1.0	63.2	64.5	64.0

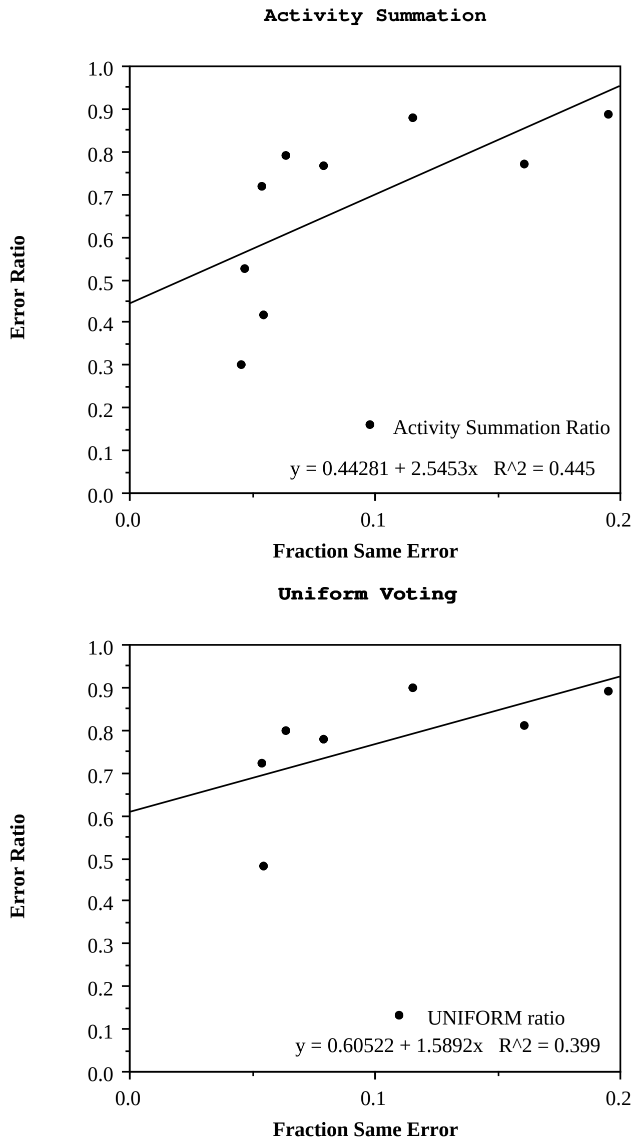


Figure 8.3: The figures illustrate that for neural network ensembles, the tendency of models in the ensemble to make correlated errors explains a significant amount of the variance in the amount of error reduction, E_r.

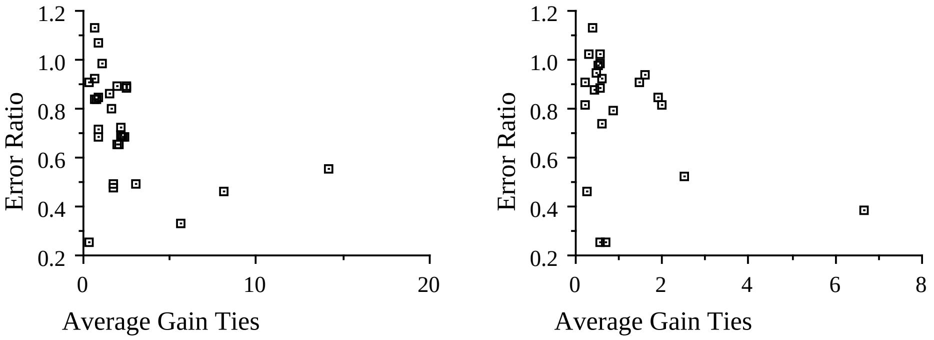


Figure 8.4: Error ratio as a function of average gain ties for decision trees (left) and rule sets (right). The ensembles of decision trees contained eleven, stochastically learned decision trees with respect to the entropy gain function. The ensembles of eleven rule sets were learned using stochastic hill-climbing and combined using likelihood combination. Similar plots are obtained for other evidence combination methods.

for other data sets. In this section, we explore whether we can approximately *predict* (during learning) the expected amount of error reduction due to the use of multiple models. In other words, we seek to understand why the stochastic learning algorithm produces models that make less correlated errors in some data sets and more correlated errors in other data sets.

Each time the stochastic generation method is run, it uses the same training data. However, it is able to generate different models because it randomly picks from the candidates whose gain is within some factor β $(\beta \in (0,1))$ of the gain of the best candidate. If there are many such candidates then the possibility for syntactic variation from model to model is greater. It is our hypothesis that greater syntactic variety leads to models that make less correlated errors. Hence, if we can measure (during learning) the amount of potential syntactic diversity, we can estimate the degree to which the resulting models will make correlated errors.

As a first approximation measure of the amount of syntactic variety in a data set as experienced by a learning algorithm, consider the number of candidates that tie for the highest information gain. If n candidates tie for gain, that event is recorded as representing $n-1$ ties in gain. The total number of ties experienced during learning a model is then divided by the number of literals or decision nodes in the model to produce the quantity g - the "average number of gain ties" for that data set. Values of g used in Figure 8.4 represent averages over thirty trials. A large number of such ties is a problem for a hill-climbing deterministic learner but represent an opportunity for the multiple model learner. Figure 8.4 plots error ratio as a function of average gain ties. The figure shows that some of the largest reductions in error are obtained for data sets for which such ties are frequent. For instance, there were, on average, 14.2 gain ties on the Wine data set and 8.1

for the Splice data set! This underscores the difficulty experienced by a greedy, hill-climbing algorithm on such data sets. Even with further lookahead, the single-model approach still can only produce a single model as its output. Therefore, unless the sample size is very large, the single-model approach will still be at a disadvantage. The figure also shows that a high average value for ties in gain appears to be a sufficient but not *necessary* condition for significant reduction of error. For example, the multiple models approach is able to achieve low error ratios on the Tic-Tac-Toe and the noise-free LED variants (bottom left of Figure 8.4) even though there are not many ties in gain for those data sets.

The gain ties measure is a rough approximation of the potential for syntactic diversity and the hypothesized resulting diversity of errors. For instance, the measure can be fooled by two attributes whose definitions are identical. We tried some variants that took the variability in the extensions of the candidates into account but found no measure that provided a significantly better estimation of error reduction. Other measures that counted "near gain-ties" or gain ties weighted by the number of examples at that candidate also did not yield a better estimate of error reduction.

8.3 Effect of class noise

The results of Table 7.4 showed that the majority (80%) of data sets for which unimpressive error ratios (above 0.8) were recorded were data sets with significant amounts of noise. Furthermore, experiments on the LED domain provided preliminary evidence that the addition of attribute noise increases (worsens) error ratios. In this section we follow up on that hypothesis by asking:

> How does increasing the amount of class noise affect the amount of error reduction?

We choose to study the effect of class noise rather than attribute noise because attributes in some domains have more values than attributes in other domains and an attribute with fewer values is more likely by chance to have large information gain. Therefore it would not be easy to compare levels of attribute noise across domains.

