ON THE DEVELOPMENT OF INDUCTIVE LEARNING ALGORITHMS: GENERATING FLEXIBLE AND ADAPTABLE CONCEPT REPRESENTATIONS

BY

RICARDO VILALTA

Ingen., Instituto Tecnológico y de Estudios Superiores de Monterrey, 1989
M.S., University of Illinois at Urbana-Champaign, 1995

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1998

Urbana, Illinois

ON THE DEVELOPMENT OF INDUCTIVE LEARNING ALGORITHMS: GENERATING FLEXIBLE AND ADAPTABLE CONCEPT REPRESENTATIONS

Ricardo Vilalta, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1998
Prof. Larry Rendell, Advisor

Vast amount of research in machine learning has focused on creating new algorithms stemming from refinements to existing learning models (e.g., neural nets, decision trees, instance-based learners, rule-based systems, bayesian estimators). This is convenient when a model $\mathcal{M}$ already exists exhibiting satisfactory performance over the class of domains of study $\mathcal{S}$. It may occur, however, that no model over $\mathcal{S}$ performs adequately; the definition of a new learning model is then necessary.

The construction of a new learning model cannot rely on assumptions common to a process based on refinements alone: that a global evaluation metric is enough to guide the development process (e.g., a comparison based on predictive accuracy alone), and that the strategy of the model is always correct (inherent limitations in the strategy may exist). When a model is defined from scratch, it is recommended to adopt a functional view. This means we must first define the set of functionalities the new mechanism will perform (e.g., to cope with feature and class noise, to generate hypotheses amenable to interpretation and analysis, to elucidate feature interaction effectively, etc.). After implementation, a functional decomposition analysis can assess how well the different components in the mechanism are able to carry out such functionalities.

In the first part of this thesis I outline basic steps for the development of new learning models. The outline conveys relevant information to the developer: 1) that the set of goal functionalities must be made clear from the beginning, and that the design and implementation must be oriented to accomplish such functionalities; 2) much can be learned from a functional decomposition analysis; by analyzing each component individually we can detect possible deficiencies; 3) it is vital to propose a design in which all components are made explicit, and thus amenable to modification or replacement.

To provide evidence supporting the importance of a functional view in the design of new learning models, I describe two experimental case studies. These studies serve to support the following ideas: that separating an algorithm into its constituent components is helpful to identify the individual contribution of each component; and that components of different nature can be integrated to achieve a desired functionality.

The second part of this thesis deals with the design and implementation of a new learning algorithm: *HCL* (Hierarchical Concept Learner). This part has two main goals: to put into practice the basic steps for the development of new algorithms, and to assess the performance of *HCL* in both artificial and real-world domains. The general strategy behind *HCL* is to construct consecutive layers in a hierarchy of intermediate concepts (i.e., partial hypotheses); the bottom of the structure accommodates original features whereas the top of the structure contains the final hypothesis. The hierarchical structure is intended to capture the intermediate steps necessary to bridge the gap between the original features and the (possibly complex) target concept. *HCL* is specifically designed for those class of domains where lack of expertise in crafting new features results in a low-level representation: information is missing to conform a representation explicitly showing the interactions among primitive features. A low-level feature representation makes learning hard, because it produces high variation in the distribution of examples along the instance space.

HCL achieves two functionalities: 1) flexibility in the representation, by increasing the complexity of the hypothesis comprised at each hierarchical layer, and 2) adaptability in the search for different representations, to know when to stop adding more layers in top of the hierarchical structure. Adaptability allows *HCL* to adjust the complexity of the representation by building few hierarchical levels when the concept is simple, and by increasing the number of levels as the difficulty of the concept grows higher. *HCL* is assessed experimentally using both artificial and real-world domains. Results show how *HCL* outperforms other models significantly when many intermediate concepts lie between the primitive features and the target concept.

To Cristabel Ortega

Acknowledgments

Special thanks go to my advisor, Prof. Larry Rendell, for his most valuable feedback. I will always be grateful to him for keeping my motivation high during the final stage of my research.

Thanks go the members of the Inductive Learning Group at the University of Illinois for their fruitful comments and suggestions. I am particularly grateful to Gunnar Blix for his invaluable assistance. Thanks go to secretary Sharon Collins who attended all my requests promptly and efficiently.

Any long-term task cannot be accomplished without the encouragement that friendship provides. I am thankful to all my friends at the University of Illinois and at the Lincoln-Green residence for their continuous help and support.

To my parents Ricardo and María Luisa, my brother Javier, and my sisters Mónica and Marisa, I am highly indebted: our union has provided me with all the peace and happiness necessary to strive for high goals.

Most important of all I am grateful to Cristabel Ortega, for her sincere love and affection gave me all the necessary strength to complete this job.

This work was supported in part by a grant from CONACyT (Consejo Nacional de Ciencia y Tecnología), México.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

An important characteristic of an intelligent agent is its ability to improve performance through experience. Faced with new circumstances resembling past events, the agent is expected to *learn*, and to modify its behavior to progressively accomplish its own goals more effectively. In this thesis we shall look into the mechanization of the learning process by computer algorithms, which has led the past decades to vast amount of research in *machine learning*. The variety of different aspects that play a key role during learning makes machine learning multidisciplinary by nature; several areas of science have been influential to the field, e.g., artificial intelligence, statistics, information theory, psychology, philosophy, neurobiology, pattern recognition and analysis. Central to machine learning is the *development of learning algorithms*, where different views of how a learning engine is assembled become amenable to assessment and refinement. As defined by Langley (1996):

> 'A central thread that hold these approaches together is a concern with the development, understanding, and evaluation of learning algorithms. If machine learning is a science, then it is clearly a science of algorithms.'

Learning algorithms in machine learning can be grouped into different types, according to the goal set by the intelligent agent, e.g., classification and prediction (supervised learning), data clustering and pattern discovery (unsupervised learning), accumulation and abstraction of past cases or events (analogical learning), improved response to new circumstances (reinforcement learning), etc. This thesis explores algorithms that belong to *supervised learning*[1] (Michalski, 1983), where the intelligent agent is presented with a set of examples (i.e. objects, situations, etc.) that belong to

[1] In what follows, and for short notation, the terms 'learning algorithm' or 'algorithm' will both mean 'supervised learning algorithm'.

different categories or classes (i.e., that are classified by an underlying target concept $\mathcal{C}$ into several classes). The goal of the learning process is to approximate the function (i.e., to approximate the target concept $\mathcal{C}$) that correctly predicts the class of all — seen and unseen— examples. We shall explore the case where no knowledge other than that provided by the examples is available (i.e., no domain knowledge exists); learning will proceed from the information gained by the available examples only.

1.1 The Development Process

Throughout its history, research in supervised learning has produced a set of *learning models*, e.g., neural nets (Rumelhart, Hinton, & Williams, 1986), decision trees (Quinlan, 1986; Breiman, Friedman, Olshen, & Stone, 1984), instance-based learners (Aha, Kibler, & Albert, 1991), rule-based systems (Quinlan, 1987; Clark & Niblett, 1989), bayesian estimators (Anderson & Matessa, 1992), and many others. Each model represents a different approach to the mechanization of learning.

The number of learning algorithms stemming from current models abound[2], and their application in real-world domains has already proved effective (Langley & Simon, 1995). Nevertheless, the problem of developing new learning algorithms remains difficult. The researcher interested in such task has two alternatives[3]:

1. Refine an existing model, or

2. Construct a new model.

The selection of one alternative or the other depends on the class of domains $\mathcal{S}$ under analysis. If a model $\mathcal{M}$ exists exhibiting satisfactory performance over $\mathcal{S}$, the development phase can be based on the refinement $\mathcal{M}'$ of $\mathcal{M}$, as long as $\mathcal{M}'$ guarantees some measurable improvement. Here, alternative 1 is to be preferred. Other cases exist for which no model over $\mathcal{S}$ is adequate; the bias (Mitchell, 1980) inherent in each model produces results that fall below the expectations. Furthermore, if the

[2] A learning algorithm can be either a model, called a basic algorithm (Section 3.1), or a variation on a model.

[3] A third alternative could be to combine existing models, which in this thesis is considered as part of alternative 1, as will be explained later on.

purpose of the researcher is to create a learning algorithm showing good performance over a large set of structured domains (explained formally in Sections 2.3 and 2.4), more than simple refinements are required: good performance may be limited to only a small class of domains, which creates the need for a new approach to learning. Here alternative 2 is to be preferred.

With regard to the construction of new learning algorithms, most research in machine learning has focused on alternative 1, through the study of techniques that improve existing models. In contrast, this thesis explores the steps necessary for the definition of new models (alternative 2). A major goal is to promote the construction of algorithms that are well suited to the class of domains of study, particularly when no existing approach seems to provide the right concept approximations. The next sections elaborate on the differences between the two alternatives.

1.2 Refining Existing Models

One alternative to the development of a new learning algorithm is to improve an existing *learning model*. In this case, the novelty of the algorithm lies in the improvement alone, with the underlying assumption that the model's basic strategy is adequate for the class of domains under study $\mathcal{S}$. The notion of a strategy that is essential to a model is elaborated in Section 3.1. What is important to note here is that the new algorithm proceeds from modifications that leave the model's basic strategy intact. The degree of success greatly depends on how well the bias attached to the selected strategy initially matches $\mathcal{S}$. This alternative has some disadvantages, and often carries erroneous assumptions, as explained next.

Disregarding the Underlying Strategy of the Model

In a typical scenario, learning algorithm $L_{A'}$ is said to be superior to L_A because the modification introduced in $L_{A'}$ conveys a more suitable form of bias. As an example, suppose a researcher decides that a neural net, $\mathrm{NN_{bp}}$, trained via backpropagation (Rumelhart et al., 1986), appears to provide the best predictive accuracy over a certain class of domains S. The researcher decides to modify $\mathrm{NN_{bp}}$'s mechanism by using a different sigmoid function (i.e., squashing function), eventually obtaining a significant increase Δ_{acc} in predictive accuracy (measured via some re-sampling technique, e.g., n-fold cross-validation). The researcher then claims to have produced

a better algorithm $\mathrm{NN_{bp}}'$ over S. In this example, the reason for the success of $\mathrm{NN_{bp}}'$ is not only unclear (no analysis is provided explaining the reason for the success of the proposed improvement), but also a strong assumption is made that the model —in this case, a neural net trained via backpropagation— needs not be studied or analyzed. The modification is gladly accepted as long as Δ_{acc} results in a positive value.

In the example above, using a different sigmoid function leaves the basic model unaltered: linear combinations of weighted inputs still form the basis to fire each neural node; lower layer nodes serve as input to upper layer nodes; error minimization (on the training set) is achieved by conducting a hill-climbing search through a weight space, etc. But, what if a model is limited, lacking certain functionalities?, what guidelines exist to suggest new forms of learning?

Disadvantages and Assumptions of the Traditional Methodology

The example above reveals two common disadvantages surrounding the traditional refinement approach:

1. Using a global evaluation metric to measure the performance of a new learning algorithm cannot help to identify the individual contribution of each component; no casual theory can be obtained to account for the new algorithm's behavior.

2. Improvements based on refinements alone restrict how far the model can go. Inherent limitations may remain attached to the model; refinements can modify the basic algorithm to a certain extent only (Section 3.2).

The disadvantages above share the common assumption that a limit exists on how deep a learning algorithm can be studied. The particular bias (Mitchell, 1980) bestowed on the algorithm is assumed to pervade the entire mechanism; the individual contribution of each learning component becomes indistinguishable. How exactly this bias is defined cannot be explained at a more refined level than the whole algorithm itself. Whatever the effects of the components embedded in the algorithm are, these tend to be averaged altogether.

An example of this assumption is seen on projects experimenting with vast number of learning algorithms to find the class of domains for which each algorithm works

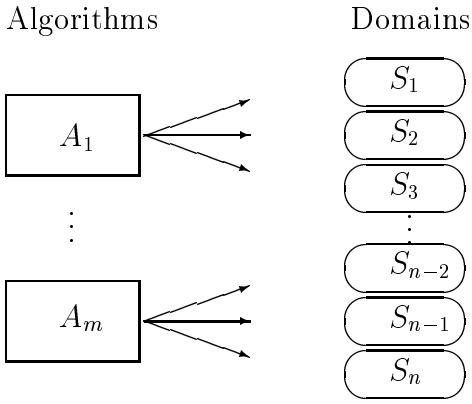


Figure 1.1: The goal of some large-scale projects is to identify the class of domains on which each algorithm performs best.

best (Michie, 1994; Weiss & Kulikowski, 1990). An implicit goal is to relate an algorithm to the domains on which this algorithm outperforms all other competitors, as shown in Figure 1.1. Results can be used to know what algorithm is to be selected on a particular domain. These projects assume each algorithm cannot be deemed correct or incorrect, but only having the right assumptions on different scenarios.

While several empirical comparisons of different learning algorithms have been reported in the machine learning literature (Fisher & McKusick, 1989; Weiss & Kapouleas, 1989; Dietterich, Hild, & Bakiri, 1990; Wnek & Michalski, 1994), the reasons for the good performance of an algorithm remain elusive. For example, Quinlan (1993) proposes that the difference between a connectionist approach (e.g., neural net) and a symbolic approach (e.g., decision tree, rule-based system) lies in the nature of the domain under study: a connectionist approach is better suited when all features are relevant to the target concept, whereas a symbolic approach works better if only few features relate to the target concept; an empirical study by Schaffer (1993), however, gives no support to such theory. Schaffer experiments with selecting the best of a set of classifiers —using a 10-fold cross validation approach— as the best approximation to the concept being studied. Although the results show that the selected classifier always performs close to the best classifier, the approach ignores the possible deficiencies characterizing each of the algorithms under analysis. Other researchers have applied meta-learning techniques to find rules identifying the domains in which an algorithm is applicable (Gama & Brazdil, 1995; Brazdil, Gama, & Henery, 1994).

In all cases, the goal of the empirical comparison is to draw a match between certain class of domains and the learning algorithm A_{best} that dominates over that class. No clear distinction on what the internal components of A_{best} is made, nor on the contribution of each component during learning. Distinguishing algorithms based on the class of domains where bias is most favorable is like classifying automobiles according to the types of terrain that best match the automobile design. Here nothing is said about the mechanism of the automobile itself: the engine, the internal controls, the safety features, the (possible) missing capabilities. An automobile can hardly be sold if these issues are not brought into consideration.

These studies disregard the degree to which the word *bias* is employed. The bias of an algorithm is the result of the integration of the biases of *all* its components. If we fail to decompose and analyze each of an algorithm's components, then limitations of current models will remain unknown. Studies that look to algorithms as indivisible mechanisms are superficial; no attempt is made to relate the internal mechanism with its performance.

1.3 Constructing a New Model

Design vs. Functional View

Constructing a new model can be seen as a search over a space of possible models. To understand how a model can be constructed it is first necessary to understand the differences between a search over two different model spaces: the design space and the functional space (Bradshaw, 1993a, 1993b). A search over a design space starts with a prototype mechanism. New designs are explored by generating small variations to this mechanism. A global evaluation metric assesses the utility of each design. This kind of search becomes problematic with large design spaces: an enormous amount of effort often must be expended. On the other hand, a search over a functional space complements the design space by studying the functions associated with each component. Rather than testing performance of the whole design, this analysis isolates the different functions required to achieve a certain goal. Each function is identified, built through the combination of components, and tested in isolation. Functional decomposition analysis can guide the design process into more effective paths without exploring useless designs.

Research in machine learning often resembles a search in a design space, where

learning algorithms are produced from model refinements. Proposed changes are not studied in isolation, nor assessed individually. This study relies on the assumption that, when no model is adequate to the class of domains under analysis, more efficient learning mechanisms can be constructed if we consider the functional space. Under this view we must decompose current models (with regard to specific functionalities) to pinpoint specific reasons for their performance; we must consider component parts and their relative strengths and weaknesses. A methodology that characterizes the development process in this way can lead to the definition of new and better models.

What is a Functionality in Induction?

The design of a new learning algorithm assumes the existence of certain goals that will serve as guidelines for the development process. In this thesis, a functionality means an operation we set as a goal for the new algorithm to carry out. It answers the question: what are we striving to attain with the new algorithm? For example, according to the particular domain under study, certain operations may be of greater importance: the ability to deal with feature and class noise, to output interpretable hypotheses, to be effective to elucidate feature interaction due to the presence of highly complex relations, to use a specific concept-language representation, to exploit in a particular way the information given by the training examples, etc.

Functionalities can appear as intermediate or final operations in attaining a goal. A functionality f_i set as a goal can be subdivided into subgoals f_i^1, f_i^2, $\cdots$, f_i^n, each of them set to satisfy specific requirements. The process may continue for each f_i^j until all operational details are clarified. For example, suppose we have decided to construct a learning algorithm that adapts the learning bias dynamically (overall goal). At a more detailed level, we may decide to accomplish that goal by searching for hypotheses in different spaces of representation; here we may recognize the importance of effectively selecting the right space of representation (subgoal), which can be attained by incorporating a metric quantifying the quality of each space according to the domain under study. Independently of the detail with which functionalities are specified, the main idea is to let the design process be guided by exactly those operations we wish the new algorithm to perform. In other words, it is the achievement of certain goals that must guide the design of an algorithm; if the development process is based on refining predefined models, then accomplishing such goals may in some cases become impossible (Section 1.2).

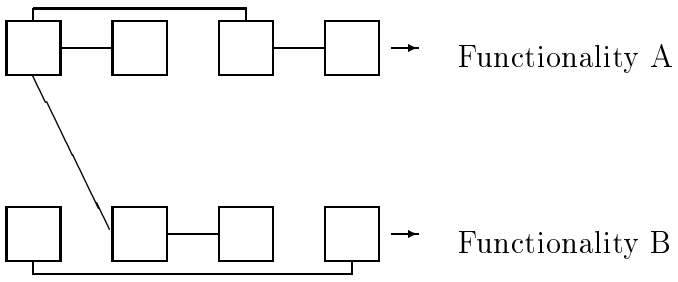


Figure 1.2: Components (blocks) combine to reach goal functionalities, but may also interact across different functionalities.

Functionalities and Components

A particular functionality is achieved through the implementation of one or more learning components. A component is defined here as a sequence of instructions executing a single-purpose operation. For example, adapting the learning bias dynamically may require quantification of the quality of each space of representation; the set of instructions applying a metric to each space is the component implementing the desired functionality. A decision tree inducer elucidates feature interaction (goal) by recursively partitioning the training data to delineate single class regions (i.e., to find disjunctive terms) over the instance space. To achieve that goal, one component (i.e., sequence of instructions) is in charge of searching for the best feature F_{best} to partition the training data; another component actually partitions the data according to the values taken by F_{best}. If a functionality is to improve the quality of the feature set, a component can perform a selection of the most relevant features, etc. As shown in Figure 1.2, each component interacts with other components to reach a certain goal functionality. It may happen that a component designed to achieve certain goal interacts outside the place where its effects are required. In this thesis I will assume a simple relationship in which interactions across different functionalities can be ignored. I believe this simplification will reveal the benefits obtained when a functional space guides the development process.