Table 8.5 shows the effect of adding class noise to four very different kinds of data sets (noise was only added to the training data). All the experiments in this section and the next were done with rule-set models. We chose the wine and tic-tac-toe data sets because the multiple models approach was able to reduce error by a large amount (error ratios of 0.16 and 0.22 respectively) for these data sets. We wanted to see if this advantage would be eroded by the addition of noise. The

Table 8.5: Effect of increasing class noise on error ratios (using 11 stochastically-learned models and Uniform Voting for evidence combination). Similar plots are obtained for other evidence combination methods and the other learning method. The table shows that increased noise prevents the multiple models approach from achieving error ratios as impressive as those obtained at lower noise levels.

Class Noise	KRK 100 Err. ratio	KRK 100 ϕ_e	TTT 200 Err. ratio	TTT 200 ϕ_e	Wine Err. ratio	Wine ϕ_e	BC-Wisc. Err. ratio	BC-Wisc. ϕ_e
10%	.85	3.6%	.70	3.9%	.23	1.5%	.55	2.7%
20%	.88	5.0%	.77	5.6%	.30	2.5%	.56	3.6%
30%	.98	7.0%	.92	7.4%	.48	4.1%	.62	4.4%
40%	.95	8.9%	.95	8.9%	.53	5.2%	.69	5.5%

Table 8.6: Distribution of ensemble errors as a function of the number of models correctly classifying a test example. Learning method: stochastic hill-climbing; evidence combination method: Uniform Voting. Eleven models were combined using Uniform Voting so an ensemble error occurs if five or fewer of the models correctly classified the example. The table shows that as noise increases, not only does the number of test examples incorrectly classified increase, but the *distribution* of errors changes - the fraction of occasions on which none of the models got the test example correct increases.

Number of models correctly classifying test example	BC. Wisc. 10% noise	BC. Wisc. 10% noise	BC. Wisc. 40% noise	BC. Wisc. 40% noise
	% of ensemble errors	Cumulative % of ensemble errors	% of ensemble errors	Cumulative % of ensemble errors
0	15.4%	15.4%	27.6%	27.6%
1	7.7%	23.1%	11.2%	38.8%
2	20.5%	43.6%	12.2%	51.0%
3	7.7%	51.3%	18.4%	69.4%
4	15.4%	66.7%	14.3%	83.7%
5	33.3%	100.0%	16.3%	100.0%

table shows that for each of the four chosen data sets the advantage yielded by the multiple models approach is indeed eroded as class noise is increased.

Perhaps surprisingly, as noise increases, the degree of correlatedness of errors also increases (one might have expected it to decrease). More careful examination of the patterns of errors of models in the ensemble shows that at 40% noise, a relatively large proportion of the test examples on which the ensemble made an error were incorrectly classified by *all* the models in the ensemble. That is, as noise increases, some of the examples become "hard" for all the ensembles. This is why the correlatedness of errors increases as noise increases.

In a follow-up experiment (Table 8.6), we studied the distribution of the ensemble errors. We wanted to know what proportion of the ensemble errors were caused by all the models making an error and what proportion were caused by a narrow majority of models making an error. The first column indicates the number of models that correctly classified the test example. The remaining columns are arranged in two groups. Columns two and three present results for 10% class noise, the last two columns present results for 40% noise. The i-th row corresponds to test examples that were *correctly* classified by i (out of 11) models. The first column in each set indicates the number of test examples characterized by that situation. Let a m/n split indicate the situation for a test example where m models make a correct classification and n make a mistake ($m + n = 11$). Therefore, the table indicates that a 0/11 split occurred on 15.4% of the test examples after learning with 10% class noise and it occurred on 27.6% of the test examples after learning with 40% class noise. Therefore, the table indicates that that as noise level increases, all the models misclassify a test example on a *greater proportion* of the test examples for which an ensemble error is made. This indicates that as noise level increases, some test examples become more difficult for all the models and hence are incorrectly classified by most of the models. This also explains why correlatedness of errors increases as noise increases.

Figure 8.5 compares the error ratio (11 models, stochastic learning, Uniform Voting) with 0% added noise to that with 40% added noise. In 28 out of 29 cases, the addition of noise causes the error ratio to increase towards 1 indicating the erosion of the advantage of the multiple models approach as noise increases. The multiple models approach is able to "milk" more out of clean (relatively noise-free) data but as noise increases, the information in the data decreases so even the multiple models approach cannot do much better than the single model approach.

In summary, the answer to the question of this section ("How does increasing the amount of class noise affect the amount of error reduction?") is that increasing class noise causes the multiple models approach (at least as described in this paper) to produce poorer (higher) error ratios. Extrapolation of these results suggests that at 100% noise the error ratios for all data sets would be 1.0. This makes sense because the training data contains no discriminatory information so there is no

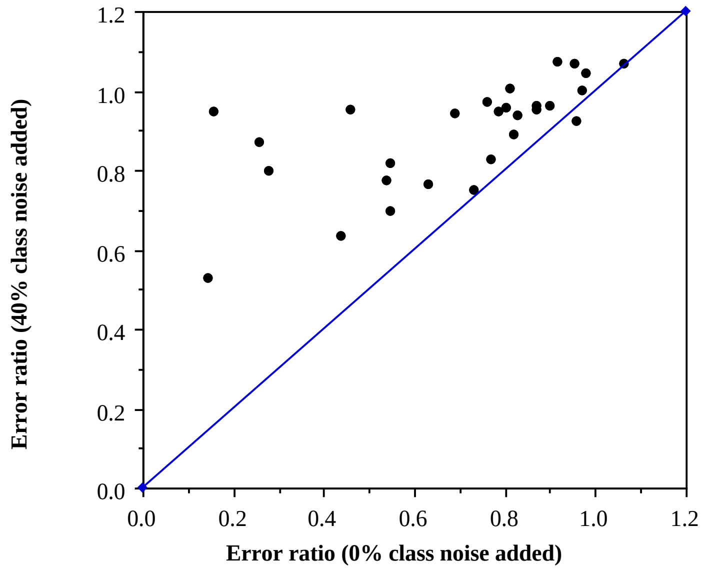


Figure 8.5: Comparison of error ratios at 0% added class noise and 40% added noise. Learning method: stochastic hill-climbing; evidence combination method: Uniform Voting.

reason to expect the multiple models approach to do better than the single models approach. When there is little noise, the multiple models approach is able to "milk" the data much better than the single model approach. However, as noise increases, the margin for advantage for the multiple models method decreases.

8.4 Effect of irrelevant attributes

The experiments presented in Section 7.3.1 provide preliminary evidence that the addition of irrelevant attributes allows the advantage gained by using multiple models to increase. That is, error ratios decrease as the number of irrelevant attributes increase. In this section we describe further experiments to explore this question:

> How does increasing the number of irrelevant attributes affect the amount of error reduction?

To study this question, we added varying number of Boolean irrelevant attributes to the Tic-tac-toe, King-Rook-King, Breast Cancer Wisconsin and Wine data sets. We chose boolean attributes rather than constructing irrelevant attributes whose values were domain specific because the attributes in some data sets can take on many more values than attributes in other data sets. Attributes that can take only a few values are more likely by chance to have high information gain than attributes that can take on many values. Thus using Boolean attributes allows easier comparability across domains.