An Example of the Importance of a Functional Decomposition Analysis

We need a functional decomposition analysis that can clarify the internal mechanism of a learning algorithm; the particular behavior of an algorithm should be related to its functional design. The next example illustrates these ideas. *SWAP1* (Weiss

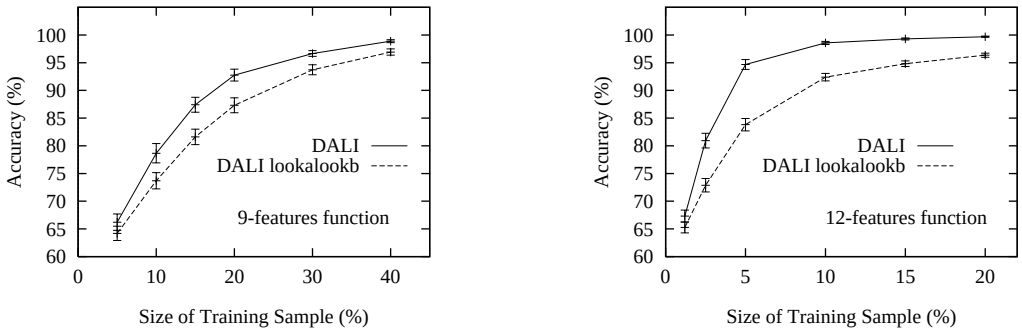


Figure 1.3: Training-set size vs. predictive accuracy for nine and twelve feature functions comparing *DALI* with *DALI*-lookahead-lookback.

& Indurkhya, 1993) is an efficient rule-based separate-and-conquer algorithm. The novelty of the algorithm is claimed by its authors to reside in the search for conjunctive expressions. Each expression serves as the antecedent for an induced rule. The search component *looks ahead and back* to augment/refine a rule. Looking ahead explores the best feature-value to add next; looking back reconsiders feature-values that are already members of the rule. To test the utility of this component, I implemented it in a different scenario: at each node of a decision tree, the search component was employed to generate a rule that would serve as the next splitting function. The resulting system, *DALI*-lookahead-lookback, is compared to *DALI* (Section 5.2), a decision tree learner that carries out, at each tree node, a beam search over the same space of feature-value conjuncts. Learning curves comparing *DALI* with *DALI*-lookahead-lookback for two DNF functions of medium difficulty (in terms of variation, Section 2.4) are shown in Figure 1.3. A significant decrease of predictive accuracy is observed with *SWAP1*'s new search mechanism when compared to *DALI*'s beam search. The lookahead-lookback component does not improve — when compared to a beam search, and in this particular domain— the quality of the splitting functions. But what is the source for *SWAP1*'s good performance? Weiss and Indurkhya (1993) introduce (but downplay) a different component in *SWAP1* where the best hypothesis (i.e., set of rules) according to a 10-fold cross validation is the one selected for classification. To determine the importance of this component I conducted a different set of experiments comparing *SWAP1*'s predictive accuracy when the hypothesis-selection is active (*SWAP1*-active) with *SWAP1* when the selection is not active (*SWAP1*-inactive). The results showed a significant advantage in predictive accuracy of *SWAP1*-active when compared to *SWAP1*-inactive. The

results highlight the relevance of this model-selection component —as opposed to the lookahead-lookback component— to the overall system's behavior.

In summary, by looking into the interior of an learning algorithm we can scrutinize current learning mechanisms and pose questions about their generalization effects. For example, what components are primarily responsible for the algorithm's behavior? Is the mechanism of model $\mathcal{X}$ effective under irrelevant features, high feature interaction, noise? What component of model $\mathcal{X}$ is negatively affecting performance? What components on models $\mathcal{X}$ and $\mathcal{Y}$ can be combined to yield some desired functionality?, etc.

Basic Steps for the Development Process

If existing learning models fall short in satisfying the desired goals, the development of a new model is necessary. A simple outline of a methodology to create a new model, detailed in Chapter 3, comprises the following steps:

1. Analyze the class of domains under study.
2. a) Define overall goals and subgoals, followed by
   b) The definition of a strategy.
3. Build, isolate, and analyze components.
4. Construct mechanism by assembling components.

The key point is to define a new model according to the characteristics of the class of domains under study, guided by the operations we wish the algorithm to perform. A functional decomposition analysis can then clarify the role of each component during learning.

An assessment of the usefulness of introducing a functional view in the development process is inherently difficult. A new learning algorithm may denote substantial improvement only if the right functionalities, say $\{f_1, f_2, \cdots, f_n\}$, are implemented. We can represent the space of all algorithms that include/omit each f_i through an n-dimensional boolean space. Hence, 2^n algorithms are required to verify that an increase of performance is observed as each f_i is included. Since such an exhaustive procedure is infeasible, I rather analyze and report on case studies that provide empirical support for the following claims:

- That existing models possess limitations (Section 4.2), and that a functional decomposition analysis can help elucidate the individual contribution of each component (Sections 4.3), and
- That components are not exclusive to particular models; new learning algorithms can be created when components from different models are combined (Section 5.3).

1.4 The Hierarchical Concept Learner

This thesis is divided into two parts. The first part is a study of the development of new algorithms in which I show the value of a functional decomposition analysis. In the second part, the basic steps for the development process above are applied to the construction of a new learning algorithm: *HCL* for Hierarchical Concept Learner. The development of *HCL* starts with two functionalities in mind:

- Flexibility in the concept representation.
  The amount of complexity in the induced hypothesis is commonly fixed by the concept language representation. Since the necessary complexity is ultimately determined by the class of domains of study, an algorithm should be able to vary the set of potential hypotheses along a spectrum of varying complexities.
- Adaptability in the degree of complexity.
  Flexibility in the concept representation is useful only if the learning algorithm is able to find, i.e., to adapt to, the degree of complexity that maximizes the predictive capabilities of the final hypothesis. Learning under this view is equivalent to a search for the best available representation.

The specific mechanism coupling these two functionalities to conform a new model is described in Chapter 7.

1.5 Main Points

The main points of his thesis can be listed as follows:

1. I formally characterize the set of structured domains over which learning is feasible (using Kolmogorov complexity), and explain how transformation operators enable us to learn difficult domains.

2. I outline basic steps for the development of new learning algorithms that highlight the importance of a search over a functional space. The considerations necessary to follow the steps are elaborated.

3. Using two case studies, I reveal specific limitations assumed in current models, and show how components from different strategies can be brought together in a single mechanism.

4. Based on my proposed procedure, I implement a new learning algorithm. I analyze the new algorithm analytically and empirically, and demonstrate its performance.

1.6 Goal Statement

The analysis and case studies of the first part of this thesis, together with the implementation and experimental study of the second part, give rise to the following questions:

- Regarding the development of new algorithms:
  - Precisely which and how many functionalities must be implemented before significant improvement is observed over standard learning models?
  - How can we avoid the limitations attached to current models within the new design?
  - Under what conditions is the construction of a new model necessary?
  - How effective is a functional decomposition analysis to identify possible limitations in the design?
  - Is the implementation of functionalities through a set of components always a feasible task?
- Regarding the design of *HCL*:
  - What lessons can be drawn from the design and implementation of the algorithm?
  - How well are the functionalities in the design achieved by the proposed implementation?

- What is the performance of the algorithm when compared to standard models?
- Is the current implementation amenable to extensions through refinements? Is the basic strategy easy to reconfigure?

Results obtained from this thesis will enable us to answer these questions (Chapter 10). Such information will help achieve the central goal of this thesis:

> **Goal:** To improve our knowledge on the development of learning algorithms so that the designer has enough information to decide on the learning strategy and its mechanism, ultimately achieving improved performance.

The idea is to open new paths to the design process by correcting common misconceptions: existing models are not the only source of learning strategies; new algorithms can be obtained by looking into a functional space, without *exclusively* relying on model refinements.

1.7 Thesis Organization

This thesis is organized as follows:

- The first part deals with the development of inductive algorithms. Chapter 2 discusses fundamental issues in machine learning, explaining why induction from examples remains an important and feasible task. Chapter 3 outlines basic steps for development, emphasizing the importance of a functional view. Chapter 4 shows the importance of identifying and analyzing individual components. Chapter 5 illustrates how components from different strategies can be combined into a single mechanism.

- The second part describes the new *HCL* algorithm. Chapter 6 expands on the strategy and design. Chapter 7 details on the mechanism of *HCL*, analytically studying each of the algorithm components. Chapter 8 performs a functional decomposition analysis over *HCL*. In Chapter 9, *HCL* is compared with standard learning models over both artificial and real-world domains.

- The last part gives a summary and conclusions (Chapter 10), giving answer to the questions posed in the previous section (Section 1.6).

Part I

On The Development of Learning Algorithms

Chapter 2

Fundamental Issues in Inductive Learning

Some studies in machine learning (Schaffer, 1994; Watanabe, 1985; Wolpert, 1996) extend formal proofs about the impossibleness to proclaim a learning algorithm superior to other algorithms, thus rendering the field of inductive learning a zero-sum enterprise. I devote this section to the analysis and clarification of these results. A study of how to develop more efficient algorithms would appear otherwise useless. This chapter contains definitions and discussion that will support the central goal of the thesis.

After a brief introduction to the comparison-of-algorithms problem, I show how inductive learning can still yield superior algorithms when attention is focused on the set of highly-structured domains (i.e., domains with low Kolmogorov complexity). For this to hold we need assumptions about the kinds of concepts worth learning, about expected sample-distributions, and other factors that can together characterize the set of typical real-world situations. I exemplify these ideas using several practical domains.

2.1 Basic Definitions

In inductive learning, the input to an algorithm L is given by a training sample T of size m, where each element or example $e_i \in T$, $1 < i \leq m$, is a pair $e_i = (X_i, c_i)$. Each X_i is a vector of n features (i.e., attributes, variables), $< x_i^1, x_i^2, \cdots, x_i^n >$, describing certain (presumably relevant) characteristics of example e_i. Together with a characterization of e_i is the class assigned by an underlying concept C (i.e., a teacher) to X_i, i.e., $C(X_i) = c_i$. Sample T is assumed to come from a fixed but unknown probability distribution D, by sampling m examples according to D. The underlying target concept, $C : X \mapsto \{1, 2, \cdots, k\}$, classifies the space of all possible examples, also referred to as the *instance space*, into k classes. A learning mechanism (i.e., inducer) attempts to discover C by analyzing the information given by the

training set $T : \{(X_i, c_i)\}_{i=1}^{m}$. The result of the analysis is a hypothesis/classifier H approximating C. Our main interest is in the ability of H to correctly *predict* the class of examples outside T. We look for hypotheses not only consistent with T, but that *generalize* beyond that set.

2.2 Is Inductive Learning a Zero-Gain Task?

Strictly speaking, a learning algorithm L_A cannot be deemed superior to any other algorithm L_B, unless certain specifications are brought into play. We shall refer to a learning domain or situation S as a 3-tuple, $S = (D, m, C)$, comprising a sample distribution D, a training-set size m, and a concept C[1]. Now, if algorithms L_A and L_B are compared by averaging predictive accuracy over training sets obtained by varying over all possible D, m, and C, then the advantage of L_A over L_B in some domains, must be counterbalanced by exactly the same advantage of L_B over L_A in all other domains. The notion of inductive learning being a zero-gain task has been identified and formalized by several researchers (Schaffer, 1994; Watanabe, 1985; Wolpert, 1996; Rao, Gordon, & Spears, 1995). Rao et al. formalize the above statement as follows[2]

$$\forall L \sum_{D} \sum_{m} \sum_{C} [\mathrm{GA}(L, D, C, m) - 0.5] = 0 \tag{2.1}$$

meaning for all learning algorithms L, averaging over all possible sample distributions D, all training samples of size m (for all possible m), and all concepts C, the generalization accuracy GA of L is on average always 0.5. The term between brackets is called generalization performance, and measures the difference in accuracy between the learner and random guessing. Generalization performance is defined as generalization accuracy minus 0.5, where accuracy is measured over the off-training set (i.e., the set of examples outside the training set). In brief, equation (2.1) states

[1] A domain is not only specified by the target concept C: the number of examples and the underlying distribution from which the examples are drawn play an equal important role in determining the difficulty of the learning process. Hence a distinction must be made between a concept C, and a learning domain or situation S comprising C.

[2] The equation is known as the conservation law for generalization performance (Schaffer, 1994). Wolpert (1996) refers to it as the no-free-lunch theorem, Watanabe (1985) as the ugly-duckling theorem.

that any algorithm showing good performance over certain domains, would do worse than random guessing in other domains.

Discussion

Each of the parameters D, m, and C must be restricted to a certain set of values before we can turn induction into a profitable task. Equation (2.1) implies that one must specify precisely the domains or situations $S^* = \{S_i\}$ in which a learning algorithm L is expected to yield improved performance. Considering all possible scenarios renders any inductive process a zero-gain task. As Wolpert (1996) comments:

> In fact if you press them, you find that in practice very often people's assumptions don't concern [priors] at all, but rather boil down to the statement "okay, my algorithm corresponds to an assumption about the prior over targets; I make that assumption". This is unsatisfying enough ...no one has even tried to write down the set of [priors] for which their algorithm works well. This puts the purveyors of such statements in the awkward position of invoking an unknown assumption.

In agreement with the statement above, we must recognize the importance of specifying the domains in which we desire good performance. In the next section I attempt to characterize the set of domains over which learning becomes truly relevant. I believe this initial step is important to identify the kinds of biases necessary during the construction of new learning algorithms. For simplification, the rest of this chapter assumes a fixed training-set size m and fixed distribution D; two domains will differ if they specify different concepts[3].

I believe general assumptions can be made about what characterizes the set S^*_{struct} of *all real-world* concepts. Consider the space of all possible concepts S_{all}. Since many of them are purely random, bearing no relationship to our world, it is reasonable to assume that S^*_{struct} occupies only a minuscule fraction of S_{all}, i.e., $S^*_{\text{struct}} \subset S_{\text{all}}$ and $|S^*_{\text{struct}}| \ll |S_{\text{all}}|$. Our goal is to develop learning algorithms uniformly superior over S^*_{struct}, which implies focusing on concepts having a high probability of occurrence in practice. This is acceptable even when worse-than-random performance is obtained somewhere else in S_{all} (e.g., random concepts).

[3]Since I assume a fixed sample size and probability distribution, a domain is equated to its corresponding concept; this is, strictly speaking, an abuse of terminology.

2.3 On Concept Complexity and Distribution

One dimension along which we can differentiate between structured and random concepts lies in the amount of data compression that can be obtained over the learning sets. Structured concepts S^*_{struct} denote regular patterns over the instance space that commonly lead to the discovery of concise representations. Random concepts, on the other hand, are characterized by many irregularities; long representations are then necessary to reproduce the original body of data. But how can we measure the degree of structure (conversely the degree of randomness) of a concept? To that end I rely on a measure of complexity known as *Kolmogorov Complexity* (Vitányi & Li, 1996; Li & Vitányi, 1992, 1993; Vitányi & Li, 1997).

Given a concept C, the Kolmogorov complexity of C, $K(C)$, is defined as the length of the shortest effective description of C. More rigorously, $K(C)$ is the length of the shortest binary program from which the concept C can be reconstructed (Vitányi & Li, 1997). Unlike other measures (e.g., classical information theory) Kolmogorov complexity considers the maximal degree of compressibility over the data under analysis. Suppose we classify all possible concepts according to its Kolmogorov complexity in the following way. For a fixed concept C, I denote the complexity of C as

$$K_{D,m}(C) = E(K_{C,D,m}(T)) = \sum_{\text{all } T_i \text{ of size } m} P(T_i)\ K_{C,D,m}(T_i) \tag{2.2}$$

$K_{D,m}(C)$ is the expected value of the complexity (i.e., degree of randomness) of training set T (conditioned on C), for a fixed size m, and a fixed distribution D over which the examples of T are drawn. This definition can serve to classify the universe of all possible concepts according to K. Since D and m are fixed in this simplified analysis, I will refer to $K_{D,m}(C)$ as simply $K(C)$.

Figure 2.1a depicts a Cartesian plane where the x-axis orders all possible concepts by increasing Kolmogorov complexity as defined in equation (2). If our learning bias grants more credit to hypotheses denoting structure and regularity, then the probability Y of finding a good approximation to a concept C can be represented as the function on Figure 2.1a. Here an assumption is made that structure pervades our universe, and that a highly compressed representation is almost always attainable. This is a philosophical argument. The reader may disagree with this view of the universe in which case the assumption can still remain valid. Consider that of common

interest is to work with structured concepts alone, where a highly compressed representation of an external pattern can explain the nature of certain phenomenon. This permits us to apply an induced rule (i.e., principle) into different scenarios. Under this assumption, we expect a higher probability of successfully learning a concept as the irregularities of C vanish.

The x-axis under the function in Figure 2.1a can be divided into two regions: Region A, where each concept C has a probability greater than $1-\gamma$ of being correctly approximated; and a region B where a probability less than $1-\gamma$ is to be expected, due to the degree of randomness present. We can now define the set of all structured concepts S^*_{struct}, given a parameter $\gamma < 0.5$, as all concepts lying within region A of Figure 2.1a. Formally, $S^*_{\text{struct}} = \{C \mid Y(C|K(C)) > 1-\gamma\}$, where $Y(C|K(C))$ is the probability of correctly approximating C conditioned on $K(C)$[4]. If we fail to approximate a concept C, we expect C to belong to region B, since random concepts offer little help for predicting the class of unseen examples.

While Figure 2.1a depicts the probability of correctly approximating a concept according to its inherent complexity, Figure 2.1b denotes the expected a priori probabilities for all concepts in a low-entropy universe like ours. Function P stands for a distribution typical of our universe, where structure and order appear so commonly. A more rigorous analysis could reveal the nature of P itself. Here we shall only be concerned that, under the analysis above, inductive learning can yield improved performance if we succeed on learning over all concepts in S^*_{struct}.

In summary, a characterization of real-world domains can be obtained by looking into concept complexity. Concepts with low (Kolmogorov) complexity facilitate learning because of the amount of data compression available: the presence of patterns and regularity provides the means to obtain a concise and correct approximation. If we think of real-world concepts as having low complexity, then a formal characterization can be made using this criterion alone. Under this view, the universe of all concepts can be divided in two regions: region A where all structured, low-complexity concepts abound; and region B, where high degree of complexity (randomness) precludes

[4]I make a distinction between PAC(C) (i.e., C is PAC learnable) and $Y(C|K(C))$. PAC - learnability is commonly used to determine the number of examples necessary to ensure strong learning results. Since I assume a fixed probability distribution and fixed sample size, Y is assumed here to provide the same strong results, but based on the complexity of the concept alone.

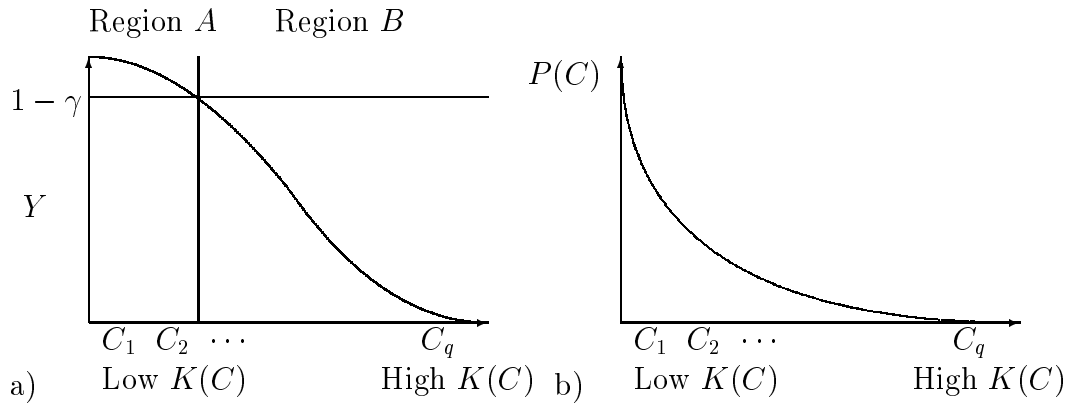


Figure 2.1: (a) Ideally, the probability Y of finding a good hypothesis should increase if the learning bias favors the presence of regularity and structure (i.e., the concept has low Kolmogorov complexity). (b) Expected a priori probabilities for every possible concept in our universe.

efficient learning. Since universal learning is impossible (equation 2.1), the design of a new algorithm must be aimed at learning all concepts in region A; this implies a duality between regions A and B, such that learning efficiently over A entails failing to learn over B. These ideas I further elaborate on the next section.

2.4 The Goal of Inductive Learning

Current learning algorithms are capable of learning on only a small subset of S^*_{struct} (Region A, Figure 2.1a). The search for powerful algorithms is intended to enlarge the number of concepts in S^*_{struct} where a significant advantage of generalization accuracy can be obtained. The goal of inductive learning is to produce algorithms that are uniformly superior over the concepts in S^*_{struct}.

To formalize these ideas consider for a moment all concepts within S^*_{struct} equally (for a given γ), ignoring their degree of complexity, as every $C \in S^*_{\text{struct}}$ is assumed already highly structured[5]. From equation (2.1), we know increasing generalization performance on certain class of concepts must be counterbalanced by poor performance on other classes. I call this the *duality effect*. Given an algorithm L, the generalization performance of L over the universe of all concepts S_{all} divides the space into two mutually exclusive and exhaustive sets: S_+, S_-, where generalization

[5]The concept can be simple, e.g., C = TRUE. Structured implies having low complexity in the sense of Kolmogorov.

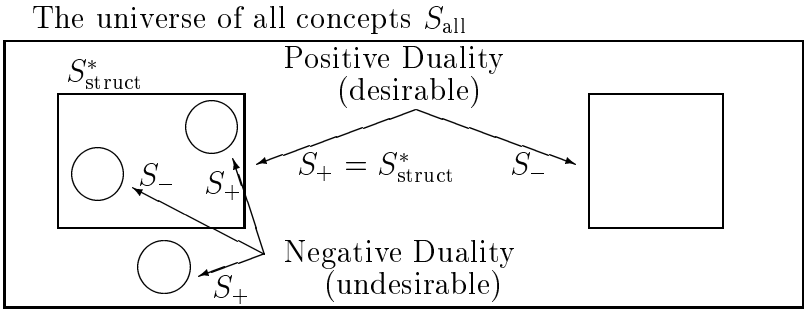


Figure 2.2: Inductive learning aims at learning all concepts within S^*_{struct}, which implies a positive learning duality. Failing to learn inside S^*_{struct} is not our goal.

performance takes on positive and negative values respectively. Now, suppose there is a class of domains $R \subset S_{\text{all}}$ of particular interest to us. Ideally we would like $R = S_+$, so that if negative performance occurs, it is left outside this class. If this is true for all concepts in R, I say a *positive duality* has been achieved. Nevertheless, it may happen that L performs negatively in some concepts within R. In that case I say a *negative duality* has occurred.

Going back to the goal of inductive learning, we are interested in developing new algorithms where the class of concepts of interest is the set of all structured real-world domains, i.e., $R = S^*_{\text{struct}}$. A *negative duality* occurs if any subset S_- of S^*_{struct}, $S_- \subset S^*_{\text{struct}}$, is poorly learned by L. In other words, a negative duality occurs when we fail to learn any concept in S^*_{struct}. The complement set to S_-: S_+, where good performance by L is observed, can exist anywhere outside S_-. The goal of inductive learning is to yield an advantage over *all* concepts in S^*_{struct}, and to create a *positive duality* between S^*_{struct} and a set of concepts outside S^*_{struct}, i.e., $S_+ = S^*_{\text{struct}}$. A positive duality means an advantage of L exists on every concept in S^*_{struct}. In summary, being successful in learning over S^*_{struct} must be (gladly) paid off with poor performance somewhere outside this set. But failing to learn inside S^*_{struct} is certainly not our goal. As shown in Figure 2.2, a negative duality appears when $S_- \subset S^*_{\text{struct}}$; it is only when L can successfully learn all concepts in S^*_{struct} that a positive duality occurs.

2.5 Concepts Difficult to Learn: Violating the Assumptions of Simplicity and Similarity

But, when do we find S_{-} lying within S^{*}_{struct}? In other words, when is a structured concept hard to learn for current learning algorithms? The difficulty attached to certain concepts may originate from common assumptions of similarity-based learning algorithms (Rendell, 1986; Pérez, Vilalta, & Rendell, 1996), in which simplicity and similarity play significant roles. Before explaining the nature of these two assumptions we shall first analyze what are the possible degrees of feature representation. On the one hand, the presence of domain experts may help in crafting features that are highly representative to the target concept. Learning in this case is simple, because feature interaction is low, and each feature conveys much information about the target concept. Nevertheless, lack of expertise forces us to use *primitive* features, where the information conveyed by each feature is too low level; interaction is then exacerbated and complex interactions take place.

The Nature of Simplicity and Similarity

Simplicity and similarity are only valid when highly representative features exist. Simplicity speaks of the concept-language representation: hypotheses are constructed assuming simple feature interaction. Frequently the hypothesis is built by adding one feature at a time, which in case of complex interactions produces large and complex structures. Similarity presupposes that two examples close to each other in the instance space share similar class value. Here an assumption is also made that features are highly representative, which may not be always true.

Before I explain why simplicity and similarity are not always valid assumptions, we shall first look into a practical measure of difficulty. Later I will argue that simplicity and similarity become inappropriate when the domain is characterized by high values of this measure of difficulty.

Concept Variation: A Measure Of Difficulty

A need exists to estimate the difficulty attached to a training sample to understand when the assumptions of simplicity and similarity become invalid. Section 2.3 gives a *theoretical* characterization of all structured domains based on the Kolmogorov complexity of a concept. In practice, however, this measure is unattainable (Li & Vitányi, 1993). This section describes a practical way of computing concept difficulty

by quantifying how uniform (conversely irregular) is the distribution of class-labels of examples lying near each other in the instance space (this measure is not an approximation to the Kolmogorov complexity of a concept; it is simply a practical approach to indicate when current models fail to attain good approximations *despite of* the (possibly) high degree of structure and regularity characterizing the class of domains under study). The idea is to determine if there exists high variation in class values among neighbor examples. If true we would expect a similarity-based bias to fail, because prediction is based on the class distribution of the neighborhood around a new example.

The degree of variation in the instance space can be estimated through ∇ (Rendell & Seshu, 1990; Pérez & Rendell, 1996), which provides an estimate of the probability that any two neighbor examples differ in class value. ∇ roughly measures the amount of irregularity in the distribution of examples along the instance space, and is defined as follows. Let $X_1, X_2, \cdots, X_n$ be the n closest neighbors – at Hamming distance one – of an example X in an n-dimensional boolean space. The degree of class *dissimilarity* of the neighborhood around X can be estimated simply as

$$\sigma(X) = \sum_{j=1}^{n} \text{diff}(C(X), C(X_j))$$

where $\text{diff}(C(X, X_i)) = 1$ if $C(X) \neq C(X_i)$ and 0 otherwise. A normalization factor $\overline{\sigma}(X) = \frac{\sigma(X)}{n}$ gives a value in $[0, 1]$. Concept variation is defined as the average of this factor when applied to every example in the instance space:

$$\nabla = \frac{1}{2^n} \times \sum_{i=1}^{2^n} \overline{\sigma}(X_i) \ \in [0, 1]$$

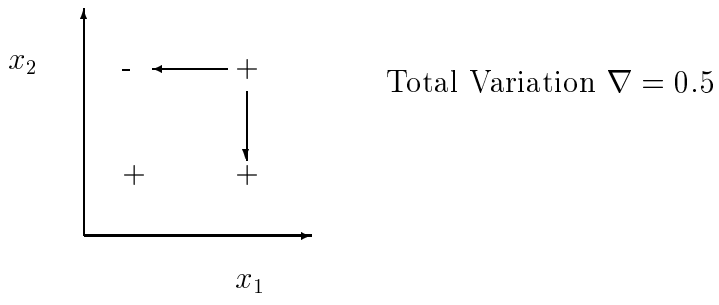


Figure 2.3: Computing variation ∇ in a Boolean 2-dimensional instance space.

As an example, Figure 2.3 depicts how concept variation is computed in a Boolean 2-dimensional instance space. The positive example in the right-top corner adds 1 unit of dissimilarity for the one negative example at hamming distance 1. This is then averaged over the two closest examples giving a variation of 0.5. Finally, the average variation for all four examples gives a value of $\frac{1.0+0.5+0.5+0}{4} = 0.5$. This means that, for any example X, the probability that a neighbor of X differs in class value is 50%.

Other practical ways of measuring difficulty are related to concept variation. One example is to count the number of disjuncts or peaks in the instance space (Rendell & Seshu, 1990). A concept characterized by many disjunctive terms complicates learning because each individual term must be identified and delineated. A different method consists of measuring the average entropy of the training data conditioned on each feature at a time, e.g., blurring (Ragavan & Rendell, 1991). High entropy means each feature alone is unable to discriminate examples of different class value; relations among features are complex and must be discovered to uncover the nature of the target concept. Preliminary tests to find a suitable measure for concept dispersion have shown that concept variation correlates better with accuracy degradation than other measures of difficulty for most similarity-based learning algorithms.

Cases Were Simplicity and Similarity Do Not Hold

Some concepts report high values of variation (i.e., high ∇) because the representation of primitive features is too low-level. Nevertheless, a concise representation may still be attainable once the dependencies among partial concepts is discovered (i.e., Kolmogorov complexity may be low). In this kind of concepts, assumptions like simplicity and similarity are inappropriate. I will refer to this group of concepts as *unfamiliar structured concepts*, since current learning techniques generally fail to generate good approximations despite the possibly low Kolmogorov complexity.

The conditions under which simplicity and similarity become inappropriate are deatailed next:

1. High degree of feature interaction. A concept may be defined as the combination of many subconcepts, each individual subconcept being equally defined as the combination of lower-level expressions. These complex dependencies may continue until we reach the set of primitive features. Since current learning

algorithms assume simplicity in the degree of feature interaction, finding good approximations for these domains is hard.

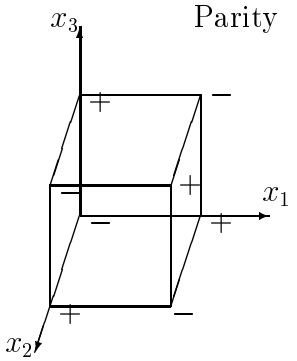
2. The existence of patterns stemming from the composition of separate regions over the instance space. The common assumption that proximity in the instance space relates to class similarity is sometimes inappropriate. As an example, assume a pattern repeating consistently throughout the instance space, but over separate space regions, e.g., parity. Discovering such pattern would correctly predict the class of an example that matches the place where the pattern occurs, even though the predicted class differs from the majority class of the neighborhood around such example.

The two domain characteristics above are similar in that both may produce a distribution of class-values over the instance space displaying high variation (i.e., a distribution where class labels differ within small regions). But the two characteristics differ in nature. High feature interaction means a single feature conveys low information about the target concept; only by evaluating subspaces of the instance space can relevant patterns be discovered. The second characteristic means there exists patterns that can be expressed as functions of clusters of examples. Most current learning algorithms search to identify a criterion to separate regions of examples with similar class. These regions could have strong correlations: the concept estimation could be more concisely described as a function of such regions (Section 10.3).

A concept may be highly structured even under the presence of feature interaction, or under patterns unfolding on separate instance-space regions. The next section analyzes some unfamiliar structured real-world concepts on which these sources of difficulty are present.

2.6 Examples in Artificial and Real-World Domains

The success of machine learning on many real-world applications indicates a subset of all structured concepts of interest, S^*_{struct}, has been efficiently learned. Performance on other concepts, however, remains unsatisfactory. In this section I analyze several real-world concepts that appear difficult for current learning algorithms. The idea behind a study of new algorithm development is to enhance current mechanisms to obtain improved performance over classes of concepts within S^*_{struct}.



Compressed Description:

$C(X) = \{X \mid ((\sum_{i=1}^{3} x_i) \bmod 2) > 0\}$

where $X = \{ x_1, x_2, x_3 \}$

Figure 2.4: (a) A 3-dimensional boolean space with odd-parity as the target concept. Parity is highly regular and structured, yet most current learning algorithms simply fail to learn this concept.

Parity Domain

To illustrate the ideas above I start by analyzing the parity domain. Odd-parity on a set $F_{\text{PAR}} = \{x_1, x_2, \cdots, x_k\}$ of Boolean features is simply defined as TRUE whenever the number of $1's$ in F_{PAR} is odd (conversely even for even-parity). A parity concept is highly structured and compressible (i.e., is characterized by low Kolmogorov complexity), because a simple pattern unfolds consistently throughout the instance space. Figure 2.4 shows a three-dimensional Boolean space where each point is labeled + or − according to whether it belongs to odd-parity over all three Boolean features. The labeling rule is simple: alternate class value among neighbor examples (i.e., examples at Hamming distance one). Why is then that most current learning algorithms are unable to learn on this domain (Thornton, 1996)? Figure 2.5 shows two learning curves for full odd-parity over training sets of 9- and 12-Boolean features when the algorithm employed is a univariate decision tree (*C4.5*(Quinlan, 1994)). The curves show a decrease in accuracy as the training-set size increases; the same effect can be observed on most current algorithms. Though it is reasonable to assume parity belongs to S^*_{struct}, current learning approaches exclude this concept from their scope (Rendell, 1986).

Chess, TicTacToe, and Other Board-Game Domains

Rules of games such as chess or TicTacToe give a complete characterization of the set of legal moves and states. In theory, any relation can be (extensionally) defined through the examples describing all board configurations. The typical combinatorial explosion associated to this space, however, restricts the use of learning algorithms to

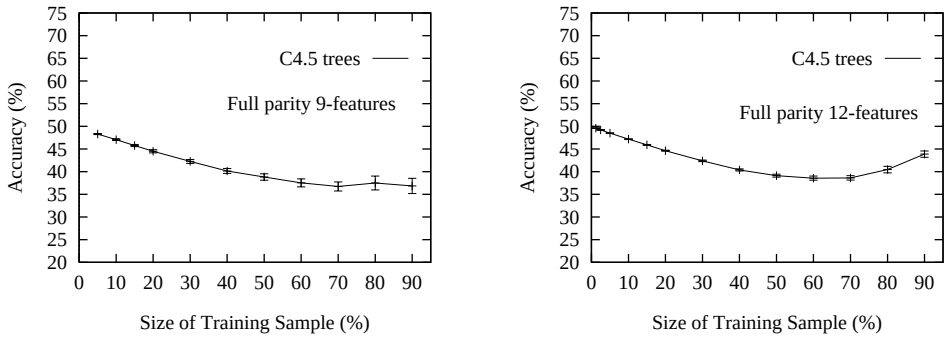


Figure 2.5: Training-set size vs. predictive accuracy for nine and twelve attribute concepts over full odd-parity.

end-games, where few pieces remain on the board. In some cases the board comprises few entries (e.g., TicTacToe), enabling the analysis of all legal states. The target concept is commonly defined as win or loss for either player within the next moves. Specifically, an example $X = (x_1, x_2, \cdots, x_n)$, describes a unique board configuration through features $x_1, x_2, \cdots, x_n$. Example X is labeled positive if a (predefined) player is guaranteed to win after the next d moves (d = depth of search), assuming both players adopt an optimal strategy.

Since the rules of the game are well defined, and the examples comprise all relevant information, these domains are structured and highly compressible, i.e., they belong to S^*_{struct}. For example, the game of TicTacToe comprises 958 legal states (out of 19,683); the concept of win can be described as a DNF expression of eight terms, each term conjoining only three literals (Matheus, 1989). In the (white) king-rook versus (black) king-knight chess end-game (Quinlan, 1983), the number of possible states with only four pieces on the board is of approximately 10^6. By creating new features that bring up high-level information about the game, the number of possible states (i.e., examples) can be reduced to a few hundred. Such degree of compression hints at the presence of few patterns from which all legal configurations can be reconstructed (though a formalism to describe relevant patterns for the winning positions may be lacking (Clarke, 1977)).

Despite the inherent structure associated with these domains, learning when features represent board configurations is complicated because each example conveys low-level information about the target concept. Poorly informative features require

the construction of many intermediate subconcepts, complicating the search for the right concept estimations.

Protein Folding Prediction

Proteins are macromolecules constituted by simpler molecules called aminoacids. An open problem in molecular biology is to predict the 3-dimensional spatial location of the atoms of a protein (i.e., tertiary structure) from sequences of aminoacids (primary sequence). While 3-dimensional structure can be known from X-ray diffraction patterns of crystallized proteins, the procedure is time consuming. Ideally an automated process could learn from sequences of aminoacids (examples) for which the structure of the protein (class) is known (Shavlik, Hunter, & Searls, 1995), in order to discover rules unveiling structural patterns.

In theory, the 3-dimensional arrangement of atoms in a protein is uniquely determined by the sequence of aminoacids. Nevertheless, a more common task than mapping aminoacid sequences directly into tertiary structure is an intermediate step attempting to predict what is known as secondary structure: a simpler, more local description of the structure of the protein. Learning secondary structure is important to bridge the gap between the information given by linear aminoacid sequences and the associated functional properties determined by the way the protein folds up (Lapedes, Steeg, & Farber, 1995). A secondary structure arises from folding patterns of the primary sequence, usually divided into three classes: alpha-helix, beta-strand, and (default) coil.

For secondary-structure prediction, a learning scenario is created by establishing a window over the primary sequence of a protein for which classes are known. The window comprises n aminoacids ($n \approx 13$), and the goal of the learning algorithm is to correctly predict the class for the middle aminoacid. Examples are created by shifting the window over the primary sequence, which presupposes a local analysis around the window's center is sufficient to discover relevant patterns.

Current attempts to predict secondary structure using machine-learning techniques have achieved performance far from optimal (Qian & Sejnowski, 1988; Rost & Sanders, 1993). Different approaches to amend this situation have been tried, e.g., a change of representation of the aminoacids along the primary sequence to more representative descriptions like size, hydrophobicity, etc. (Qian & Sejnowski, 1988; Ioerger, Rendell, & Subramaniam, 1995), with no substantial improvement. One dif-

ficulty (for non-homologous proteins) may stem from the local-analysis assumption described above.

Another difficulty is the underlying number of possible patterns needed to stabilize alpha-helixes ($\approx 10^3$), which complicates the mechanism of learning algorithms searching for one pattern at a time (Pérez, 1997). Consider, however, that the information necessary to encode proteins originates on a gene-alphabet of only four letters and a language to decode that information. Secondary-structure prediction is governed by rules achieving a high degree of compression when compared to an extensional definition based on linear sequences of aminoacids; it is an example of a concept of low Kolmogorov complexity for which current algorithms encounter serious difficulties.

2.7 Developing New Learning Algorithms

The past section suggests a direction for developing better learning algorithms. There exists real-world domains of a structured nature for which most algorithms fail to provide high predictive accuracy. The purpose behind this thesis is to provide guidelines on how to create new mechanisms that enable us to efficiently learn these unfamiliar structured concepts, i.e., concepts within $S^*_{\text{structured}}$ (Section 2.4). Therefore, we must extend our capabilities to cover those concepts in $S^*_{\text{structured}}$ eluding current learning techniques.

The set $S^*_{\text{structured}}$ can be split into two subsets: S^*_{simple} and $S^*_{\text{difficult}}$. The former subset encompasses those concepts for which current learning algorithms are presumed already effective. The latter set corresponds to concepts displaying high degree of structure (i.e., low Kolmogorov complexity), but that are difficult to learn (i.e., to unfamiliar structured concepts). The idea behind developing new algorithms is to bridge the gap between these two sets. We look for algorithms having built-in biases that guarantee a high probability $(1 - \gamma)$ of successfully learning a concept whenever this concept is highly compressible.