Table 8.4 corroborates the hypothesis that the multiple models approach is able to attain especially impressive error reductions when many irrelevant attributes are present in the data. The table shows that error ratio decreases as a function of increasing numbers of irrelevant attributes. To understand this, consider the Uniform Voting evidence combination scheme. For the multiple models approach to make an error due to irrelevant attributes, at least half of the learned models need to involve an irrelevant attribute that leads to a classification error. If the number of irrelevant attributes is not too large, it is unlikely that at least half of the models will be affected in this manner. Therefore, the multiple models approach will not make an error in this situation. But the single model approach need only make a mistake due to learning a rule involving an irrelevant attribute early in its separate and conquer strategy for most of the subsequent rules to go off track. Hence the single model approach is much more likely to suffer due to irrelevant attributes.

Figure 8.6 extends the irrelevant attributes experiment to all 29 data sets. It plots the error ratio obtained after the addition of 50 irrelevant binary attributes against the error ratio before the addition of any irrelevant attributes. The fact that most of the plotted points lie below the diagonal indicate that for most of the data sets adding irrelevant attributes leads to situations where the multiple models approach is able to do better by a greater extent - i.e. the error ratios with 50 added irrelevant attributes are lower than they are with 0 added irrelevant attributes.

Table 8.4 also shows that the average number of gain ties experienced increases as the number of irrelevant attributes increases. This confirms the results (Section 8.2, Figure 8.4) that better (lower) error ratios are obtainable for data sets where the learning algorithm experiences more gain ties.

Consider, however, what would happen if an arbitrarily large number of irrelevant attributes were to be added to a data set. By adding enough irrelevant attributes, one could force all the learned models to go astray. In this situation, one would expect that the error ratio should go to 1 as both the deterministic and multiple models approaches would perform at chance level. Hence, we predict that for large enough numbers of irrelevant attributes the error ratio would *increase* with increasing numbers of irrelevant attributes. This hypothesis is difficult to test with data sets of reasonable size because such data sets tend to have literals with high information gain and one needs exponentially many irrelevant attributes for an irrelevant attribute to have higher information gain purely by chance. So, to test this hypothesis, we performed 100 trials with training sets of size 20. In particular, we were interested to see if the exceptionally low error ratio obtained on the wine data set could be made to increase with increasing numbers of irrelevant

Table 8.7: Error ratio as a function of number of added boolean irrelevant attributes (using Uniform Voting of eleven stochastically generated models). The number below each data set identifier indicates the number of training examples. "5a" indicates 5% attribute noise. Similar results are obtained for other evidence combination methods and the other learning method.

Number of irrelevant attributes	KRK 100 Error Ratio	KRK 100 Avge. gain ties	KRK 5a 160 Error Ratio	KRK 5a 160 Avge. gain ties	Splice 200 Error Ratio	Splice 200 Avge. gain ties	BC. Wisc. 200 Error Ratio	BC. Wisc. 200 Avge. gain ties	Wine 118 Error Ratio	Wine 118 Avge. gain ties
0	0.85	0.42	0.81	0.54	0.44	2.51	0.55	0.85	0.13	5.03
3	0.73	0.47	0.67	0.58	0.42	2.78	0.53	0.45	0.13	6.61
20	0.66	0.55	0.64	0.59	0.39	2.81	0.45	0.67	0.11	23.1
50	0.52	0.96	0.55	1.00	0.38	2.93	0.41	1.19	0.11	104.2

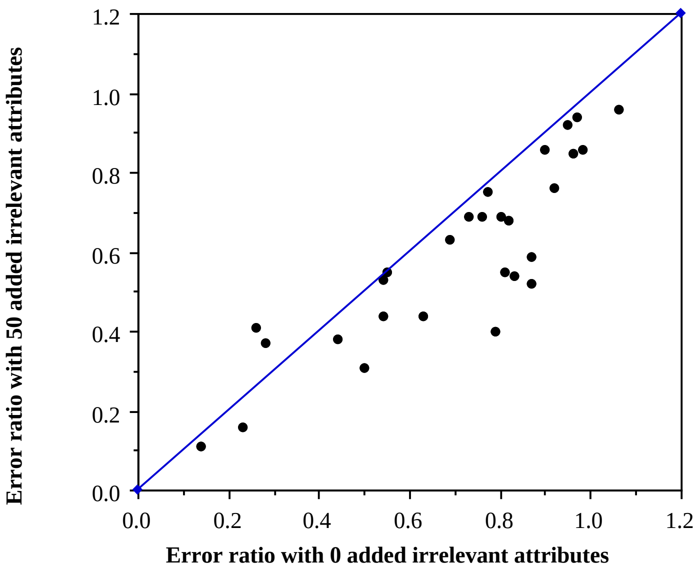


Figure 8.6: Comparison of error ratios with 0 added irrelevant binary attributes and 50 added irrelevant binary attributes. Learning method: stochastic hill-climbing; evidence combination method: Uniform Voting.

Table 8.8: Error ratio as a function of number of added boolean irrelevant attributes for *small* sample sizes (using ensembles of eleven stochastically-learned models; combined with Uniform Voting). The number below each data set identifier indicates the number of training examples. Adding irrelevant attributes to these small samples models the effect of adding thousands of irrelevant attributes to samples of normal size - the prediction is that given enough irrelevant attributes, error ratios will go towards 1.0 for any data set.

Number of irrel. attributes	KRK 20	KRK 5% attr. 20	Splice 20	BC. Wisconsin 20	Wine 20
0	1.00	0.98	1.00	0.68	0.44
3	0.99	1.06	0.91	0.67	0.45
20	1.08	1.01	0.92	0.68	0.59
50	0.95	0.97	0.92	0.65	0.64

Table 8.9: An arbitrary arrangement of individual model errors on 100 test examples. A "X" indicates an individual model error. An ensemble error occurs for test examples (columns) in which there are more than 2 individual model errors.

	Eg1	Eg2		Eg99	Eg100
Model1	X	X			X
Model2		X			X
Model3	X		...	X	
Model4				X	X
Model5		X			

attributes. Table 8.4 shows that for very small training set sizes, adding irrelevant attributes makes no significant difference to error ratios in 4 domains hence negating the benefit that irrelevant attributes provided at normal sample sizes. On the wine data set, which has a very low error ratio at its normal sample size, adding irrelevant attributes at this small sample size actually increases the error ratio providing further support for the hypothesis that given enough irrelevant attributes, both the single model and multiple model approaches will perform at chance levels and error ratios will increase towards 1.

In summary, the answer to the question posed in this section ("How does increasing the number of irrelevant attributes affect the amount of error reduction?") is that error ratios initially decrease as irrelevant attributes are added thus providing an opportunity for the multiple models approach. However, beyond some point, adding irrelevant attributes will begin to hurt the multiple models approach and error ratios will begin *increasing* towards 1. In the limit, neither the single model approach or the multiple models approach will be much use, and the error ratio will be 1.

8.5 On making negatively-correlated errors

In this section we consider whether making errors in a negatively-correlated manner leads to lower values of error reduction than if the errors are made in an uncorrelated manner. Consider what an optimal arrangement of errors (that minimizes ensemble error rates) would look like. That is, we consider what kind of pattern of individual model errors will lead to the lowest possible ensemble error rate given that we cannot vary the error rates of the individual models.

Consider an arbitrary pattern of errors as shown in Table 8.9 for 5 learned models and 100 test examples. Assume that Uniform Voting is used so an ensemble error occurs if there are more than 2 errors for any example. Therefore, in order to minimize the ensemble error rate, the models should make errors in a pattern that minimizes the number of columns in which more than 2 errors are made.

Now, because we are able to rearrange the errors but we are not able to modify the error rates it follows that we can permute each row. The ensemble error minimization procedure operates by ordering the models - most error-prone first. Then the errors of the second model are permuted so that as many of them as possible occur on examples that were correctly classified by the first model. That is, the models should make errors on *disjoint* subsets (to as great a degree as possible) rather than on independently drawn subsets. This process continues so that for each model we arrange for the mistakes to be made on examples on which the fewest mistakes have been made by previous models. However, once the number of errors on an example exceeds $\lfloor \frac{T}{2} \rfloor$ (T is the number of models) then an ensemble error is conceded on that example. Then in order to keep minimizing ensemble error rate, it is best to arrange for subsequent models to make their errors on such "conceded" examples.

From this analysis it becomes clear that in order to minimize the number of ensemble errors, it is better for the constituent models to make errors in a *dependent* but negatively correlated way rather than in an independent (uncorrelated) way. Furthermore, once an ensemble error has been conceded on a test example, it is best if "subsequent" models make errors on that example rather than on other examples.

ϕ_e is a perfect measure of ensemble error rate *within* a domain given that only two models are in the ensemble, the domain only contains two classes and that ties occur on a negligible proportion of the examples. However, for ensembles of larger sizes, consider how the arrangement of errors such as that in Table 8.9 impacts the ability of ϕ_e to measure ensemble error rate. The number of ensemble errors simply counts the number of columns in the table on which more than $\lfloor \frac{T}{2} \rfloor$ errors occurred. Therefore, any rearrangement in the pattern of errors on the columns in which $\lfloor \frac{T}{2} \rfloor$ or fewer errors occurred has no bearing on the number of ensemble errors as long as the rearrangement does not cause more than $\lfloor \frac{T}{2} \rfloor$ error to occur in any given column. But these rearrangements do have an impact on ϕ_e which is simply a pairwise (2nd order) measure. A $\lfloor \frac{T}{2} \rfloor$-order measure would be needed to perfectly model the ensemble error rate. This is another reason why ϕ_e does not do a perfect job of modeling error rate within data sets or of modeling error ratio between data sets.

8.6 Previous work

There has been no previous work that models the *degree* of error reduction. The closest related work is that of Breiman (1994), in which he addresses the question: "When will multiple models help?" That is, under what circumstances will the multiple models approach lead to significant error reduction. Breiman postulates that whether or not error reduction occurs is a property of the learning algorithm. This contrasts with our work in which we look for domain or data set characteristics which will allow the multiple models approach to work. The work presented here also differs from that of Breiman in that it offers an approximate model of the *amount* of error reduction.

Breiman calls a learning procedure "unstable" if small changes in the learning set can cause large changes in the functional behavior (classifications on test examples) of the learned model. However, he does not offer a precise definition of unstable or the degree of unstability. He presents arguments that nearest-neighbor learning algorithms are not unstable, whereas regression and decision tree induction algorithms such as found in CART (Breiman et al., 1984) are unstable. Although Breiman explains the efficacy of the multiple models approach by learning program characteristics, and we do so using domain or data-set characteristics, both sets of factors must be needed to better explain the amount of observed error reduction.

8.7 Discussion

There are five main results presented in this chapter:

- **Correlatedness and error reduction:** We have presented results that show that a significant amount of the variation in the error reduction variable (the ratio of the ensemble error rate to the single model error rate) can be explained by variation in the variable ϕ_e which measures the tendency to make correlated errors. At the extreme case this is obvious ... if all the models make completely correlated errors, nothing can be gained by combining them. However, prior to this work, there has been no work that explains the amount of error reduction due to the use of multiple models. These results have been shown to hold for four kinds of learning algorithms (HYDRA - a rule-set model learning algorithm, ID3 - a decision tree learner, ID3 with pruning, and a neural network learner) over many evidence combination methods. We also present analysis (Section 8.5) explaining why error correlatedness cannot perfectly explain error reduction.

- **Gain ties:** However, the explanation of error reduction involving the tendency to make correlated errors is a post-hoc explanation. It would be useful if one could *predict* (during learning) the amount of error reduction that might be expected on a given domain from statistics about the domain gathered during learning. It would also be useful to understand why it is possible to learn models that make uncorrelated errors in one data set but why this is not possible for other data sets.

  To this end, the chapter presents the concept of "gain ties" - events during learning in which more than one candidate (literal being added to a rule, or decision tree split) has the equal highest gain.[6] The results show that if the data set has many gain ties, then it will probably have a lot of error reduction due to the use of multiple models. Significantly, the results also show that having many gain ties is not a *necessary* criterion for error reduction - few gain ties were encountered on the Tic-Tac-Toe data set and yet error was reduced on this data set by a large factor.

- **Class noise:** The chapter presents surprising results showing that the multiple models approach is most effective at low levels of noise. This runs counter to ideas about "averaging out the noise." The results show that as the noise level is increased, the error ratios obtained by the multiple models approach become less and less impressive and asymptote to 1.0. At 100% class noise the error ratios will all be 1.0 as expected. The results also show, perhaps surprisingly, that as noise increases, the models make errors in a more correlated manner.

- **Irrelevant attributes:** Irrelevant attributes (or weakly relevant attributes) offer an opportunity for the multiple models approach. The results show that as the number of irrelevant attributes is increased, the margin by which the multiple models approach out-performs the single model method increases. However, the story is not so simple. As one continues to add irrelevant attributes, eventually, both methods begin to do equally badly. In the limit with an infinite number of irrelevant attributes, any approach using a finite number of models is as lost as an approach using a single model. The exact point at which addition of irrelevant attributes begins to hurt the multiple models method is domain dependent.

- **Negatively-correlated errors:** Much prior theoretical work in the theory behind the multiple models approach suggests that one should learn models that make errors completely independently (in an uncorrelated manner). However, we have presented simple arguments to show that it is better to learn models that make errors in a dependent but negatively-correlated manner. Although asymptotic analytic results for such ensembles are not as easy to come by, it is easy to see that such ensembles will have lower error rates than ensembles consisting of models that make errors independently. The analysis also shows that "once" an ensemble error has been conceded on a test example, subsequent learned models should make their errors on such conceded examples.

[6] This is not done for neural networks since there is no such thing as a "gain tie" for that model class.

Chapter 9
Conclusions

The research presented in this thesis has produced results in a number of areas. First, it has produced results in the area of one-per-class concept learning and has addressed the main questions in that area: "what to do if a test example satisfies more than one class description?" and "what do to if a test example satisfies no class description of any class?" The approach used in this thesis is to learn a class description for each class consisting of probabilistic rules - that is rules augmented with reliability estimates that can be probabilities. Second, the research presents results on the small disjuncts problem. Third, it is shown that the one-per-class approach, together with reliability estimates provides a new, and in some ways better, method of noise-tolerant learning of relational rules. The research also presents a number of results in the area of multiple model learning. The conclusions one can draw from this research and the lessons to be learned are presented below. More detailed summaries can be found at the end of each chapter.