One approach to extend our learning coverage is to develop tools that transform a concept in $S^*_{\text{difficult}}$ to a form similar to S^*_{simple} (Figure 2.6). In other words, for any two concepts $C_i \in S^*_{\text{simple}}$ and $C_j \in S^*_{\text{difficult}}$, C_j can be transformed into a form resembling C_i, even though the Kolmogorov complexity of both concepts remains similar, i.e., $K(C_i) \sim K(C_j)$. The idea here is to exploit the capabilities of current

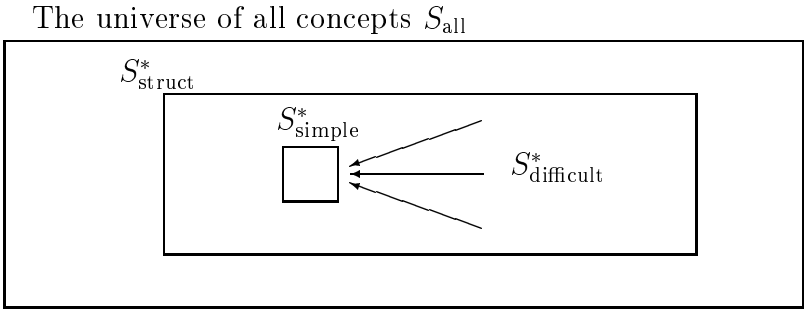


Figure 2.6: The set S^*_{struct} of structured concepts can be divided into two subsets: S^*_{simple} and $S^*_{\text{difficult}}$. A transformation process can modify the latter subset into a form similar to the former subset.

learning algorithms by reducing all concepts to a form that suits common learning biases. But, how can a concept be transformed?

Transforming The Learning Problem

I define two different approaches into which a learning problem, i.e., a training set of fixed size obtained from an underlying probability distribution, may be modified: 1) externally or 2) internally.

A training set can be modified externally by augmenting the dimensionality of the instance space through the addition of new features, or by reducing dimensionality through feature elimination. In either case, a transformation of the learning problem takes place. Automating the first operation falls within the realm of constructive induction (Michalski, 1983), in which the addition of new informative features to the learning problem helps regularize the distribution of examples over the instance space (Rendell & Seshu, 1990). The second operation avoids the negative effects that appear under the presence of irrelevant features.

A learning problem may be transformed internally by subdividing the problem into smaller subproblems; a simple mechanism for classification may then be applied on each subproblem. For example, most learning algorithms proceed by partitioning the instance space according to the information given by the training set; single-class regions are delimited in a way that depends on the specific algorithm's bias. An unseen example is classified by simply assigning the majority class of the region into which the example falls. Thus, it can be said that most current approaches to learning engage in some form of transformation (either external or internal) of the learning problem.

The notion of a transformation of the original problem into a form that becomes easier to analyze is already common. What remains to be explored is the kinds of learning algorithms that we need to develop to surmount the difficulties present in domains such as parity, board-games, protein folding prediction, and many others (Section 2.6). The purpose of this chapter is to show that the development of new algorithms is a legitimate task with a clear goal: to provide transformations that permit us to learn structured concepts efficiently.

Chapter 3

Basic Steps for the Development of New Learning Algorithms

The previous chapter characterizes the class of structured concepts where learning is possible, showing how improved algorithms can be constructed. This chapter deals directly with the problem of how to develop new learning algorithms. After introducing basic terminology, I provide definitions and concepts that help clarify the limitations attached to a search of new learning algorithms that is restricted to a design space. Afterwards I compare results when a functional space is brought into play. The chapter ends outlining basic steps for the development of new learning algorithms. This outline guides the design process according to the functional goals to accomplish, and the class of domains under study.

3.1 Definitions: Basic Algorithm, Strategy, and Mechanism

Research in inductive learning has produced a number of inductive models, e.g., neural nets, decision trees, instance-based learners, rule-based systems, bayesian estimators, and others (Section 1.1). The simplest form of a model I refer to as a *basic algorithm*. In a basic algorithm, only the necessary steps of the attached model are included, as shown in Figure 3.1 for a k nearest-neighbor algorithm. A basic algorithm is like a schema that can be instantiated in several ways (e.g., change distance metric DM in Figure 3.1), but retaining characteristics essential to the model.

The construction of a basic algorithm requires two specifications: a learning strategy, and a mechanism implementing the strategy (i.e., set of components implementing the strategy). A strategy can be seen as the general plan for learning, whereas the mechanism is the actual implementation executing the plan. For example, the nearest-neighbor algorithm of Figure 3.1 predicts class membership of an unseen example X by inspecting the neighborhood around X (strategy). This is accomplished by analyzing the classes of the k closest examples —in the training set— of X (mecha-

Algorithm 1: k Nearest-Neighbor
Input: Training Set $T : \{(X_i, c_i)\}_{i=1}^{m}$, where c_i is the class of example X_i; No. Closest Neighbors k; Distance Metric DM; Unseen example X
Output: Class predicted for example X
NEARESTNEIGHBOR(T, k, DM, X)
(1) According to metric DM, let $\{X_1, X_2, \cdots, X_k\}$
(2) represent the k closest examples to X in T
(3) Let class c be the class label output by
(4) a function over $\{c_1, c_2, \cdots, c_k\}$
(5) **return** c

Figure 3.1: A basic description of a k nearest-neighbor algorithm.

nism). Other basic algorithms adopt different strategies and mechanisms: A decision tree inducer recursively partitions the training set to identify regions in the instance space of similar class value. This divide-and-conquer strategy is attained by growing a tree in a top-down manner, selecting the best splitting function at every tree node according to some evaluation criterion (mechanism). Some rule-based systems use a separate-and-conquer strategy, by iteratively removing examples belonging to the same disjunct (mechanism).

Research in empirical learning often revolves around studying and modifying basic algorithms. A basic algorithm is simply a description of the mechanism implementing some strategy. A strategy is essential to a learning algorithm; it defines the particular approach adopted for learning. Refining a basic algorithm presupposes the strategy is correct (Section 1.2). Contrary to this view, this work rests on the assumption that strategies may have limitations. A strategy comprises a set of specifications that may either mistakenly perform an operation, or lack important functionalities. Therefore, attention must be paid to verify that a strategy is performing the right operations, before focusing on how legitimately a mechanism puts the strategy into work. In order to do this, we must first distinguish between what constitutes a strategy, and what lies outside its dominion.

3.2 Fixed, Flexible and Additional Components

We shall refer to each building block holding the structure (i.e., mechanism) of a learning algorithm as a component (Figure 1.2). I divide the structure of an algorithm into three types of components: a set of *fixed* components, a set of *flexible* components, and a set of *additional* components. This classification will help clarify how to adopt

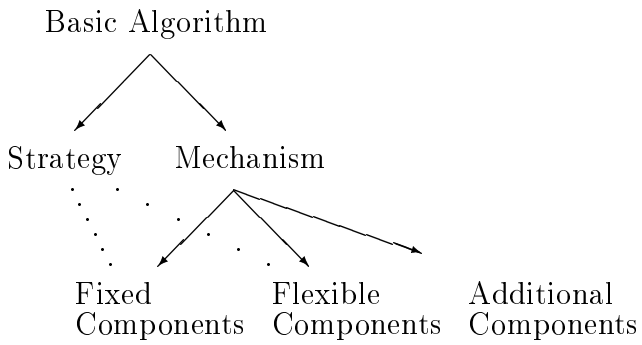


Figure 3.2: A basic algorithm needs two specifications: a strategy and a mechanism. The mechanism can be divided into a set of fixed components, a set of flexible components, and a set of additional components. The first two types of components support the underlying learning strategy; the last type lies outside the strategy's foundation.

a functional view during the development of new algorithms.

The first two components, fixed and flexible, are in charge of supporting the basic strategy; they are part of the algorithm's foundation. Removing any of these components would force the algorithm to collapse; it would suppose modifying the inherent strategy and thus the definition of a different approach to learning. The difference between these two components depends on if the designer can change their structure or not. Fixed components are unmodifiable: the designer cannot alter their structure without leaving the strategy intact. Flexible components, on the other hand, allow for modifications and refinements. Additional components are building blocks outside the core of the learning strategy. They incorporate important operations during learning, but only externally, as a means to obtain further improvements. These building blocks lie outside the strategy's foundation. Figure 3.2 depicts the structure of a basic algorithm.

To illustrate the above definitions consider the following examples:

- The nearest-neighbor algorithm of Figure 3.1 is always based on an inspection of the neighborhood around example X. The mechanism implementing this strategy comprises two flexible components: 1) a search for the k closest examples, and 2) a criterion to predict a class based on the classes of the k closest examples. The first component can be modified by choosing one of several pos-

sible metrics DM. The second component can select among different functions, e.g., take a majority vote over $\{c_1, c_2, \cdots, c_k\}$.

- The strategy behind a Bayesian estimator employs probability estimations to determine class membership. The mechanism contains at least one flexible component, because it allows the researcher to assume dependence or independence in the estimations for conditional probabilities.

- A decision tree can be modified in several parts, as for example, in deciding what kind of test (e.g., univariate, multivariate), stopping criterion, or evaluation metric (e.g., entropy, information gain, gain ratio, χ^2, Laplace) is more appropriate when building each node of a decision tree (flexible components). The construction of the decision tree through a recursive training-set partitioning, however, remains constant, i.e., is made of fixed components. Table 1 shows some fixed, flexible, and additional components for a decision tree algorithm.

- Some systems adopt additional components such as feature selection, numeric discretization, pruning[1], etc. before or after a candidate hypothesis is elaborated.

When the development of new learning algorithms is limited to a search on the design space (Section 1.3), a common approach is to propose modifications to the set of flexible components alone (additional components may be subject to modification too). This may result in an increase of performance, but always leaves the underlying strategy unaltered. Two common assumptions characterize this approach:

- Fixed components are exempt from analysis. The net effect that each of these components has during learning needs not be measured individually. Fixed components are commonly left unaltered, because it is not easy to detect where modifications can take place.

[1]Some researchers consider pruning a fundamental component in induction. The current analysis is based not on importance, but rather on what forms part of the basic strategy. Pruning may result in significant gains in predictive accuracy, but comes after a strategy has been already established (e.g., divide-and-conquer in decision tree induction).

Table 3.1: Components for a decision tree algorithm.

Fixed	Flexible	Additional
Partition training set for each splitting-function value.	Apply evaluation metric to identify best splitting function.	Detect relevant features using feature-selection.
Recursively grow a new subtree on every example subset	Stop splitting current tree node.	Prune final tree.

- The sequence that dictates the order in which components are executed is never questioned. Such sequence decides how the strategy is to be implemented. The sequence comprises all types of components (i.e., fixed, flexible, and additional) and specifies an arrangement that inevitably results in interactions. An analysis specifying how a component affects other components is assumed unnecessary.

With these two assumptions, a global evaluation of the new design serves as the only basis to assess the utility of the modifications. If the development process takes the strategy for granted, all we need is a persistent tweak of flexible (or additional) components to eventually obtain improvements. Fine-tuning a learning algorithm, however, does nothing to deepen our understanding of the new algorithm's behavior. Creating a new strategy is hard because no guidelines exist to conduct the process of assembling components. What is needed is a clear layout of how components interact within a certain strategy to achieve certain functionalities.

3.3 Components and Functionalities

When the development process is guided by the attainment of functionalities (i.e., by the set of goals we want the learning algorithm to achieve), rather than by the refinement of basic algorithms, fixed, flexible, and additional components become equally valuable tools to the designer. A strategy can be implemented through the assembly of components (Section 3.4), guided by a search over the functional space. Components are merged into a set of tools that will help achieve desired functionalities. Figure 3.3 contrasts the two views of the development process. Figure 3.3a

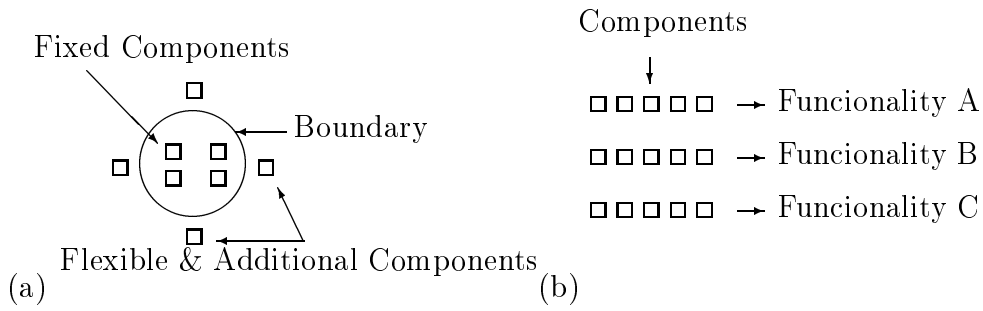


Figure 3.3: Two views of the development process. (a) The traditional view where only flexible and additional components (outside the boundary) are subject to modification. (b) A functional view where all components can be modified; components differ by their different contribution into the functionality being pursued.

illustrates the common view. Here, fixed components seem protected by taking refuge inside the boundary that decides what can be modified (or replaced) and what not. Figure 3.3b denotes a different scenario where components are distinguished only by their contribution to the functionality achieved. In the latter view, all components may be subject to modification or replacement.

While components are the building blocks to construct algorithms, functionalities serve as guidelines for the combination of components. A functionality can dictate selection of components, and can also indicate the order in which each component is invoked. For example, if a particular domain of application requires certain functionalities: effectiveness in elucidating feature interaction, ability to deal with feature and class noise, and output interpretable hypotheses; we can guide the design process bearing these functionalities in mind.

3.4 Basic Steps to Develop New Approaches to Learning

When no current learning model looks appropriate, a new approach to learning is desired. This section briefly describes some of the steps that are fundamental to the construction of new approaches to learning, i.e., to the construction of new models.

1. Analyze the Class of Domains Under Study

A first step consists of identifying the properties of the class of domains under study. If the application of the new algorithm is restricted to a certain type of domains, the design process can be guided by the particular domain qualities, e.g., presence of noise, feature interaction, abundance of small disjuncts, what kind of concept-language representation introduces the best bias (logical expressions, linear discrimi-

nants, high-order polynomials, etc.)?, are the features describing examples known to be relevant to the target concept?, are the features known to be primitive?, is there available information about the type of feature interaction?, etc. Information on these properties can greatly reduce the effort in building efficient algorithms. In most cases, however, little or nothing is known about the domains; properties then must be investigated through experimentation. An analysis must be carefully planned that can reveal the nature of the domains under study. This can be accomplished through external sources of information (i.e., domain knowledge), or by experimenting with the data itself.

One form to carry out the preceding analysis is by defining measures to assess the degree in which each property is present on every domain. Assume property P (e.g., noise) is known to exist within the class of domains of study $\mathcal{S}$, and that the degree of P in $\mathcal{S}$ can be estimated through a measure $R(P, s_i)$, for every domain $s_i \in \mathcal{S}$. Now, as an initial experimental phase, suppose we have a learning algorithm L we want to test over $\mathcal{S}$. L could be a new approach to learning or an existing model, i.e., basic algorithm, that we wish to analyze. By testing L over $\mathcal{S}$ we can observe how performance, e.g., predictive accuracy, varies with respect to $R(P, s_i)$. In other words, we can determine how much property P affects learning. The goal is to ensure that our new algorithm works well under all observed values of $R(P, s_i)$. This may lead us to a better understanding of how to build robust learners over $\mathcal{S}$, as well as to identify current limitations.

As an example, the class of domains $\mathcal{S}_{\text{artificial}}$ (Appendix A) comprises concepts varying over a wide range of different levels of concept variation (Section 2.5). Low variation means the class of any example will resemble the class of its immediate neighbors, i.e., a similarity-based bias is appropriate. High variation implies frequent class disagreement between neighboring examples. Suppose we decide to investigate how the property of concept variation $P_{\text{variation}}$ varies across $\mathcal{S}_{\text{artificial}}$. In particular, we wish to know the degree of $P_{\text{variation}}$ present on each domain in $\mathcal{S}_{\text{artificial}}$. One way to proceed is to apply existing measures of concept difficulty, e.g., variation (Rendell & Seshu, 1990; Pérez & Rendell, 1996), blurring (Ragavan & Rendell, 1991; Rendell & Ragavan, 1993), to every domain in $\mathcal{S}_{\text{artificial}}$. After each domain is assigned a degree of variation we could compare how this value correlates to predictive accuracy, to determine how much this property affects performance. We would then discover

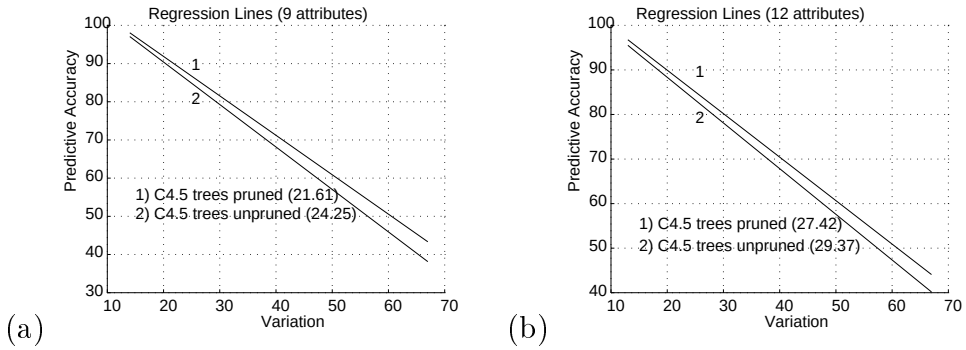


Figure 3.4: Regression lines comparing preditive accuracy vs. variation for 9- and 12-feature functions. Each regression line fits 9 points (i.e., 9 functions). Numbers enclosed in parentheses show the mean of the absolute differences between the regression model and the actual data.

that most learning algorithms are not designed to deal with high levels of variation. Figure 3.4 plots predictive accuracy against various degrees of variation over $\mathcal{S}_{\text{artificial}}$ when the learning algorithm is a decision-tree inducer (C4.5 (Quinlan, 1994); the same result has been observed with other algorithms). The plot indicates a degradation of performance as variation increases. We can conclude that learning algorithms can be armed to cope with domains in which the (possibly) high degree of certain property significantly affects performance, as long as the property can be effectively isolated and measured.

In summary, the design of a new algorithm should start by attending to the class of domains of study. Ideally, we would be able to estimate the degree to which each relevant property is present. Experimentation can then reveal how performance correlates with the presence or absence (spread over different degrees) of each property. One goal is to find limitations of current learning techniques, and then suggest new ways to correct deficiencies.

2. Define Goals and Strategy

Based on the information given by the class of domains under study, a second step is to define the set of goal functionalities we desire to achieve in the new algorithm (Section 1.3). Knowledge of domain properties can narrow down what the set of important goals is. For example, knowing that the available examples have been perturbed by noise patterns, we may wish to concentrate on a strategy able to detect these patterns, and to systematically eliminate them. In other situations, finding a

good concept estimation may be simplified by knowing how features interact; a design must be devised enabling us to select the most plausible interactions. If nothing is known about the domain, the designer may decide to concentrate on specific goals first, and to augment the learner incrementally until enough considerations have been heeded.

The definition of a set of goal functionalities must be done within a learning plan, i.e., within a strategy (see Section 3.1). The plan must ensure not only that the goals are being achieved, but that the search for good hypotheses fits within it. This is the hardest task during the design process because of the many choices available, and of the lack of support that arises when a pre-defined strategy is not adopted. A trade-off exists between having complete freedom to conceive a plan, and on choosing an existing model where the plan is already defined. The first alternative may prove the only acceptable solution when no existing models give satisfactory results, but at the expense of having to define new courses of action.