1. **One-per-class learning:**

   (a) *What to do if a test example satisfies more than one class description?"* In the framework of the research presented in this thesis, this question boils down to: "What kind of reliability weight should be used?" The results are that using Laplace accuracy or LS with data-driven priors is most accurate. Data-driven priors do better than uniform priors.

   (b) *What to do if a test example satisfies no rules of any class description?"* Here the result is that it is better to use partial clauses (which use the notion of "nearly satisfied clauses") than to just guess the most frequently occurring class in the training data.

   (c) *What to do if more than one rule from a class is satisfied by the test example?"* Two methods were compared: maximizing and taking the product (of LS's). It was shown that maximizing did best overall. That is, within one class description, one should just use the reliability of the most reliable satisfied rule. However, see the results of Section 7.4 which show that for very noisy data sets, it is better to combine evidence from *all* satisfied rules.

2. **Small disjuncts:** A good way of measuring the *magnitude* of the small disjuncts problem is to compare the error distribution graph (Figure 5.1) to the diagonal distribution graph. The main result that was presented in this area was that the one-per-class approach, together with reliability estimates reduces the small disjuncts problem as compared to FOIL. This is the first work in this area which has been successful at reducing the small disjuncts problem *and* improving overall error rate.

3. **Noise-tolerant learning:**

   (a) *More accurate than FOIL:* HYDRA's one-per-class plus reliability estimates' method proved to be more accurate (and hence, more noise-tolerant) than FOIL on many noisy data sets (Tables 6.1 and 6.2). FOIL was slightly more accurate on some noise-free data sets but that is not a problem since the focus of this research is to learn accurately on noisy, "real-world" data sets.

   (b) *Equally or more accurate than REP:* As Table 6.4 shows, the one-per-class plus reliability estimates approach is more accurate than Reduced-Error Pruning (REP, Quinlan, (1987); Brunk & Pazzani, (1991)) on small training sets. For large training sets, the algorithms are equally accurate but HYDRA is much less computationally expensive than Reduced-Error Pruning.

   (c) *One-per-class approach helps in noisy data sets:* The one-per-class approach was compared to the approach used in FOIL in which a concept description for just a single class is learned. Comparisons on noisy 2-class data sets showed that the one-per-class approach was better overall because it was able to cut down significantly on the number of errors of commission.

4. **The multiple models approach:**

   (a) *"Are multiple models really needed?"* The large number of average gain ties in some domains suggest that even with increased search, it may still be necessary to retain multiple models for classification. That is, the multiple models approach is better not just because it does more search but also because it retains many models for classification. However, more research is needed on this question.

   (b) *Comparison of model generation methods:* Bagging and Stochastic search did equally well overall and both were significantly better than the Partition method. However, Bagging did significantly better than Stochastic search on very noisy data sets unless the sample sizes were small. Stochastic Search did significantly better than Bagging on artificial, noise-free data sets. Bagging falls apart quickly if the sample size is small.

(c) *Comparison of evidence-combination methods:* Likelihood Combination did best overall, but only by a small margin over Uniform Voting. This suggests that the reliability estimates being used by all the methods except Uniform Voting need to be improved. Distribution Summation was the best on very noisy data sets. Bayesian Combination which has been shown to work with hundreds of models (Buntine, 1990; Oliver & Hand, 1995) did not do well with just eleven models.

5. **Error correlation and error reduction:**

   (a) *Modeling the degree of error reduction:* The main result of this work is experimental evidence that the degree of error reduction is substantially linearly correlated with the degree to which models in the ensemble make errors in an uncorrelated manner. This is the first experimental validation of the widely-believed hypothesis that making errors in a less correlated manner leads to greater error reduction. The thesis presents research that shows the link between amount of error reduction and uncorrelatedness of errors holds for rule-set models, decision trees and neural networks.

   (b) *Gain ties:* Gain ties offer a way of explaining why it is possible to learn models that make uncorrelated errors in one domain while this is not possible (or at least, realized) for other domains. The main result is that for data sets in which there are many gain ties, error reduction is quite large. However, although many gain ties seem to be sufficient to get large error reduction they are not necessary - large error reduction is obtained on the Tic-tac-toe domain even though there are not many gain ties in that domain. More research needs to be done here since the gain ties measure is fooled by duplicate definitions and has some other problems.

   (c) *Negatively-correlated errors:* Even though our finding is that the degree of error reduction is related to the degree of (positive) correlatedness of errors, we present arguments that the goal should not be to learn models that make completely uncorrelated errors. In fact, it is better to make negatively-correlated errors. This contrasts with previous beliefs that the goal is to learn models that make errors in a completely independent (uncorrelated) manner.

   (d) *Effect of domain characteristics:* This research is the first time that the effect of domain characteristics on the efficacy of the multiple models approach has been studied. The results are that when there are irrelevant attributes, the multiple models method is able to do much better than the single model method. However, when there is noise, the amount of information in the data is limited, so as the noise level increases, the advantage afforded by the multiple models method is eroded. Taken together, these results imply that the multiple models method is better able to "milk" as much out of the data as possible when there is something to be milked (i.e. the noise level is not too high).

9.1 Contributions of this research

The research presented in this dissertation has made contributions both by supplying new algorithms (for noise-tolerant relational learning) and by providing a better understanding of existing algorithms (for learning multiple models). The research has made contributions in three areas in machine learning - learning multiple models, learning relational rules from noisy data and reducing the error of small disjuncts. Each of these contributions is now discussed and the role of each piece of research within the previous research in that area is described.

1. **Multiple models:** The research provides experimental confirmation of the hypothesis that ensembles that consist of models that make their errors in a less correlated manner will reduce error more than ensembles whose models make errors in a more correlated manner. This experimental confirmation is provided for rule-set models, decision trees and artificial neural networks. The work is also the first to study the *degree* of error reduction. Previous work did not provide a precise definition of the degree of correlatedness and there was no experimental evidence that the *degree* of error reduction is linked to the degree to which the models make uncorrelated errors. The work is also the first to present multiple models results on data sets for which there are striking error reductions up to factors of seven (Wine data set). These data sets highlight the need to understand the degree of error reduction.

   The research also presents a reason why it is possible to learn less correlated models in some data sets but not in others. This involves the notion of "gain ties." The research shows that error is reduced by large factors in data sets in which gain ties are common. The only previous work that addresses the question of when/how the multiple models approach works is that of Breiman (Breiman, 1994). However that work focuses on learning-algorithm properties that enable error reduction whereas the work here focuses on data set properties. In addition, Breiman's work does not model the degree of error reduction.

   Another contribution to the research in the multiple models area is the observation that one should aim to learn ensembles whose models make errors in a dependent but negatively correlated manner. Previously, it had been believed that the best course of action was to learn models that made errors in an uncorrelated manner (independently).