The preceding tradeoff can be made less critical if we define an outline of a general plan to learning. With this outline the designer can retain freedom of choice, but at the same time be guided by general principles. The following sequence enumerates basic steps for a learning strategy:

1. Define a concept-language representation. The language defines a space of admissible concepts; according to this language, identify how partial subconcepts are represented.
   For example, a decision tree is made of disjunctive expressions; a k-nearest neighbor extensionally defines a hypothesis by listing the k closest examples; a neural net is made of several weighted nodes.

2. Design a way to search for a hypothesis to the target concept by constructing —sequentially or in parallel— the partial subconcepts. Determine how subconcepts interrelate to build the final hypothesis.
   For example, a decision tree follows a divide-and-conquer approach to find multiple disjuncts; a k-nearest neighbor uses a distance metric to find the k closest examples; a neural net carries out a gradient-descent search to find the weights for each node.

3. Alternatively, refine the final hypothesis by adjusting structure. e.g., pruning.

Here a criterion must exist to prefer one structure over another. Generally the criterion is based on complexity, and on estimated performance.

The above framework is general enough to account for many different strategies. If adopted, the designer task is to integrate the set of goal functionalities into this framework. How this is done is explained in the next steps.

3. Build, Isolate, and Analize Components

Once a learning strategy has been devised, and a set of goal functionalities established, the designer is in position to work on the mechanism, i.e., the implementation executing the plan. This step is described through a sequence of two operations:

1. For each functionality, define and implement the associated components; components must ensure that the desired goal is accomplished. This operation is conditioned on the learning strategy, and thus limits the set of possible implementations. Assume, as an example, a separate-and-conquer rule-based strategy, where evidence advocates the presence of noise in the class of domains of study. A functionality to combat noise can be implemented by pruning rules, dropping off rule-elements according to a statistical-significance measure. Other forms to combat noise are possible, but each of them is subject to the specific kind of strategy governing how hypotheses are generated. Thus, a strategy that ignores functionalities hampers the implementation of potential components. For example, interpretability of a neural-net's output is a hard task (Towell & Shavlik, 1993, 1991), because the learning strategy excludes such goal from the beginning. Nevertheless, since complete control exists over the algorithm design, both strategy and functionalities can be planned simultaneously (or at least having each of them in mind).

2. Design experiments to test and verify that each component is working properly. Functional decomposition analysis can lead to better designs. Rather than relying on a single measure to assess the global behavior of the new algorithm, we must be able to distinguish the individual contribution of each component, and to verify that the right operations are taking place. This may become a difficult enterprise if it were planned for all components: some operations may be impossible to isolate from the rest; in other situations, individual tests

may be easy to perform. Alternatively, the analysis of components can be done analytically, by predicting their effect on a theoretical basis. In any case, the designer must submit the algorithm to careful analysis, to account for the proper working of each relevant operation.

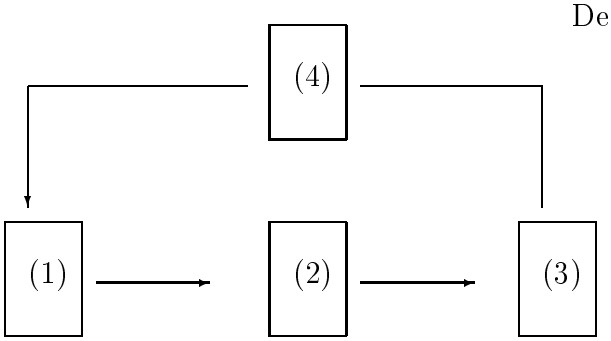
As in step 1, where I emphasize the importance of measuring the degree of every domain property, this step claims for measures that can evaluate each operation contribution. The most global performance criterion, e.g., predictive accuracy, conciseness, interpretability, etc, may differ substantially from the nature of individual evaluation functions. For example, suppose we wish to know the effectiveness of a feature-selection method f_{sel}. One approach is to evaluate L with and without f_{sel} run first; call the algorithms L_w and L_{wo} respectively. Even if L_w results significantly better than L_{wo}, it is difficult to see the differences that f_{sel} causes in L: no understanding is gained from the integration, except that the new approach outperforms the standard. Additional experiments could clarify the issue, if for example, artificial concepts are used to compare the features selected by f_{sel}, with the features that belong to the target concept; this degree of matching can say more about the selection method alone, without involving any algorithm.

4. The Assembling Process

Assume components guided by functionalities have been successfully implemented. What remains is the assembling process: the phase where a specific strategy is materialized, by arranging components into certain sequence. The process can be captured in a flow diagram, where each component is a building block, and the set of functionalities decide upon the sequence. For example, Figure 3.5 illustrates a flow diagram for the design of a simple decision tree algorithm. The diagram explicitly states the sequence of components that are followed during the learning process. We can identify the general strategy involved: elucidate interaction by partitioning the instance space into an exhaustive and mutually exclusive set of regions, following a divide-and-conquer approach. One functionality is to construct a DNF estimation of the target concept that is amenable to interpretation.

By separating an algorithm into different components we better understand its internal mechanism, which can lead to modifications, or even new strategies (i.e., new models). For example, components 3 and 4 in Figure 3.5 recursively grow a

Decision-Tree Algorithm



(1) if single-class region, create terminal node.
(2) if not, select/construct splitting function (using an evaluation metric).
(3) partition node sample on each splitting-function value.
(4) recursively apply sequence 1,2,3 to each new subsample.

Figure 3.5: The design of a learning algorithm specified through a flow diagram. A decision-tree algorithm is outlined through blocks, each block representing a single learning component.

new subtree on each new partitioned subsample. This kind of search for various disjunctive hypotheses may, in some scenarios, adversely affect accuracy (Section 4.2 details this specific issue). Since the components are part of the strategy (i.e., fixed components in the traditional view), *any* decision tree algorithm operates in this limited form.

Of particular relevance to the methodology above is the freedom of choice in the design process. In searching for functionalities, the designer can control both strategy and mechanism. This sharply contrasts with a traditional methodology based only in the refinement of basic algorithms. The next chapter provides evidence supporting the major claims of this study. Part II exemplifies these basic steps with a practical implementation.

Chapter 4

Case Study: Isolating and Analyzing Components

This chapter shows how decomposing learning algorithms into their constituent components is necessary to identify inherent strategy limitations. The goal is to provide evidential support to the main claims of this thesis. The analysis looks into the strategy behind decision tree algorithms (Quinlan, 1986; Breiman et al., 1984). Experimental results point to a key component during learning.

4.1 Introduction

This chapter is divided in two parts. The first part is a study of a limitation of decision tree induction (and other inductive models) known as the *fragmentation problem*[1]. I show how the divide-and-conquer strategy of *any* decision tree algorithm has this limitation. I describe a component to solve the problem named *global data analysis*. The study is significant in that a global data analysis necessarily breaks apart the strategy of decision tree algorithms. Thus, although the fragmentation problem is well known (Matheus, 1989; Kohavi, 1994), this study shows how a solution cannot rely on refinements to the mechanism alone, but must modify the strategy itself. The second part isolates and tests a component from a rule-based system (*C4.5*rules (Quinlan, 1994)), that incorporates this global data analysis. Empirical results clearly show how this component is responsible for eliminating the fragmentation problem.

Contributions from this chapter can be found on two levels. First, I show the importance of separating and individually studying learning components (Section 1.3), which enables us to assign the correct credit to each component in a learning mechanism. Second, I show experimental results quantifying the benefits of using a global

[1] This study is a summary of the paper by Vilalta, Blix, and Rendell (1997).

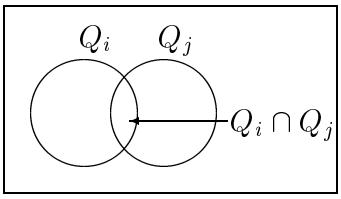


Figure 4.1: Disjunctive terms Q_i and Q_j represented on a Venn-diagram. Examples in the intersection, $Q_i \cap Q_j$, may support either Q_i or Q_j but not both.

data analysis when evaluating partial terms of a final hypothesis by resorting to all training examples.

4.2 The Fragmentation Problem in Decision Trees

A decision tree inducer adopts a DNF concept representation: each branch from the root of the tree to a positive leaf is equivalent to a disjunctive term Q_i. The final set of disjunctive terms must be mutually exclusive, i.e., $\text{COV}(Q_1) \cap \text{COV}(Q_2) \cap \cdots \cap \text{COV}(Q_l) = \emptyset$, where $\text{COV}(Q_i) = \{X \in T \mid Q_i(X) = +\}$ is the set of examples covered by term Q_i on training set T (Watanabe, 1969). The method to find every Q_i carries out a continuous partition-refinement over the instance space; every tree branch is grown until a terminal node or leaf delineates a single-class region. A limitation inherent to this approach is that, while searching for a disjunctive term Q_i, each splitting of the training data may separate or pull apart examples in support of a different term Q_j – due to the irrelevancy of the splitting function to Q_j. This situation not only requires that several approximations Q_j', Q_j'', Q_j''', etc., be found on dispersed regions of the instance space, giving rise to replicated subtrees along the output tree (Pagallo & Haussler, 1990; Matheus, 1989), but also reduces the support or evidential credibility of each individual term, eventually complicating its identification. This problem is known as the *fragmentation problem*, and has been attacked in different ways (Pagallo & Haussler, 1990; Ragavan & Rendell, 1993; Fayyad, 1994; Quinlan, 1994, 1987; Kohavi, 1994; Oliveira, 1995; Rymon, 1993; Rivest, 1987). Nonetheless, no clear solution has emerged.

To illustrate this problem, Figure 4.1 shows a Venn-diagram representation of the coverage of two disjunctive terms, Q_i and Q_j. Suppose a decision tree is built by first searching for term Q_i. Term Q_j is then left with the examples in $Q_j - Q_i$ as the only source of evidence. The examples in $Q_i \cap Q_j$ can either be sent to Q_i or Q_j, but not both. If the concept is characterized by having many disjunctive terms, growing a

hypothesis becomes progressively harder, as examples are continuously set apart by previous terms.

The fragmentation problem stems from two main causes:

1. The requirement that the coverage of disjunctive terms be mutually exclusive precludes the representation of any Q_i and Q_j such that $\mathrm{COV}(Q_i) \cap \mathrm{COV}(Q_j) \neq \emptyset$. The examples in $\mathrm{COV}(Q_i) \cap \mathrm{COV}(Q_j)$ are directed towards Q_i or Q_j, but not both. This is illustrated in the following example. Assume a 3-dimensional boolean space where each point represents an example $X = (x_1, x_2, x_3)$, with target function $C = x_1x_2 + x_1x_3$, as shown in Fig. 4.2a. Concept C can be decomposed into two subconcepts: $C_1 = x_1x_2$, with examples $(1,1,0)$ and $(1,1,1)$, and $C_2 = x_1x_3$, with $(1,0,1)$ and $(1,1,1)$. Let Q_1 and Q_2 be the disjunctive terms approximating C_1 and C_2 respectively. With all examples available and single features as splitting functions, two possible decision trees are depicted in Figs. 4.2b and 4.2c. In the tree for Fig. 4.2b, splitting on feature x_2 directs two positive examples to the right branch in support of Q_1, but only one positive example (out of two) to the left branch in support of Q_2. As a consequence, $Q_1 = C_1$ but $Q_2 \neq C_2$, since the irrelevant condition $\bar{x}_2$ is incorporated: $Q_2 = x_1\bar{x}_2x_3$. This was caused because $\mathrm{COV}(C_1) \cap \mathrm{COV}(C_2) = \{(1,1,1)\}$, which could only be covered by Q_1 or Q_2. The same phenomenon occurs in the tree on Fig. 4.2c, except here the loss of support occurs in Q_1.

2. Each partition over the instance space is too coarse, such that many steps are required to delimit a single-class region. The search for a disjunctive term Q_i inevitably results in the fragmentation of a different disjunctive term Q_j. Consider the tree in Fig. 4.3a for boolean concept $C = C_1 + C_2$, where $C_1 = x_1x_2$ and $C_2 = x_3x_4$. Assume all possible examples available, Q_1 the approximation to C_1, and $Q_2{}'$ and $Q_2{}''$ the approximations to C_2. Splitting on feature x_1 separates the examples in C_2, directing two positive examples (out of four) to the left branch in support of $Q_2{}'$, and two examples to the right branch. Splitting on feature x_2 reduces the support of $Q_2{}''$ to only one example. Since C_2 is represented by $Q_2{}'$ and $Q_2{}''$, the final tree replicates subtrees. This replication effect originates from the fragmentation of Q_2 (into $Q_2{}'$ and $Q_2{}''$) at the root of

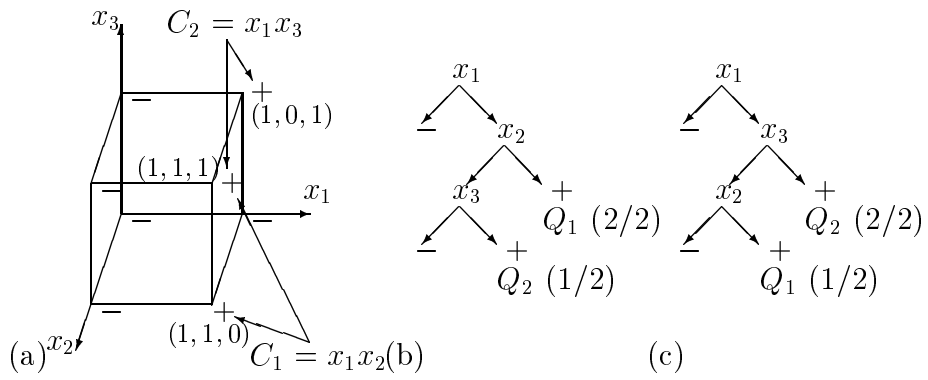


Figure 4.2: (a) A 3-dimensional boolean space for target concept $C = C_1 + C_2$, where $C_1 = x_1 x_2$ and $C_2 = x_1 x_3$. (b) and (c) Two decision trees for (a) where the fragmentation problem reduces the support of either Q_1 or Q_2, as shown by the fraction of examples that belong to C_1 and C_2 arriving at each positive leaf.

the tree.

One approach to combat the fragmentation problem is to conjoin several features at every tree node, which results in more refined partitions over the instance space. As shown in Fig. 4.3b, using the conjunction of single features as splitting functions eliminates the replication of the subtree approximating C_2. Nevertheless, Q_2 continues experiencing loss of support, since $\mathrm{COV}(Q_1) \cap \mathrm{COV}(Q_2) = \{(1,1,1,1)\} \neq \emptyset$. Hence, using multiple-feature tests at every tree node can reduce the number of partition-steps required to delimit single-class regions (cause 2), but cannot avoid pulling apart examples lying in the intersection of several partial subconcepts (cause 1).

The fragmentation problem is not exclusive to decision tree inducers but to any learning mechanism that progressively lessens the evidential credibility of its induced terms. Consider the separate-and-conquer strategy common to the construction of rule-based systems (Michalski, Mozetic, Hong, & Lavrac, 1986; Clark & Niblett, 1989). In this case, an iterative process starts by selecting a positive example or *seed* on the training data; this example is generalized to produce the next disjunctive term Q_i. The set of examples covered by Q_i, $\mathrm{COV}(Q_i)$, is *removed* before another seed is selected, potentially weakening the support of other disjunctive terms. A similar effect occurs in the mechanism for building decision lists (Pagallo & Haussler, 1990; Rivest, 1987).

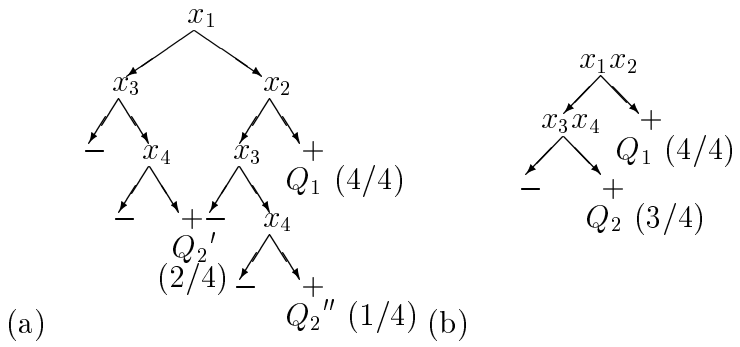


Figure 4.3: (a) A decision tree for $C = C_1 + C_2$, where $C_1 = x_1x_2$ and $C_2 = x_3x_4$. Examples in C_2 are separated into two regions after splitting on feature x_1. (b) A decision tree for (a) with multiple-feature tests. The replication of subtrees is eliminated, but Q_2 experiences loss of support when splitting on x_1x_2.

Concept Variation And The Fragmentation Problem

The degree of difficulty of a concept can be known through concept variation ∇, which estimates the probability that any two neighbor examples differ in class value (Section 2.5). The fragmentation problem is irrelevant over domains with low ∇ (i.e. over simple domains), because here each term (i.e. tree branch) delimits a region for which the majority of examples belong to the same class. But when dissimilarity in the vicinity of any example is high, as is characteristic in domains with high ∇ (i.e., as in unfamiliar structured concepts, Section 2.5), then separating examples away reduces the credibility of each disjunctive term. Experimental results in Section 4.3 show the effects of high variation in decision-tree induction.

4.3 A Component To Solve the Fragmentation Problem

The fragmentation problem results from continuously dividing examples belonging to the same disjunctive term. A simple solution consists of building each partial term independently, by assessing its value against all training examples together. The main idea is to preserve the evidential support of each term by looking into all data. This component I call *global data analysis*.

Figure 4.4 depicts a basic algorithm that incorporates a global data analysis. The goal is to better estimate the value of each partial term by avoiding the effects of previously induced terms. Under this framework, an approximation Q_i to a subconcept C_i is built under the support of all available data (lines 4-5, Fig. 4.4). The final

Algorithm 2: Learning with Global Data Analysis
Input: Assuming a target concept of the form $C = C_1 + C_2 + \cdots + C_l$, Training Set T, Metric M
Output: Final Hypothesis H_f
GDA-MECHANISM(T)
(1) Let $H_f = \emptyset$
(2) **foreach** $i = 1 \cdots l$
(3) Generate terms approximating subconcept C_i
(4) Evaluate each term by using all examples in T
(5) Select best term Q_i according to M
(6) Let $H_f = H_f \vee Q_i$
(7) **end for**
(8) Refine/Prune H_f by using all examples in T
(9) **return** H_f

Figure 4.4: Learning with global data analysis.

hypothesis H_f may also be refined (e.g., pruned) in this way (line 8, Fig. 4.4). Global data analysis is a component that ensures partial terms are properly supported.

In contrast, the search for disjunctive terms in decision tree induction is not global but local: often a term is supported by only a fraction of the examples of the subconcept being approximated. This holds irrespective of the modifications exerted on the learning mechanism (e.g., splitting function, pruning mechanism, stopping criteria, etc.), because such search is limited by the learning strategy. In the next section I evaluate the importance of a global data analysis.

The Value of Global Data Analysis

To measure the gains obtained when global data analysis is used to tackle the fragmentation problem, I performed a functional decomposition analysis of the mechanism of *C4.5*rules. The system can be described as a sequence of three components (see Quinlan (1994) for details): Component 1, where each individual rule is obtained from a previously constructed decision tree. Component 2, where the final set of rules is refined through a global data analysis: conditions on each rule are dropped until the error rate cannot be reduced; each rule is evaluated independently on all training data. This differs from component 3 where rules are refined to minimize description lengths. To isolate each learning component, I created three system versions:

- *C4.5*rules-*Std*, including components 1, 2, and 3 (i.e., Standard);

Table 4.1: Tests on predictive accuracy for both artificial and real-world domains. Columns for the different versions of *C4.5*rules and *C4.5*trees-pruned show the increase/decrease of accuracy against *C4.5*trees-unpruned. Significant differences (p = 0.05 level) are marked with an asterisk.