Finally, the research also contributes to our understanding of how effective the multiple models approach is in dealing with noisy data and irrelevant attributes. There has been no previous study of the effect of domain characteristics on the degree of error reduction afforded by the multiple models approach. Perhaps counter to intuition, the results show that the multiple models approach is most effective at lower noise levels. At higher noise levels, there is not much information in the training data. The results show that at higher noise levels, the models make more *correlated* errors and the distribution of errors changes so the proportion of test examples incorrectly classified by all models increases - that is, some of the examples become difficult for all the models. However at lower noise levels, the multiple models approach is able to "milk" the information in the training data better than the single models approach. The results also show that irrelevant attributes are an obstacle to the single model algorithm but an opportunity for the multiple models algorithm. The multiple models method gets greater error reduction as the number of irrelevant attributes is increased. However, beyond some domain-specific point, as too many irrelevant attributes are introduced, both the single and multiple models methods become ineffective.

2. **Learning probabilistic relational rules from noisy data:** Another contribution of this research is the presentation of a novel method for noise-tolerant learning of relational rules. The method, as embodied in the system HYDRA, is based on learning a separate class description for each class and attaching reliability estimates to each learned (relational) rule. Such rules are called probabilistic, relational rules. This method is a contribution since it is more accurate than two previous methods for noise-tolerant relational learning (Reduced-Error Pruning (Quinlan, 1987; Brunk & Pazzani, 1991) and Minimum-Description Length as used in FOIL (Quinlan, 1990)) from small training sets. Additionally, for large training sets, it is as accurate as Reduced-Error Pruning but much more time efficient. The method is also shown to be more accurate on data sets containing attribute noise than mFOIL (Dzeroski & Bratko, 1992). mFOIL is a method for noise-tolerant learning of relational rules.

3. **Small disjuncts:** In the area of research into rule-induction, a new method for improving the error rate of a particular form of rule (the small disjunct) is provided. Previous work by Holte *et al.* (1989) into the small disjuncts problem was only successful at reducing the error rate of small disjuncts at the expense of increasing error rate of large disjuncts. The overall error rate did not decrease. However, the method of learning a separate description per class and using reliability weights presented here is able to reduce the overall error rate *and* the error rate of the small disjuncts. There have also been two other studies of small disjuncts (Danyluk & Provost, 1993; Weiss, 1995) but although these studies improve our understanding of the small disjuncts problem, they do not provide a method for reducing the problem.

9.2 Limitations and future work

This section discusses the limitations and potential for future work in each of the following three areas in turn: multiple models, learning probabilistic relational rules from noisy data and dealing with the small disjuncts problem.

1. **Multiple models:** Perhaps the greatest limitation of the work in multiple models presented in this thesis is that the "gain ties" area needs work. In particular, the gain ties metric is fooled by multiple definitions of attributes. One way to get around this would be to look at the extensions of the tying candidates - if the extensions were the same it would mean that the two attributes did not really represent different alternatives. However, towards the end of a rule, when there are only a few training examples left, many attributes that have identical extensions on the training data may behave quite differently on test data. Thus, identical extensions which cover hundreds of examples imply duplicate definitions and should not be counted as a tie, whereas identical extensions over small numbers of rules should be counted as a tie. The real question is: "Given that these two candidates have identical extensions of n examples, what is the probability that they will behave differently on test data?" The measure should also count "near gain ties" where "near" can be defined with respect to Fisher's exact test (Finney et al., 1963).

   Another limitation of the work on multiple models is that the measure of correlatedness, ϕ_e, is just a pair-wise measure. More sophisticated measures such as those found in factor analysis might better characterize the ability of the ensemble as a whole to reduce error.

   The stochastic method for generating multiple models uses the β parameter to construct the set of candidates from which it must stochastically select one candidate. It would be better to use Fisher's exact test at some confidence threshold to decide how many candidates to select. The results in the dissertation show that the single model method is more accurate than the multiple models implementation for noise-free sufficient sample data sets (noise-free KRK data sets). For such data sets, the best strategy is to always go with the candidate that has the highest gain as the next highest gain is too far below the best gain. Hopefully, Fisher's exact test will only select a single candidate in such data sets thus simulating the single model approach when it is appropriate.

   A pressing question for future work in the area of multiple models research is the following:

   > Are multiple models able to do better than a single model because they involve more search or is it because multiple models are retained for classification?

To test this empirically, one would need to construct a learning algorithm that does as much search as the multiple models method but only produces a single model. That method would only differ from the multiple models method in one variable: the number of models retained for classification. Then the two methods would be compared on many data sets and if on some group of data sets, the multiple models method did significantly better then one could definitively conclude that the multiple models method does not do better simply because of more search - the ability to retain multiple models for classification is crucial (at least for that group of data sets).

Future work should also look at automated methods to decide how many models to learn. In relatively noise-free, sufficient sample data sets (e.g. noise-free King-Rook-King with 200 training examples), it may be sufficient just to learn a single model. However, in data sets with an abundance of weakly-informative or irrelevant attributes (DNA, Splice...) the method should learn many models.

2. **Noise-tolerant learning of relational rules:**

   Perhaps the main limitation of the one-per-class approach with reliability estimates as embodied in HYDRA is that the rules are overfitting the data. Although the one-per-class approach and reliability estimates compensate for this overfitting, even better results may be possible if one were to put more emphasis on learning less overfitted rules. Since Reduced-Error Pruning (Quinlan, 1987) is so expensive, a form of pre-pruning is recommended. In particular, pre-pruning based on Fisher's exact test (or equivalently, the hyper-geometric gain metric, Martin, (1995)) is recommended. The MDL-based pre-pruning of Quinlan (1987) is not recommended since it requires an ad-hoc parameter and the chi-squared statistical test or variants of it [1] are also not recommended since they makes assumptions that fail precisely when they are most needed - towards the end of a rule when the number of examples is small.

   Another limitation of HYDRA is that for some domains, it may be easy to learn a class description for one class and very difficult to learn for the complementary class (with respect to the given set of background predicates). If in addition, there is not too much noise in the data set, and the user has instructed FOIL to learn for the easy class, it is probable that FOIL will be more accurate than HYDRA since HYDRA will learn error-prone rules for the difficult class.

   Future work should also address the problem of obtaining reliable reliability estimates. HYDRA makes these estimates from the training data - it would be better to estimate them from a hold-back set. If there is enough data to keep some as hold-back, then everything is fine. If however, there is not, one is forced to decide between two unacceptable alternatives. If the small

[1] E.g. the likelihood ratio statistic (Kalbfleish, 1979) as used in CN2 (Clark & Niblett, 1989).

training set is divided into a learning set and a hold-back set, one will end up with pretty good reliability estimates of poorly learned rules. The rules will be poorly learned since the amount of data which they were learned from is so small. Alternatively, if one does not form a hold-back set, one will get poor reliability estimates but the rules will be better. It would be useful to search for automated methods of deciding when there is enough data for a hold-back set.

3. **Small disjuncts:** HYDRA has been shown to significantly reduce the error contribution of small disjuncts as compared to FOIL. However, the small disjuncts problem still remains - although reduced in degree. More work needs to be done to try and reduce error from such disjuncts. In particular, new algorithms rather than studies are needed. Or, one could make a contribution by showing that it is not possible to further reduce the error of small disjuncts without increasing error for the large disjuncts.