Concept	∇ (%)	*C4.5*trees-		*C4.5*rules-		
		unpruned	pruned	*Std*	*GDA*	*MDL*
DNF9*a*	14	99.2	−0.7	+0.5	+0.3	+0.0
CNF9*a*	17	99.5	−0.6	+0.5*	+0.5*	+0.0
MAJ9*a*	17	100.0	+0.0	+0.0	−0.8	+0.0
MAJ9*b*	21	82.1	−0.1	+2.1*	+1.4*	−0.2
CNF9*b*	22	94.0	+2.3*	+6.0*	+3.8*	+0.0
MUX9	22	86.8	0.9	+8.5*	+7.0*	−0.8
DNF9*b*	24	83.0	+0.6	+10.7*	+5.0*	−0.7
PAR9*a*	33	62.5	+13.5*	+27.7*	+21.7*	−2.2
PAR9*b*	67	43.1	+1.9*	+2.6*	+0.6*	+3.6*
MAJ12*a*	13	100.0	+0.0	+0.0	+0.0	+0.0
DNF12*a*	15	99.7	+0.0	+0.3*	+0.3*	+0.0
CNF12*a*	15	99.6	−0.1	+0.4*	+0.2*	+0.0
MAJ12*b*	18	84.5	−0.1	+3.0*	+2.7*	−0.6*
CNF12*b*	19	89.8	+0.7	+9.8*	+5.1*	−1.3
DNF12*b*	20	89.5	+3.2*	+9.6*	+6.6*	−0.6
MUX12	21	84.9	+0.1	+12.9*	+8.7*	−2.9*
PAR12*a*	33	58.6	+11.9*	+33.7*	+24.0*	−3.8*
PAR12*b*	67	46.5	+0.7*	+1.6*	+0.2*	+2.5*
TicTacToe		85.8	−0.3*	+13.3*	+10.4*	+0.5*
Lympho[2]		80.2	+3.3*	+3.0*	+2.1*	−0.5*
Lympho[3]		74.4	+3.1*	+6.1*	+5.4*	+1.7*
Promoters		81.9	−0.6	+5.0*	+3.5*	−0.9
Cancer		95.1	−0.6*	+0.8*	+0.5*	−0.3*
Hepatitis		82.3	−1.2*	−0.4	−0.6	−0.5
ABS						
AVRG		83.46	85.0	90.0	86.5	83.2

- *C4.5*rules-*GDA*, including only components 1 and 2 (i.e., isolating the Global Data Analysis component); and
- *C4.5*rules-*MDL*, including only components 1 and 3 (i.e., isolating the Minimum Description Length component).

Experiments

The next set of experiments evaluate the different contributions that *C4.5*rules-*GDA* and *C4.5*rules-*MDL* have over *C4.5*rules-*Std*. The main idea is to determine the effect that each component has over the whole system. Of particular relevance is to determine if *C4.5*rules-*GDA* is the component in charge of solving the fragmentation problem. To this end, each version of *C4.5*rules was compared to a simple decision

tree algorithm (*C4.5*trees (Quinlan, 1994)).

Table 4.1 illustrates results for predictive accuracy on artificial and real-world domains. The definition for artificial concepts is found in Appendix A. Each result is the average over 50 runs; a significant difference in accuracy (at the p = 0.05 level) is marked with an asterisk. Each group of 9- and 12-feature concepts is ordered by increasing variation ∇ (i.e., by a measure of data complexity; ∇ is defined in Section 2.5). Columns for the different versions of *C4.5*rules and *C4.5*trees-pruned show the increase/decrease of accuracy when compared to *C4.5*trees-unpruned.

On the set of artificial concepts, both *C4.5*rules-*Std* and *C4.5*rules-*GDA* (global-data-analysis component) show an increase of accuracy when compared to *C4.5*trees-unpruned as ∇ grows higher (the effect being more evident for 12-feature concepts than for 9-feature concepts). For $\nabla < 20\%$ the difference is indistinguishable. Under high values of ∇, *C4.5*rules-*GDA* achieves improvements of up to 27% accuracy on 9-feature concepts (parity PAR9a), and up to 34% on 12-feature concepts (parity PAR12a). The results show how the global-data-analysis component contributes significantly in the performance of the whole rule-based system (i.e., in the performance of *C4.5*rules-*Std*). The contribution is even more significant on difficult domains where the fragmentation problem is critical. The same trend is not observed on *C4.5*rules-*MDL*, where a maximum increase of only 3.6% is observed. Thus, a functional decomposition analysis is shown here to be important in assigning the right credit to each of the components under analysis. As variation increases, the minimum-description-length component in *C4.5*rules shows no significant contribution in the performance of *C4.5*rules-*Std*.

The same results above are depicted in Figures 4.5a and 4.5b for 9 and 12-feature concepts respectively. Here I computed a regression line for the difference in accuracy between each version of *C4.5*rules and *C4.5*trees against ∇. Concepts PAR9b and PAR12b were excluded from the regression analysis; these domains are outside the scope of similarity-based algorithms, because proximity in the instance space does not correlate to class similarity (Section 2.6). Numbers enclosed in parentheses represent the mean of the difference between the regression model and the actual data. An overall comparison reveals that *C4.5*rules-*GDA* is the component providing the most significant contribution. The regression line corresponding to *C4.5*rules-*GDA* lies close to *C4.5*rules-*Std*. In contrast, the regression line for *C4.5*rules*MDL* shows a

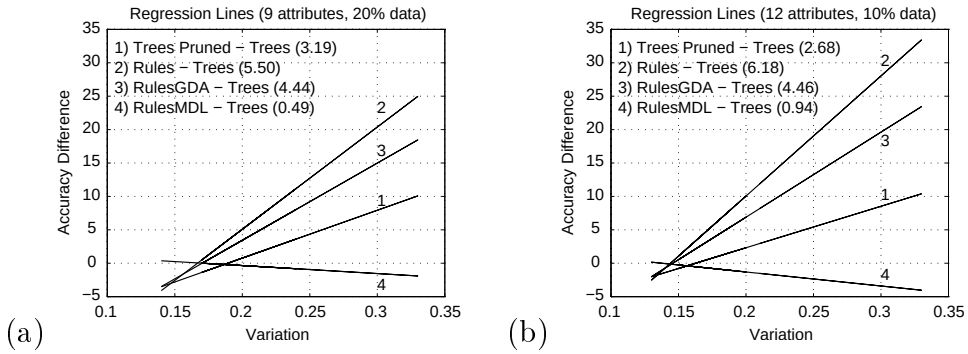


Figure 4.5: Regression lines for the columns of *C4.5* (all different versions) on Table 1 vs. Variation ∇ on (a) 9- and (b) 12-attribute concepts. Numbers enclosed in parentheses show the mean of the absolute differences between the regression model and the actual data.

general disadvantage when compared to *C4.5*-trees (for high variation).

To find out if the claims above hold outside artificial domains, the bottom part of Table 4.1 shows results on various real-world domains. The improvement gained with *C4.5*rules-*GDA* is observed on the Tic-Tac-Toe, Lymphography, and Promoters domains, where ∇ may be relatively high due to interaction among features bearing a poor representation to the target concept (e.g., board-configurations used to determine win or loss). These domains are examples of unfamiliar structured concepts for which current learning techniques fail to produce good aproximations (Section 2.5). The maximum difference is seen in the Tic-Tac-Toe domain where *C4.5*rules-*GDA* shows an advantage of 13.3% accuracy over *C4.5*trees-unpruned. The Promoters and Lymphography domains achieve a difference in accuracy varying between 3% and 6%. Thus, on real-world domains characterized by a low-level feature representation, the contribution of *C4.5*rules-*GDA* is shown to be significant, helping to improve the performance of *C4.5*rules. The same cannot be said in general for *C4.5*rules-*MDL*, where the difference in accuracy with respect to *C4.5*trees is in average approximately zero. Once more, decomposing an algorithm into its constituent components is shown here to be instrumental to understand the contribution of each component during learning.

4.4 Summary and Conclusions

Decision tree algorithms (and other inductive models) have an inherent limitation: a progressive loss of statistical support occurs at every new partition, as the number of examples giving credibility to every disjunctive term diminishes. The reason for such loss of statistical support is that upper nodes in the decision tree split the training examples in favor of a partial disjunctive term; but the partition may be irrelevant to other terms. This limitation is part of the basic learning strategy: refinements over the model cannot give a complete solution.

One way to solve the fragmentation problem is to incorporate all training data when assessing the value of every induced term. This global data analysis requires a concept-language representation other than a decision tree. Rule-based systems are more expressive than decision trees (the latter being a subset of the former). Since rules may overlap in their coverage, this representation facilitates adjusting the structure of the final hypothesis such that each rule is evaluated according to the information given by the entire training set.

A functional decomposition analysis evaluating separately a global data analysis and a minimum description length principle enabled us to quantify the contribution of each of these components in a particular mechanism (*C4.5*rules). A minimum-description-length principle gives a criterion to rank all consistent hypotheses (according to their size) based on a particular encoding, enabling us to select the most concise hypothesis. A component incorporating this principle in *C4.5*-rules shows a minimum contribution to the whole rule-based system. Figures 4.5a and 4.5b show that the global-data-analysis component gives the greatest contribution. This last component does more than a syntactic "refinement" over the final set of induced rules, effectively recovering the statistical support of every rule over the entire training set.

Observations

This study shows how a better understanding of what causes a learning algorithm to succeed or fail can be attained if the algorithm is viewed as the combination of multiple components, each component exerting a particular effect during learning. The kind of information obtained from a decomposition analysis is necessary when choosing the right functionalities in the design of new learning algorithms (Section 1.3). By identifying possible limitations in current designs, and by identifying the components

that can overcome those limitations, we become in a better position to create better algorithms (algorithms that enable us to learn a broader set of structured domains).

Section 1.6 in the introductory chapter addressed the following question (among others): How effective is a functional decomposition analysis to identify possible limitations in the design? This chapter shows how a functional decomposition analysis can be effective in elucidating the contribution of each component in a learning algorithm. Results have shown the importance of recurring to all available training examples when assessing the value of each disjunctive term: a consideration missing in the design of decision trees. Such consideration may be relevant in the construction of future algorithms.

While decomposing an algorithm is important to identify possible limitations in design, the construction of new algorithms needs to combine components into new strategies. The next chapter shows how components from different strategies can be combined into a single mechanism. This will serve as further evidence to support the claim that the construction of new learning algorithms can be guided by goal functionalities (Section 1.3).

Chapter 5

Case Study: Effectively Combining Components

This chapter shows how components that belong to different strategies can be integrated into a single algorithm. The idea is to look for the right combination of components once a collection of functionalities have been set as goals. I refute the common notion of some components being applicable as pre- or post-processing techniques to learning exclusively. I show how a more flexible view is to look at them as building blocks readily available for the internal design of new algorithms. The study serves as evidence to support the notion that goal functionalities can serve as guidelines in the development of new algorithms (Chapter 3).

5.1 Introduction

After learning components have been identified and analyzed, the design of a new algorithm must look into the integration of components. The purpose of this chapter is to show how components of different natures can be integrated to achieve a desired functionality. Thus, while Chapter 4 emphasizes the analysis of individual components, this chapter exemplifies how components can be combined. In particular, the common idea that some learning techniques must remain outside the internal mechanism of an algorithm is refuted. I claim most of these techniques can be part of the internal design of a new algorithm.

In this study I integrate techniques to combine multiple classifiers (Breiman, 1996a; Freund & Schapire, 1996; Wolpert, 1992; Breiman, 1996b; Quinlan, 1996; Buntine, 1991; Kwok & Carter, 1990) with feature construction (Matheus, 1989; Rendell & Seshu, 1990) in the context of decision tree induction[1]. The rationale

[1]The study is a summary of the paper by Vilalta and Rendell (1997).

behind this approach is explained next. Feature construction can be attained by conducting a search for the best logical feature combination at every node of a decision tree (Ragavan & Rendell, 1993). The exponential size of this search space, however, causes an instability: adding or removing few examples on each example subset tends to produce different best-feature combinations. In other words, slight variations in the examples populating certain instance-space region results in different best features to partition that region. Some immediate questions arise: How can this mechanism be stabilized? How can we avoid the effect that small variations in the examples exert over the construction mechanism? I set a functionality to pursue as the construction of a learning mechanism displaying stability in the production of new constructed features.

One possible solution to the problem above is to employ techniques that combine multiple classifiers (Section 5.4). These techniques are known to reduce variance (Breiman, 1996a), thus having a stabilizing effect. Nevertheless, such techniques are commonly seen as independent from the classifiers being combined; they are thought to perform a post-processing step to learning. I claim this view is limiting. Techniques to combine classifiers are applied outside the boundary of an algorithm's strategy because of the assumption that the bias of the algorithm is inscrutable (Section 1.2). This study shows how these techniques can be embedded into a decision tree algorithm, and thus help stabilize the feature construction mechanism.

The contributions of this chapter are enumerated as follows. First, the common view of multiple-classifier techniques as an approach to improve the performance of a learning algorithm, independently of the algorithm itself, is replaced. This chapter shows how these techniques can be viewed as local components to a learning algorithm. Second, the combination of feature construction with multiple classifiers, shown to be effective in Section 5.4, will serve as part of the mechanism for the design of a new algorithm (Chapter 7).

5.2 Background Information

Feature Construction

One form of feature construction carries out a search for the most promising logical expression. Using conjunction and negation as the only logical operators, a feasible approach is to conduct a beam search over the space of all boolean-feature conjuncts

or *monomials* (i.e. conjunction of boolean features or their complements), and to select that monomial with best performance according to an evaluation metric. This component is described and analyzed with detail in Section 7.1. A decision tree can be constructed by calling this component at every new tree node; I call the resulting decision-tree algorithm *DALI* (Dynamic Adaptive Lookahead Induction). The important issue here is that a splitting function conjoins several original features (e.g., $x_1\bar{x}_2x_3$).

This feature-construction mechanism is unstable. Slightly varying the examples on each node subset generates different best features. When a training set is obtained by sampling examples from an underlying probability distribution (i.e., stochastically), these variations are unavoidable.

Combining Multiple Classifiers

Multiple classifiers can be combined to improve classification. Specifically, a learning algorithm L is said to be *unstable* if small changes in the original training set T (i.e., adding or removing few examples in T) produce different hypotheses. Formally, let $R = \{T_1, T_2, \cdots, T_k\}$ be a set of training sets, each obtained by exerting some small changes in T. Applying L over each element in R produces hypotheses $H_1, H_2, \cdots, H_k$. In a typical scenario, L is applied to the original training examples in T to output a single hypothesis H; unseen examples are classified according to the predictions made by H. But if L is unstable, H may be a poor representative of the set of hypotheses obtained from variants of T. To avoid the instability of L, the algorithm can be designed to generate a set of different training sets R from T, apply L to each element in R, and then classify a new example X by voting over the prediction of all output hypotheses (i.e., by voting over $H_1(X), H_2(X), \cdots, H_k(X)$). The classification of X is now based on a combination of different hypotheses or classifiers. Figure 5.1 illustrates a basic algorithm for a general multiple-classifier system.

This chapter describes an empirical study employing two multiple-classifier techniques known as *bagging* (Breiman, 1996a) and *boosting* (Freund & Schapire, 1996, 1995). To understand the mechanism behind *boosting*, I first describe the simpler approach behind *bagging*. In *bagging*, a variant or replicate of the original training set T is generated (k times) by randomly sampling with replacement examples from T. Each variant approximates the underlying distribution D from which T originated, but may omit or duplicate some of the examples in T (i.e., each $D_i, 1 \leq i \leq k$,

Algorithm 3: Combining Multiple Classifiers
Input: Learning system L, Number of iterations k,
Training set T
Output: Compound classifier H_C
COMBINE-CLASSIFIERS(L,k,T)
(1) **foreach** $i = 1 \cdots k$
(2) Set current distribution to D_i
(3) Generate variant T_i from T according to D_i
(4) Apply L into T_i to generate hypothesis H_i
(5) Let a_i measure the performance of H_i
(6) **end for**
(7) Let $H_C = \{a_i H_i\}_{i=1}^{k}$
(8) **return** H_C
(9) An unseen example is classified on a majority-vote basis over H_C

Figure 5.1: A general description of a multiple-classifier algorithm

gives equal weight to every example in T). Applying algorithm L over each variant of T produces k hypotheses. An unseen example is assigned the majority class of the predictions made by these (equally weighted) hypotheses.

The mechanism behind *boosting* (Freund & Schapire, 1996, 1995) differs from *bagging* in that each example is assigned a weight; a variant of T is generated by sampling with replacement under this weighted distribution (i.e., examples with higher weight increase their probability of being selected). This distribution is modified for every new variant of T, by paying more attention to those examples incorrectly classified by the most recent hypothesis. Thus, in contrast to *bagging*, *boosting* takes into consideration the accuracy of each hypothesis (over the original training set T) to progressively improve the classification of those examples in T for which the last hypothesis failed. As in *bagging*, an unseen example is classified on a majority-vote basis, but the vote of each hypothesis H_i is weighted according to the accuracy of H_i on T.

5.3 Integrating Components

One approach to integrate feature construction with multiple classifiers incorporates both techniques locally, into a decision tree inducer. Current approaches to combining classifiers require a learning system L as input to generate several hypotheses (Figure 5.1); iterating over several decision trees to find new feature combinations is one form of feature construction (Pagallo & Haussler, 1990). In contrast, I view

Algorithm 4: Combining Multiple Expressions
Input: Number of iterations k, Current training subsample T_s
Output: Linear combination of expressions H
COMBINE_EXPRESSIONS(k,T_s)
(1) Let D_1 assign equal weight to every example in T_s
(2) **foreach** $i = 1 \cdots k$
(3) Set current distribution to D_i
(4) Generate variant T_s^i from T_s according to D_i
(5) Search for best expression F_i to split T_s^i
(6) Let a_i measure the error of F_i on T_s
(7) **if** $a_i = 0$
(8) Exit-Loop
(9) Let D_{i+1} assign more weight to those examples
(10) incorrectly classified by F_i
(11) **end for**
(12) Let $H = \{a_i F_i\}$
(13) **return** H

Figure 5.2: Expressions are linearly combined following the *boosting* algorithm.

both techniques as local components of L (L seen as a decision tree inducer), which permits analysis of their combined effect while maintaining a single hypothesis.

Specifically, every node of a decision tree covers a subset T_s of examples of the original training set T, i.e., $T_s \subseteq T$. Slight changes in T_s may produce different splitting functions, according to the instability of the mechanism F employed to select the best candidate function. Henceforth, techniques for combining multiple classifiers can be utilized locally, at each node of a decision tree, by generating variants of T_s, $\{T_s^1, T_s^2, \cdots, T_s^k\}$, and then applying F to each T_s^i to output different splitting functions $F_1, F_2, \cdots, F_k$. Example $X \in T_s$ is sent to the left or right branch of current tree node by voting over $F_1(X), F_2(X), \cdots, F_k(X)$.

To reduce the instability of the feature construction component in *DALI*, I experiment with both *bagging* and *boosting*. I apply each method at each tree node, by taking the set of examples in current node subset T_s as the training set T, and *DALI*'s search mechanism for new expressions as the learning algorithm L (Figure 5.2). Each method generates k new features from variants of T_s, so that each splitting function F can now be seen as a linear feature combination $F(X) = a_1F_1(X) + a_2F_2(X) + \cdots + a_kF_k(X)$. In *bagging*, $a_i = 1$ (i.e., each a_i is constant); in *boosting* each a_i reflects the accuracy of F_i over subset T_s. In the latter case, each feature F_i is seen as a hypothesis over the examples in T_s. The ma-

jority class of the examples in the coverage of F_i, $\{X \in S \mid F_i(X) = 1\}$, is taken as the class predicted by F_i in that set. The accuracy of F_i is the proportion of examples in T_s being correctly classified by F_i. In either *bagging* or *boosting*, the branch to which an example $X \in S$ is directed depends on whether $F(X) > (0.5 \cdot \sum_i a_i)$ is true or false. Both methods output a single decision tree structure.

5.4 Experiments

This section reports on experiments assessing the benefits of combining feature construction with multiple classifiers *locally* at each node of a decision tree. The techniques employed to combine expressions are *bagging* (Breiman, 1996a) and *boosting* (Freund & Schapire, 1996). I refer to combining *bagging* and *boosting* with *DALI*'s search mechanism at every decision tree node as *DALI-Node-Bag* and *DALI-Node-Boost* respectively (Section 5.3). For comparison purposes, the same strategies are applied (as typical) in a global fashion, over the decision trees output by *DALI*, here referred to as *DALI-Tree-Bag* and *DALI-Tree-Boost* respectively. Both *DALI-Tree-Bag* and *DALI-Tree-Boost* output multiple decision trees, thus complicating the interpretability of the final hypothesis. Selecting only the lowest-entropy feature at every tree node in *DALI-Node-Bag* is named *DALI-Node-Best-Bag*, and selecting only the most accurate tree from *DALI-Tree-Bag* (using the original training set as testing set) is named *DALI-Tree-Best-Bag*. I also report on *DALI* alone, to isolate the feature-construction component, and on *C4.5*-trees (Quinlan, 1994), as a simple decision tree inducer.

Observations For Results on Real-World Domains

Table 5.1 shows results on predictive accuracy after testing all algorithms on a broad set of real-world domains. Each entry value is the average over 50 runs. On each row, the best performance is marked with an asterisk; two asterisks means the system outperforms all others significantly (at the p = 0.05 level). The results can be separated into three different groups. The first group comprises domains where a simple decision tree learner, without any feature construction component, outperforms the rest. The second group favors a global application of multiple-classifier techniques. Finally, the third group favors the local application of these techniques at each node of a decision tree.

Feature Construction Renders Unnecessary

Table 5.1 shows a disadvantage of any form of change of representation when the domain under study is simple (first group). *C4.5*trees outperforms significantly all competitors on the credit and liver disorder (bupa) domains. The difference between *C4.5*trees and *DALI* is of almost 6% points in the credit domain and of 2% points in the liver disorder domain. In average *C4.5*trees shows a difference of almost 4% points when compared to the worst competitor in this group (*DALI-Tree-Boost*). In this group, using feature construction techniques probably leads into data overfitting. A change of representation is unnecessary to capture relevant portions of the structure of the target concept.

Global Application of Multiple-Classifiers

On the second group (Table 5.1), *DALI-Tree-Bag* performs best, showing an advantage in average of 2% over its closest competitor (*DALI-Node-Boost*) and of almost 11% over the worst competitor (*DALI-Node-Bag*). *DALI-Tree-Bag* works particularly well on the promoters and heart domains, significantly outperforming all other algorithms (in the promoters domain by approximately 3% above the closest competitor, and by approximately 10% accuracy above the worst competitor; in the heart domain the difference is of around 2% and 6% respectively). A possible explanation for these results is that an extension of the concept-language representation in decision tree induction —through both the combination of multiple trees and the construction of new features— enables the discovery of the intricate structure of these concepts.

Local Application of Multiple Classifiers

Applying *bagging* and *boosting* locally exhibits a general advantage in the third group of domains. In this case, a change of representation is carried out to improve on each splitting function, but the combination of whole classifiers is omitted (Section 5.3). Within this group, *DALI-Node-Bag* shows the best average performance, with an advantage of almost 14% over *DALI-Tree-Boost*. *DALI-Node-Boost* outperforms significantly all other algorithms in the thyroid (hyper) domain (by less than 1% compared to *C4.5*trees, but by almost 13% compared to *DALI-Tree-Boost*). When compared to *DALI* alone, both *DALI-Node-Bag* and *DALI-Node-Boost* show improved performance on most domains, which justifies the use of multiple-classifier

techniques to improve the quality of the splitting functions on each node of a decision tree.

Overall Results

Overall (last row, Table 5.1), *DALI-Tree-Bag* seems to have the greatest advantage (not always significant). The difference goes up to 3% with respect to *DALI-Node-Bag*. Except for the ionosphere domain, *DALI-Tree-Boost* remains equally competitive. A local use of *bagging* and *boosting* is comparable to a global one, but with the advantage that the design of the algorithm integrates all components into a single mechanism. *DALI-Node-Boost* is on average able to improve over *DALI* alone (difference is less than 1%). The results support the idea that components from different strategies can be combined to achieve some desired functionality. Here, the instability of the feature construction component is amended through multiple-classifier techniques.

Observations

Table 5.1 shows how using different degrees of complexity in the concept language representation results in improved performance over different groups of domains. The highest degree of complexity is observed by using multiple-classifier techniques over whole decision trees. An intermediate level of complexity is to use these techniques at each node of a decision tree. The simplest approach works under the absence of any feature construction component. One reason why a local application of multiple-classifier techniques cannot reach —in some domains– the same performance as a global application does, relates to the notion that refinements over models impose a limit on how far the new algorithm can go (Section 1.2). Using combination of logical expressions as splitting functions in a decision tree modifies a flexible component (Section 3.2), leaving the basic strategy unaltered; inherent deficiencies such as the fragmentation problem continue harming the evidential support of every disjunctive term (Section 4.2). On the other hand, a global application of multiple-classifier techniques augments the space of concept representations. For some domains such expressiveness is necessary to capture the structure of the target concept.

5.5 Conclusions

Results obtained from this case study support the following observations:

Table 5.1: Tests on predictive accuracy for real-world domains.

Concept	C4.5	DALI	DALI-node (global)			DALI-tree (local)		
			boost	bag	best-bag	boost	bag	best-bag
Group 1								
Mushroom	100.0*	99.80	99.88	99.84	99.82	99.92	99.84	99.80
CreditApp	86.36**	80.71	84.17	85.51	80.98	84.31	84.71	80.25
Hepatitis	82.32*	80.18	80.18	82.00	78.55	81.09	82.00	80.00
LiverDis	67.14**	65.18	57.68	66.00	63.94	65.26	65.06	64.23
Average	84.00	81.47	80.48	83.34	80.82	82.64	82.90	81.07
Group 2								
Cancer	95.20	95.68	65.21	96.47	95.26	96.74*	96.23	95.74
Lympho[2]	83.46	87.86	87.71	88.57*	88.00	84.00	88.29	87.71
Lympho[3]	77.54	84.57	87.43	87.00	84.14	85.43	87.57*	85.29
NThyroid [Hyper]	96.78	96.19	84.76	96.10	96.00	97.52**	96.38	96.48
NThyroid [Hypo]	96.40	96.29	86.86	96.38	96.38	95.71	96.67*	96.00
StarCluster	98.40	98.38	68.00	98.48*	98.19	98.19	98.48*	98.29
Average	91.30	93.16	80.00	93.83	93.00	92.93	93.94	93.25
Group 3								
TicTacToe	85.80	98.40	99.73*	98.74	99.28	96.11	93.71	97.77
KrvsKp	96.20	94.87	96.87	97.00*	94.87	96.07	95.60	94.67
Promoters	81.86	83.40	89.00	92.20**	83.20	88.20	87.67	84.67
HeartC	76.16	77.44	79.31	81.17**	75.17	78.83	77.66	75.10
Ionosphere	91.22	91.31	63.97	92.57*	90.86	92.29	53.14	91.52
Average	86.25	89.10	85.78	92.34	88.68	90.30	81.56	88.75
ABS AVRG	88.14	88.96	82.89	90.84*	88.67	89.67	87.36	88.82

- Understanding the effects of learning components enables us to apply these components in different scenarios. This can be useful in the design of learning mechanisms that are guided by functionalities (Section 1.3). As an example, this case study identifies a correlation between the instability of a splitting function mechanism F and the size of the space of all possible splitting functions. A large space increases F's instability because small changes in the examples over a training set T_s, may cause different splitting functions look superior. A search mechanism that explores the space of feature conjuncts (with size exponential in the number of original features) is expected to produce instability on F when applied to variants of T_s. This information suggests how the problem can be rectified through the stabilizing effect of multiple-classifier techniques. Empirical results show how a local application of *bagging* and *boosting* (Section 5.2) at each node of a decision tree denotes an advantage of predictive accuracy

over certain group of domains when compared to other approaches using decision trees (Table 5.1). Since a local application of multiple-classifier techniques results in a single decision tree, the output is amenable to interpretation and analysis.

- Modifying the bias of a decision tree locally, by applying techniques originally conceived to work over whole hypotheses, illustrates that an algorithm's bias should be analyzed at a more refined-level than the whole algorithm itself (Section 1.2). Techniques such as *bagging* and *boosting* have been described in the machine learning literature as offering substantial improvement when applied over learning algorithms. This view is incomplete, not only because it precludes the use of such techniques locally into the internal mechanism of algorithms, but because it disregards the alteration of the algorithm's strategy (Section 3.2). This section shows that multiple-classifier techniques can be seen as components, that they can be integrated internally into the design of new algorithms.

- Section 2.7 explains how one approach to bridge the gap between difficult domains and simple domains is to use transformation operators. The goal is to transform a domain in a way that can be learned by current techniques. Empirical results in this study show that different degrees of representational complexity favor different groups of domains (Table 5.1). A transformation operator, as defined in Section 2.7, could simply consist on dynamically adjusting the complexity of the final approximation according to the domain under study. In other words, one could first devise a way to explore different spaces of representation, before a search over the hypotheses space is conducted. The first step attempts to match the domain under study to that representation that can capture the (simple or intricate) structure of the target concept; the problem is in this way simplified by reducing the learning phase to a search for the hypothesis with best predictive capabilities —once the right degree of representational complexity has been chosen. This idea is explored with more detail in Chapter 6.

Part II

Flexible and Adaptable Representations

Chapter 6

HCL: Domains, Goals, and Strategy

Previous chapters have suggested how the development of new algorithms can be improved by driving the design according to desired goals, underlining the importance of a functional decomposition analysis. The second part of this thesis describes the design, implementation, and testing of a new algorithm. The objective in this second part is to determine how the ideas and concepts laid down in the first part can be applied in a practical scenario. This chapter illustrates the initial steps for the design of *HCL* (Hierarchical Concept Learner). The design embodies unique requirements: no existing model satisfies (simultaneously) the specified goals. The first two basic steps in creating a new approach to learning (Section 3.4) are elaborated in this chapter: analyze the class of domains of study, and define goals and strategy. The other steps are deferred to Chapter 7. The last section in this chapter reviews selected work in the machine-learning literature related to the strategy and functionalities of *HCL*.

6.1 The Class of Domains

Feature Interaction

Real-world domains cover a spectrum of varying *feature interaction* (Section 2.5). On the one side of the spectrum, low interaction means the features describing the examples convey much information about the target concept. In this case, a simple representation suffices to produce a hypothesis space $\mathcal{H}$ comprising good estimations: since features interact in a simple manner, the space $\mathcal{H}$ is expected to be small (i.e. $\mathcal{H}$ reflects a strong bias (Utgoff, 1986)). As an example, knowing when the acceleration a of an object of mass M will exceed certain limit τ, can be expressed as a simple relation involving the force F exerted over the object. Assuming a, M, τ, and F are given, the concept can be simply described as follows:

$$Class = \begin{cases} \text{TRUE} & \text{if } a = \frac{F}{M} > \tau \\ \text{FALSE} & \text{if } a = \frac{F}{M} \leq \tau \end{cases} \tag{6.1}$$

In contrast, high interaction implies each feature must be combined with many other features to unveil concept structure. Here $\mathcal{H}$ is larger than before, since features can interact in very complex manners: the learning process is more complicated because we must find a way to express all possible feature combinations. For example, determining win or lose in chess from raw board-configurations is hard, because the features (board positions) interact in non-obvious ways (Section 2.6).

Domain Characteristics

The design of *HCL* rests on the assumption that the domains under study vary along a wide range of different degrees of interaction[1]. To isolate this characteristic from other possible sources of difficulty, I assume the domains are free of noise (i.e., feature and class values come unperturbed), and that a complete set of relevant features exists (i.e., in theory, the feature set can produce an exact concept definition), although the individual contribution of each feature may be difficult to identify. Similarly, I assume no domain knowledge exists on how features interact other than the information given by the data itself. Table 6.1 summarizes the characteristics of the class of domains for which the *HCL* algorithm is designed.

After identifying the characteristics or properties present in the class of domains under study $\mathcal{S}$, we want to know the degree to which each property is present in $\mathcal{S}$ (Section 3.4). In *HCL*, we shall pay attention to the amount of feature interaction. Since I assume no other sources of difficulty are present, the amount of feature interaction roughly indicates the complexity of the domain under study. An important issue is to determine how *HCL* behaves under domains exhibiting different degrees of complexity. Feature interaction is estimated by a measure of concept variation ∇, explained in Section 2.5. Experiments in Section 9.1 evaluate the performance of *HCL* as ∇ varies (i.e., as the complexity of the domain varies).

Table 6.1: Class of Domains for *HCL*.

Characteristics Present	**Characteristics Not Present**
Various degrees of feature interaction	Noise perturbation of feature or class values.
Features are relevant, although their contribution hard to discern.	Domain knowledge on how features interact.

Discussion

Under the assumption that real-world domains require concept approximations that can deal with different levels of feature interaction, how can we develop new algorithms that are designed to satisfy this requirement? Most common algorithms assume a fixed representation, where simplicity plays a significant role Nevertheless, there are cases where complex interactions must be found (e.g. unfamiliar structured concepts, Section 2.5), as the features representing the examples are too primitive.

If a concept exhibits high degree of interaction, a large number of intermediate terms is necessary to bridge the gap between the input (i.e., original) features and the target concept. The design of an algorithm dealing with all types of feature interaction must consider the necessary steps to dynamically construct these intermediate terms. In this situation, it is more convenient to describe a concept in a hierarchical fashion, by building a structure in which lower layers denote the most simple relations, and upper layers denote increasingly more complex relations.

As an illustration, observe that the hypothesis output by a learning algorithm is often expressed as the logical combination of partial hypotheses; each partial hypothesis comprising a combination of original features. For example, the DNF expression

[1]But where the degree of Kolmogorov complexity is still low (Section 2.3), i.e., a highly compressible representation remains available.

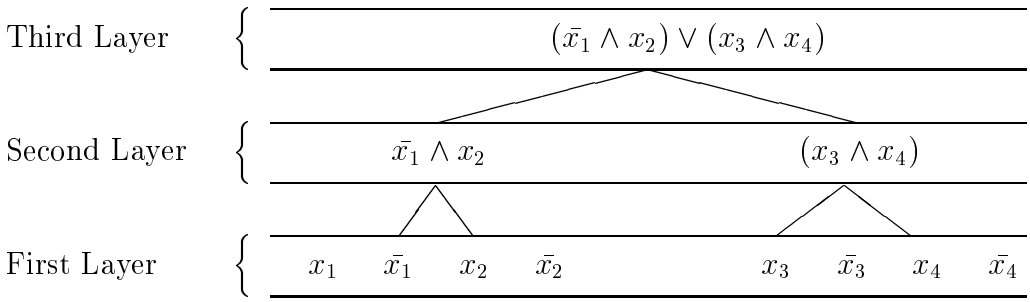


Figure 6.1: Boolean concept $(\bar{x_1} \wedge x_2) \vee (x_3 \wedge x_4)$ represented as a hierarchy of partial subconcepts.

$(\bar{x_1} \wedge x_2) \vee (x_3 \wedge x_4)$ can be formed by first finding the terms $(\bar{x_1} \wedge x_2)$ and $(x_3 \wedge x_4)$, followed by applying logical disjunction. Learning simple disjunctive terms, however, is inadequate when features convey low information about the target concept: relations among features are too complex; a search over the space of original features is too low level (Rendell, 1988). Figure 6.1 shows a DNF expression represented by a multi-layer hierarchy in which original features —and their negations— occupy the first layer, partial hypotheses intermediate layers, and the final concept estimation the last layer. Intermediate concepts serve as input to more complex relations, in this way helping to disentangle the complex structure of the target concept.

6.2 Main Goal and Strategy

The main goal behind the *HCL* algorithm is stated next:

Main goal in *HCL*: To discover the structure of the target concept by finding dependencies among partial subconcepts. The dependencies must be represented in an explicit and intelligible manner.

The goal above differs from other learning approaches in that the dependencies among subconcepts or partial terms is made explicit. Here, we expect the learning mechanism to search for increasingly more complex intermediate terms, dynamically discovering the possible structure of the target concept. To reach this goal, the concept-language representation must shift along different levels of complexity.

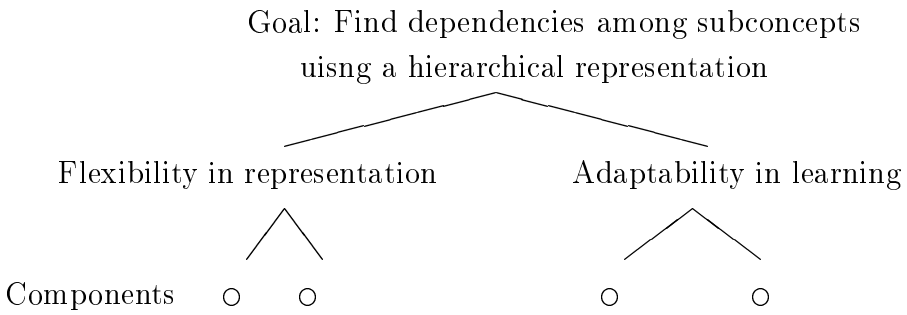


Figure 6.2: The main goal in *HCL* is divided into two subgoals; each subgoal is reached by a specific functionality; each functionality is implemented by a set of components.

Therefore, we must avoid a fixed representation that can capture only a limited number of terms.

The general strategy behind *HCL* offers an approach to attain the main goal:

General learning strategy in *HCL*: To construct subsequent layers in a hierarchy of partial hypotheses that progressively accommodate better approximations to an underlying target concept. The bottom of the structure corresponds to all original features, whereas the top formula in the hierarchy denotes the output hypothesis. Intermediate layers represent increasingly more complex partial hypotheses.

The following sections show how the strategy above requires at least two functionalities in the algorithm design: *flexibility* in the concept-language representation, and *adaptability* in the learning process. Figure 6.2 shows how each of these functionalities contributes to reach the main goal (Section 1.3). Flexibility enables the algorithm to accommodate more complex expressions at each new hierarchical level (first subgoal); adaptability provides the means to identify when enough levels have been constructed, according to the degree of feature interaction displayed by the domain under analysis (second subgoal). Each of these functionalities (defined next) is implemented through a set of components (Section 7.1).

Flexibility

I use the term *flexible*, to refer to any learning algorithm constantly introducing a change of representation in the concept-language representation. An algorithm capable of dynamically building a hierarchical structure as the one described above is flexible: the final hypothesis is not restricted to a simple combination of original features, but rather builds over these features to generate more complex expressions. Flexibility in the concept-language representation is an important functionality because it allows exploration of different levels of complexity.