9.3 Summary

I have presented an in-depth analysis of why the multiple models approach works (by making uncorrelated errors) and why the approach works better in some data sets than in others - it works better in data sets with many gain ties. This is also the first work to provide a model of the degree of error reduction due to the use of multiple models.

The work also explains why noise inhibits the ability of the multiple models approach to reduce error and why moderate numbers of irrelevant attributes provide an opportunity for the multiple models approach. Finally, the research provides arguments showing that it is better to learn ensembles whose models make errors in a dependent but negatively-correlated manner rather than to learn ensembles whose models make errors in a completely uncorrelated manner.

The work on noise-tolerant learning of rules with probabilistic attachments (reliability estimates) shows that the one-per-class approach together with reliability estimates as used in HYDRA is able to equal or beat three algorithms for noise-tolerant learning of relational rules (on the tested domains). For small sample sizes, HYDRA is more accurate than Reduced-Error Pruning (REP) and equals it for large sample sizes. However, HYDRA is much less computationally expensive than Reduced-Error Pruning for large sample sizes. HYDRA is also more accurate than Quinlan's implementation of MDL for FOIL since Brunk & Pazzani (Brunk & Pazzani, 1991) already show that REP is more accurate than Quinlan's MDL. The results refute the negative one-per-class results of Dietterich & Bakiri (1995). The crucial difference between the experiments presented here and those in Dietterich & Bakiri is that they break conflicts randomly. By resolving conflicts in a more informed manner through the use of reliability estimates, HYDRA is able to do better than many algorithms at noise-tolerant learning of probabilistic relational rules.

Finally, the work is the first in which an algorithm is successful at reducing the error rates of small disjuncts and simultaneously reducing the overall error rate of the algorithm. Previous work (Holte et al., 1989) could only reduce the error rates of the small disjuncts at the expense of increasing the error rates of large disjuncts. Other work on the small disjuncts problem (Weiss (1995); Danyluk & Provost, (1993); Quinlan (1991)) has studied the problem but has not provided or suggested a method for solving it.

References

Aha, D. (1990). *A Study of Instance-Based Algorithms for Supervised Learning Tasks*. PhD thesis, Dept. of Information and Computer Sciences, University of California, Irvine.

Ali, K. & Pazzani, M. (1992). Reducing the small disjuncts problem by learning probabilistic concept descriptions. In T. Petsche, S. Hanson, & J. Shavlik (Eds.), *Computational Learning Theory and Natural Learning Systems, Vol. 3*. Cambridge, Massachusetts: MIT Press.

Ali, K. & Pazzani, M. (1993). HYDRA: A noise-tolerant relational concept learning algorithm. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, France. Morgan Kaufmann.

Ali, K. & Pazzani, M. (1995a). Error reduction through learning multiple descriptions. Technical Report UCI TR #95-39, University of California, Irvine, Dept. of Information and Computer Sciences.

Ali, K. & Pazzani, M. (1995b). HYDRA-MM: Learning multiple descriptions to improve classification accuracy. *International Journal on Artificial Intelligence Tools*, *4, 1 & 2*, 115–133.

Ali, K. & Pazzani, M. (1995c). Learning multiple relational rule-based models. In D. Fisher & H. Lenz (Eds.), *Learning from Data: Artificial Intelligence and Statistics, Vol. 5*. Fort Lauderdale, FL: Springer-Verlag.

Ali, K. & Pazzani, M. (1995d). On the link between error correlation and error reduction in decision tree ensembles. Technical Report UCI TR #95-38, University of California, Irvine, Dept. of Information and Computer Sciences.

Ali, K. & Pazzani, M. (submitted). Error reduction through learning multiple descriptions. *Machine Learning*, *?*

Angluin, D. & Laird, P. (1988). Learning from noisy examples. *Machine Learning*, *2, 4*, 343–370.

Bacchus, F. (1990). *Representing and Reasoning With Probabilistic Knowledge*. MIT Press.

Brazdil, P. & Torgo, L. (1990). Knowledge acquisition via knowledge integration. In *Current Trends in Knowledge Acquisition*. IOS Press.

Breiman, L. (1994). Heuristics of instability in model selection. Technical Report TR. 416, Statistics Department, University of California at Berkeley.

Breiman, L. (in press). Bagging predictors. *Machine Learning, ?*

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and Regression Trees*. Belmont: Wadsworth.

Brunk, C. & Pazzani, M. (1991). An investigation of noise-tolerant relational concept learning. In *Proceedings of the Eighth International Workshop on Machine Learning*, Evanston, IL. Morgan Kaufmann.

Buchanan, B. & Shortliffe, E. (1984). *Rule-based expert systems : the MYCIN experiments of the Stanford Heuristic Programming Project*. Addison Wesley.

Buntine, W. (1990). *A Theory of Learning Classification Rules*. PhD thesis, School of Computing Science, University of Technology.

Buntine, W. (1991). Classifiers: A theoretical and empirical study. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Sydney, Australia. Morgan Kaufmann.

Cestnik, B. & Bratko, I. (1991). On estimating probabilities in tree pruning. In *Machine Learning - EWSL-91*. Pitman Press.

Chaitin, G. (1987). *Algorithmic information theory*. Cambridge University Press.

Chan, P. & Stolfo, S. (1995). A comparative evaluation of voting and meta-learning on partitioned data. In *Machine Learning: Proceedings of the Twelfth International Conference*, Tahoe City, CA. Morgan Kaufmann.

Cheeseman, P. (1986). Probabilistic versus fuzzy reasoning. In L. K. . J. L. (Ed.s) (Ed.), *Uncertainty in Artificial Intelligence*, volume 2. Elsevier Science Publishers B.V.

Clark, P. & Boswell, R. (1991). Rule induction with CN2: Some recent improvements. In *Proceedings of the European Working Session on Learning, 1991*. Pitman.

Clark, P. & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, *3*, 261–283.

Clearwater, S. & Provost, F. (1990). RL4: A tool for knowledge-based induction. In *Proceedings of the Second International IEEE Conference on Tools for Artificial Intelligence*. IEEE C.S. Press.

Cohen, W. (1992). Compiling prior knowledge into an explicit bias. In *Machine Learning: Proceedings of the Ninth International Workshop*, Aberdeen, U.K. Morgan Kaufmann.

Danyluk, A. & Provost, F. (1993). Small disjuncts in action: Learning to diagnose errors in the local loop of the telephone network. In *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, MA. Morgan Kaufmann.

DeGroot, M. (1986). *Probability and Statistics*. Reading, MA: Addison-Wesley.

Dietterich, T. & Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, *2*, 263–286.

Drucker, H., Cortes, C., Jackel, L., LeCun, Y., & Vapnik, V. (1994). Boosting and other machine learning algorithms. In *Machine Learning: Proceedings of the Eleventh International Conference*, New Brunswick, NJ. Morgan Kaufmann.

Drucker, H., Schapire, R., & Simard, P. (1993). Improving performance in neural networks using a boosting algorithm. *Neural Information Processing Systems*, *5*, 42–49.

Duda, R., Gaschnig, J., & Hart, P. (1979). Model design in the PROSPECTOR consultant system for mineral exploration. In D. Michie (Ed.), *Expert systems in the micro-electronic age*. Edinburgh, England: Edinburgh University Press.

Dzeroski, S. & Bratko, I. (1992). Handling noise in inductive logic programming. In *Proceedings of the International Workshop on Inductive Logic Programming*, Tokyo, Japan. ICOT Press.

Finney, D., Latscha, R., Bennett, B., & Hsu, P. (1963). *Tables for Testing Significance in a 2 x 2 Contigency Table*. Cambridge University Press.

Gams, M. (1989). New measurements highlight the importance of redundant knowledge. In *European Working Session on Learning (4th: 1989)*, Montpeiller, France. Pitman.

Good, I. (1950). *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. MIT Press.

Hamming, R. (1986). *Coding and information theory*. Prentice Hall.

Hansen, L. & Salamon, P. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *12, 10*, 993–1001.

Holte, R., Acker, L., & Porter, B. (1989). Concept learning and the problem of small disjuncts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI. Morgan Kaufmann.

Hunt, E., Marin, J., & Stone, P. (1966). *Experiments in Induction*. New York: Academic Press.

Kalbfleish, J. (1979). *Probability and Statistical Inference*. Springer-Verlag.

Kong, E. & Dietterich, T. (1995). Error-correcting output coding corrects bias and variance. In *Machine Learning: Proceedings of the 12th International Conference*, Tahoe City, CA. Morgan Kaufmann.

Kononenko, I. & Kovacic, M. (1992). Learning as optimization: Stochastic generation of multiple knowledge. In *Machine Learning: Proceedings of the Ninth International Workshop*, Aberdeen, Scotland. Morgan Kaufmann.

Kovacic, M. (1994). MILP - a stochastic approach to inductive logic programming. In *Proceedings of the Fourth International Workshop on Inductive Logic Programming*, Bad Honnef/Bonn, Germany. GMD Press.

Kruskal, W. & Tanur, J. (1978). *International encyclopedia of statistics.* New York, NY: Free Press.

Kwok, S. & Carter, C. (1990). Multiple decision trees. *Uncertainty in Artificial Intelligence*, *4*, 327–335.

Lloyd, J. (1984). *Foundations of Logic Programming.* Springer-Verlag.

Madigan, D. & York, J. (1993). Bayesian graphical models for discrete data. Technical Report UW-93-259, Statistics Department, University of Washington.

Martin, J. (1995). Small sample statistics for classification error rates. Technical Report UCI TR #95-1, Department of Information and Computer Sciences, University of California, Irvine.

Michalski, R. (1986). The AQ15 inductive learning system: An overview and experiments. In *Proceedings of the International Meeting in Advances in Learing*, Les Arcs, France.

Michalski, R. & Stepp, R. (1983). Learning from observation: Conceptual clustering. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach.* Tioga Publishing Co.

Muggleton, S., Bain, M., Hayes-Michie, J., & Michie, D. (1989). An experimental comparison of human and machine-learning formalisms. In *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, NY. Morgan Kaufmann.

Muggleton, S., Srinivasan, A., & Bain, M. (1992). Compression, significance and accuracy. In *Machine Learning: Proceedings of the Ninth International Workshop*, Aberdeen, Scotland. Morgan Kaufmann.

Murphy, P. & Aha, D. (1992). University of California, Irvine, repository of machine learning databases. A machine-readable data repository.

Nilsson, N. (1986). Probabilistic logic. *Artificial Intelligence*, *28*, 71–87.

Nisbett, R. & Ross, L. (1980). *Human inference : strategies and shortcomings of social judgment.* Prentice-Hall.

Oliver, J. & Hand, D. (1995). On pruning and averaging decision trees. In *Proceedings of the Twelfth International Conference on Machine Learning*, Tahoe City, CA. Morgan Kaufmann.

Pazzani, M. & Brunk, C. (1991). Detecting and correcting errors in rule-based expert systems: an integration of empirical and explanation-based learning. *Knowledge Acquisition*, *3*, 157–173.

Pazzani, M., Brunk, C., & Silverstein, G. (1991). A knowledge-intensive approach to learning relational concepts. In *Machine Learning: Proceedings of the Eighth International Workshop (ML91)*, Ithaca, NY. Morgan Kaufmann.

Pazzani, M. & Kibler, D. (1991). The utility of knowledge in inductive learning. *Machine Learning*, *9, 1*, 57–94.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann.

Perrone, M. (1993). *Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure Optimization.* PhD thesis, Department of Physics, Brown University.

Perrone, M. & Cooper, L. (1993). When networks disagree: Ensemble methods for hybrid neural networks. In R. E. Mammone (Ed.), *Neural Networks for Speech and Image Processing.* Chapman Hall.

Quinlan, J. & Rivest, R. (1989). Inferring decision trees using the minimum decision length principle. *Information and Computation*, *80*, 227–248.

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

Quinlan, R. (1987). Simplifying decision trees. *International Journal of Man-Machine Studies*, *27*, 221–234.

Quinlan, R. (1990). Learning logical definitions from relations. *Machine Learning*, *5, 3*, 239–266.

Quinlan, R. (1991). Improved estimates for the accuracy of small disjuncts. *Machine Learning*, *6, 1*, 93–98.

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

Ripley, B. (1987). *Stochastic Simulation.* John Wiley & Sons.

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

Rumelhart, D. & McClelland, J. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition.* MIT Press.

Schaffer, C. (1994). Cross-validation, stacking and bi-level stacking: Meta-methods for classification learning. In W. Cheeseman, P. & Olford (Ed.), *Selected Models from Data.* Springer-Verlag.

Segal, R. & Etzioni, O. (1994). Learning decision lists using homogeneous rules. In *AAAI 94: Proceedings, Twelfth National Conference on Artificial Intelligence*, Seattle, WA. Morgan Kaufmann.

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

Shannon, C. & Weaver, W. (1949). *The mathematical theory of communication.* University of Illinois Press.

Smyth, P. & Goodman, R. (1992). Rule induction using information theory. In G. Piatetsky-Shapiro (Ed.), *Knowledge Discovery in Databases.* Menlo Park, CA: AAAI Press, MIT Press.

Smyth, P., Goodman, R., & Higgins, C. (1990). A hybrid rule-based/Bayesian classifier. In *Proceedings of the 1990 European Conference on Artificial Intelligence*, London, UK. Pitman.

Solomonoff, R. (1964). A formal theory of inductive inference. *Information and Control*, *7*, 376–388.

Weiss, G. (1995). The impact of small disjuncts on learning. In *Proceedings of the Twelfth International Conference on Machine Learning*, Tahoe City, CA. Morgan Kaufmann.

Weiss, S. & Kulikowski, C. (1991). *Computer Systems That Learn.* Morgan Kaufmann.

Wogulis, J. (1994). *An Approach to Repairing and Evaluating First-Order Theories Containing Multiple Concepts and Negation.* PhD thesis, Department of Information and Computer Science, University of California, Irvine.

Wolpert, D. (1992). Stacked generalization. *Neural Networks*, *5*, 241–259.

Zadeh, L. (1980). Inference in fuzzy logic. In *IEEE 10th Annual Synmposium on Multiple-valued Logic.*