The amount of complexity in the induced hypothesis is commonly fixed by the concept language representation. Since the necessary amount of complexity is ultimately determined by the class of domains of study, an algorithm should be able to vary the set of potential hypotheses along a spectrum of different complexities.

Adaptability

A flexible representation is only useful if the learning algorithm is able to find, i.e., to adapt to, the degree of complexity that maximizes the predictive capabilities of the final hypothesis. Learning under this view is equivalent to a search for the best available representation.

Augmenting the sophistication in the description of the final hypothesis, however, may reach the point of data overfitting. This is expected if the domain is characterized by low feature interaction, on which complex representations are not necessary. At the other extreme, some domains may need a complex representation for accurate classification. How can we adapt to various needs of representation? One approach is to apply a component that can automatically detect stop increasing the complexity of the generated hypotheses. In *HCL* this will require knowing when to stop adding levels on top of the hierarchical structure.

This chapter has considered two steps that are part of the definition of goals and strategies (Section 3.4): define a concept-language representation (in *HCL* a hierarchical structure of partial terms), and search for partial subconcepts in that representation (in *HCL* the construction of new terms at each hierarchical level). Under the chosen representation, flexibility and adaptability are two functionalities that can combine to reach the main goal. The next section reviews related work

in the machine-learning literature. Studying the advantages and disadvantages of each related method will prove helpful in both the analysis and definition of *HCL*'s mechanism.

6.3 Related Work

The implementation of an algorithm following the strategy of *HCL* (Section 6.2) could rely on existing approaches to learning inference networks, e.g., a neural net, a Bayesian network, etc. In most of these models, however, the structure of the network is defined before learning starts. Few algorithms exist that construct this structure dynamically (Baffes, 1992; Fahlman & Lebiere, 1990; Hirose, Yamashita, & Hijiya, 1991; Romaniuk, 1994, 1996). A similar argument holds for symbolic learning algorithms where a fixed representation is commonly assumed, although a few exceptions exist (Fu & Buchanan, 1988; Rendell, 1985; Wogulis & Langley, 1989). The rest of this section reviews selected work that builds hierarchical structures allowing flexibility and/or adaptability during learning.

Neural Networks

A neural network is trained by adjusting the weights attached to each link connecting adjacent nodes in the hierarchical structure. A common approach is to adopt a hill-climbing search over the weight space until a minimum is reached in the performance metric (e.g., squared differences between the network output and the true output). Neural networks exhibit at least two major disadvantages: slow convergence to a solution, and the need to specify in advance the network topology. The last requirement forces the learning process to adopt a fixed representation, which might not match to the complexity required by the domain under study (due to high feature interaction).

Cascade Correlation

One of the first attempts to imbue flexibility in a connectionist framework is the cascade-correlation architecture (Fahlman & Lebiere, 1990). The general mechanism works as follows. Initially, the input features are fully connected to all output nodes, yielding a one-layer structure. If no significant error reduction is observed after certain number of epochs (and performance is still below the expectations), new hidden units start to be integrated to the network. This change to the original

concept-language representation adds flexibility to the learning process, effectively increasing the complexity of the hypothesis. Such flexibility responds to a need for high-order feature detectors as suggested by the data.

Each additional unit creates a new one-unit layer. The new unit is connected to all input features and previously defined units, its weights adjusted to maximize the correlation between the new unit's output and the error observed at each network-output unit. The new unit is then simply considered as one more input feature —its weights frozen. The process stops adding new units until the error is small (or no additional improvement is observed). In this sense, adaptability (Section 6.2) is dictated by the degree of error reduction on the training set.

Back-Propagation Varying the Number of Hidden Units

A different approach to adjust the topology of the neural network dynamically varies the number of hidden units using back-propagation (Hirose et al., 1991). This approach begins with a fixed network structure; the need for new hidden units is triggered when the network becomes trapped in a local minimum (i.e., when the total error remains the same after certain number of epochs). The algorithm is trained using back-propagation (Rumelhart et al., 1986). Training continues after the new hidden unit is added; the change in the weight space is expected to overcome the local minimum. Flexibility in the concept-language representation, as in Cascade Correlation, is achieved by adding new units to the network. In addition, a pruning mechanism is invoked at the end of the training, to eliminate hidden units that cause no increase in accuracy. Thus, adaptability in selecting the right configuration is maybe more complex than *HCL*, because it relies on a refining mechanism. Similar to Cascade Correlation, the process stops when no more error reduction is observed on the training set.

Learning with Evolutionary Growth Perceptrons

Incorporating the notions of evolutionary computation, Romaniuk (1994, 1996) proposes to increase the size of a neural network by adding new hidden units, each competing to attain best training performance. One hidden unit is created at each new layer, its inputs connected to all previous original features and previously lower-layer units. Competition is established by viewing each proposed unit as a chromosome characterized by a performance value over the training set. Using the common

operators of reproduction, cross-over, and mutation, a population of units (trained each via the perceptron algorithm) is evolved. The process continues until all output units have been trained. Although the mechanism to generate each new unit differs from the approaches above, flexibility and adaptability remain alike: the former is attained by adding new units to the network, whereas the latter stops adding new units until enough error reduction on the training set is observed.

Symbolic Methods

In symbolic learning[2], categorical data is admissible, and the output hypothesis is commonly amenable to interpretation. Each learning model specifies a concept-language representation over which the hypothesis space is built, e.g., decision trees, rule-based systems, instance-based learners, etc. Here I describe several approaches where the initial representation can change, yielding a hierarchy of new intermediate terms.

Learning a Hierarchical Knowledge Base

Fu and Buchanan (1988) suggest a method to build a hierarchical knowledge base by finding intermediate terms. Their approach varies according to whether partial knowledge is available or not. If partial knowledge is available (albeit incomplete), the learning process exploits the existing information to either construct new terms, advancing upwards in the hierarchical structure (bottom-up learning), or downwards (top-down learning). Search can also proceed in a bidirectional way. If no partial knowledge is available, the system proceeds by constructing a taxonomy tree based on a measure of similarity (conversely dissimilarity). Experimental results show the benefits obtained when intermediate knowledge is incorporated: system prediction is significantly improved.

Flexibility is achieved by constructing new intermediate terms. Adaptability is conditioned on how much partial knowledge is available: the addition of new terms proceeds until high-level terms are reached, and depends on how much knowledge initially exists.

---

[2] Other areas in learning may employ a hierarchical structure to represent intermediate concepts, e.g., conceptual clustering (Fisher, 1987; Hadzikadic & Y., 1989; Michalski & Stepp, 1983). This thesis, however, deals with inductive learning exclusively, as defined in Section 2.1.

Constructive Induction and Layered Information Compression

Rendell (1985) describes a model of induction with three levels of knowledge structures. At the lowest level, examples in the instance space sharing similar class value (and close to each other) are coalesce into regions. In case features are too primitive, further compression of class-membership information from neighboring regions in instance space leads to formation of patterns (or clusters). Next, similar patterns coalesce into pattern classes, merging possibly distant, but homogeneous regions of instance space. The highest level uses transformation operators (induced or suggested by domain knowledge) to create pattern groups, which can predict unobserved instance space regions by transformation of similar structures discovered at lower levels. Each level involves a different form of data compression: 1) gather examples with similar class value into regions, 2) gather regions into patterns, and 3) gather patterns into groups. This approach differs from others in that each layer introduces a new concept-language representation leading to further information compression.

Flexibility is observed at each new level, through the incorporation of increasingly more abstract knowledge structures. Adaptability is dependent on how primitive the original features are, and is limited to three knowledge levels.

Transforming a Domain Knowledge with Intermediate Concepts

Wogulis and Langley (1989) report on a system that refines a knowledge base by constructing internal concepts. The learning process is similar to that of explanation-based learning (Mitchell, Keller, & Kedar-Cabelli, 1986; DeJong & Mooney, 1986). The system begins with an initial domain theory and new instances. No changes occur if a new instance is explained by the theory. If an instance is not explained by the theory, it is re-expressed with the existing concepts and generalized. Each generalization could (potentially) become an internal concept. Empirical evidence shows that, in most cases, adding intermediate concepts saves time and improves efficiency. Flexibility is observed by extending the domain theory with new subconcepts; adaptability is contingent to the remaining instances not yet explained by the theory: the theory continues being refined until all instances can be explained.

6.4 Discussion

The systems above share the common goal of changing the concept-language representation to improve performance. In all of them, the goal is to explore different degrees of complexity in the hypothesis, setting aside the common assumption of simplicity in the representation that pervades most algorithms. In neural networks, flexibility in the representation consists of adding new nodes to the hierarchical structure, although the methodology to add the new nodes may differ with each algorithm (e.g., by using evolutionary computation (Romaniuk, 1994, 1996), or by adding new nodes to avoid a local minimum (Hirose et al., 1991)). In symbolic methods, flexibility is based on constructing new terms in the knowledge structure. Similar to neural networks, different paths may be followed to build and integrate the new terms (e.g., by using partial knowledge (Fu & Buchanan, 1988), or by searching for information compression (Rendell, 1985)). Thus, although different mechanisms have been explored to change the concept-language representation, an invariant operation is to add new terms or nodes into the structure of the hypothesis.

Despite improved accuracy in classification, the systems above have shown successful results in only a limited number of domains. Moreover, the domains in which good performance has been reported (here called $\mathcal{S}^*_{\text{dynamic}}$) differ from the class of domains in which algorithms employing a fixed representation perform satisfactorily (here called $\mathcal{S}^*_{\text{fixed}}$)[3]. It may seem that adding flexibility in a learning mechanism switches good performance from one class of domains to another, but that it is impossible to learn both $\mathcal{S}^*_{\text{dynamic}}$ and $\mathcal{S}^*_{\text{fixed}}$ simultaneously. Section 2.4, however, explains how the goal of inductive learning is to find concept approximations that lead to accurate classification within the set of all structured domains $\mathcal{S}^*_{\text{structured}}$. Therefore, it is in theory possible to learn both classes of domains with reasonable performance, as long as both belong to $\mathcal{S}^*_{\text{structured}}$ (i.e., $\mathcal{S}^*_{\text{dynamic}} \in \mathcal{S}^*_{\text{structured}}$ and $\mathcal{S}^*_{\text{fixed}} \in \mathcal{S}^*_{\text{structured}}$). It is, however, common to observe an algorithm L showing above-average performance on either class of domains, but displaying a significant degradation of performance on the other class. To amend this situation, a first step is to construct algorithms

[3]The distinction between these two classes is similar to the one made before between simple, $\mathcal{S}^*_{\text{simple}}$, and difficult $\mathcal{S}^*_{\text{difficult}}$, domains in Section 2.7.

showing average performance on both $\mathcal{S}^*_{\text{dynamic}}$ and $\mathcal{S}^*_{\text{fixed}}$. Otherwise, without knowing the nature of the target concept in advance, it becomes difficult to decide under what conditions is the algorithm applicable or not(Section 1.2).

The ideas above suggest that other functionalities, besides flexibility alone, must be explored before an algorithm is able to extract useful patterns in domains where different degrees of complexity are required. Changing representation is more than creating new nodes or terms: inherent difficulties arise that must be solved before good results are obtained. Two difficulties associated to any change of representation are listed next.

- A difficulty associated with the addition of new nodes or terms in a concept structure is the risk of data overfitting. Simple domains need not have complex representations to accurately approximate the target concept (Matheus, 1990). Augmenting the complexity of the representation may in some cases be unnecessary.

- The transition from one hypothesis structure, h, to the other, h', may be too coarse. In other words, it is possible that the right complexity in the representation lies somewhere between h and h', in which case we would fail to produce an accurate estimator.

As explained before, flexibility alone does not guarantee good performance. This occurs because the necessary degree of complexity in the concept-language representation may denote, among different domains, significant variation. Therefore, a crucial step within a learning framework is to know when to stop adding new terms, i.e., to adapt to the right concept representation. The systems described above use different approaches to adapt to the data; a common technique is to add new terms until enough reduction in training error is obtained. I claim this approach is limited for the following reason: the goal of a learning algorithm is to generate a hypothesis (i.e., classifier) that generalizes beyond the observed examples. Verifying the efficiency of the algorithm by checking performance on the training set is prone to underestimate the true error rate (i.e., the asymptotic error rate as the sample approximates the entire population). An alternative is to guide this search by obtaining a better estimation of the true error rate (e.g., via some resampling technique: cross-

validation, repeated subsampling, bootstrap, etc.). The next chapter explores these ideas in the *HCL* algorithm.

Chapter 7

HCL: Mechanism and Implementation

Section 3.4 enumerates and explains four basic steps in creating a new approach to learning. The previous chapter (Chapter 6) covered how the first two steps (analyze the class of domains of study and define goals and strategy) are applied to the *HCL* algorithm. Here I elaborate the last two steps: isolate individual components and assemble components into a final sequence. In essence, this chapter details the mechanism of the *HCL* algorithm. In the next sections I separate the implementation of the two main functionalities in *HCL*: flexibility and adaptability. Here, the functional decomposition analysis that helps to analyze individual components is mainly analytical (contained in Chapter 8 is an empirical study). The last section explains how all components are integrated.

7.1 Implementing Flexibility

Before being able to look at *HCL* from a global perspective, I first separate flexibility and adaptability; this section addresses flexibility. *HCL* augments the complexity of the concept-language representation at each new level of a hierarchical structure. Each level represents a hypothesis to the target concept exhibiting higher complexity than the hypothesis at the lower below. A new hypothesis (at each layer) is constructed from the integration of three components (Figure 7.1):

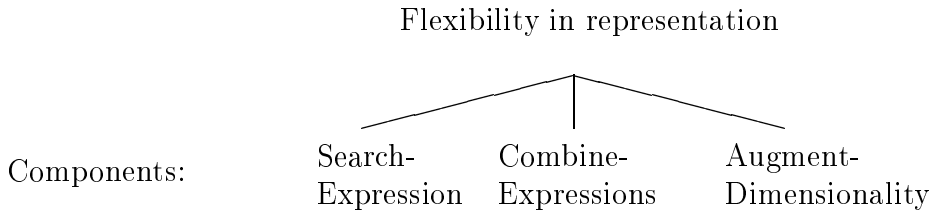


Figure 7.1: Flexibility in *HCL* is attained through three main components.

1. A **Search-Expression Component** that constructs a new logical expression as a function of the original features and previously defined terms.

2. A **Combine-Expressions Component** that creates a linear combination of logical expressions (each expression obtained from component 1), to create a full hypothesis to the target concept at the current hierarchical level.

3. Lastly, an **Augment-Dimensionality Component** that integrates the expressions from component 2 into the set of original and previously defined terms, to augment the number of potential operands in the construction of new expressions at upper layers of the hierarchy.

A new hierarchical level is created by executing these components; the resulting hypothesis is a linear combination of logical expressions, and the set of potential operands to construct new terms is augmented with each of the linearly combined expressions. Figure 7.2 outlines how a new hypothesis —representing a hierarchical layer— is constructed. The first two components (Search-Expression and Combine-Expressions) were briefly discussed in Chapter 5 and shown to be effective when incorporated at every tree node of a decision tree. The same components are used in *HCL*, but within a different learning strategy (Section 6.2). The last component (Augment-Dimensionality) becomes crucial to attain flexibility; its role in the general algorithm is explained later on.

An Example

Before a more detailed analysis of each of these components is given, we shall first look into a practical example. Figure 7.3 shows the result of applying the three

Algorithm 5: Generate New Hypothesis
Input: Training Set T, Original features and previously defined expressions L_{all}
Output: New Hypothesis H
GENERATE_HYPOTHESIS(T, L_{all})
(1) Let COMBINE_EXPRESSIONS generate a linear combination of
(2) new expressions by calling SEARCH_EXPRESSION(L_{all}) on
(3) multiple variants of T
(4) Let H be the final linear combination of expressions
(5) **return** H

Figure 7.2: The mechanism to build a new hypothesis at each hierarchical level.

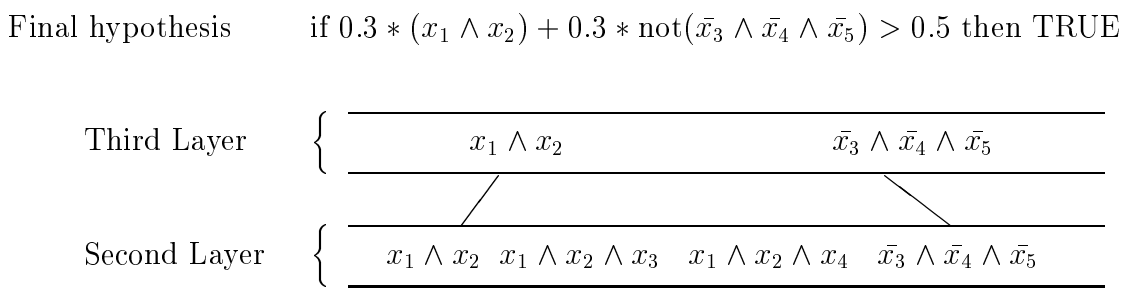


Figure 7.3: Applying *HCL* over boolean function $(x_1 \wedge x_2) \wedge (x_3 \vee x_4 \vee x_5)$.

components above over boolean function: $(x_1 \wedge x_2) \wedge (x_3 \vee x_4 \vee x_5)$. This function could also be represented as $(x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_4) \vee (x_1 \wedge x_2 \wedge x_5)$.

At the second hierarchical layer (the first layer is occupied by all original features and their negations), the Search-Expression component is responsible for creating the observed logical expressions. The Combine-Expressions component creates a linear combination of these expressions to form a hypothesis to the target concept (too long to be shown in the figure). The Augment-Dimensionality component adds each of these expressions into the pool of original features, to use them as operands in the construction of new expressions.

At the third hierarchical layer the system simply reproduces the two most relevant expressions found on the previous layer. The Combine-Expressions component outputs the final hypothesis displayed at the top of Figure 7.3. The final linear combination is equivalent to the conjunction of expression $(x_1 \wedge x_2)$ with expression $\text{not}(\bar{x_3} \wedge \bar{x_4} \wedge \bar{x_5})$. The latter expression is the disjunction of three terms using conjunction and negation as logical operators.

Together, these three components are responsible for shifting the degree of complexity of the representation while constructing an approximation to the target concept at each layer in the hierarchical structure. The next sections explain each component in detail.

Searching for Logical Expressions

Figure 7.4 outlines the Search-Expression component (Figure 7.2). New logical expressions are constructed by conducting a beam search over the space of all boolean-

Algorithm 6: Search Mechanism for New Expressions
Input: Original features and previously defined expressions L_{all}
Output: Best expression F_{best}
SEARCH_EXPRESSION(L_{all})
(1) Let L_{bool} be the list of literals from L_{all}
(2) (i.e., boolean features or expressions and their complements)
(3) $L_{\text{beam}} \leftarrow$ best literals in L_{bool}
(4) **while** (**true**)
(5) $L_{\text{new}} \leftarrow$ Systematically form the conjunction
(6) of every $F_i \in L_{\text{beam}}$ with every $F_j \in L_{\text{bool}}$
(7) Apply pruning into L_{new}
(8) **if** $L_{\text{new}} = \emptyset$
(9) **return** global best combination F_{best}
(10) $L_{\text{beam}} \leftarrow$ best combinations in L_{new}
(11) **end while**

Figure 7.4: The search mechanism outputs the best logical feature combination (i.e., best monomial).

feature conjuncts or monomials (i.e. conjunction of boolean features or their complements). The general idea is to add one literal (i.e., boolean feature or its complement) at a time to the best previously retained expressions in the beam (the beam starts with the best single literals). Adding literals extends the depth of the search; retaining the best expressions limits the width of the search. The process continues while keeping track of the best global expression F_{best}. An example of the space of all possible expressions on three boolean features is illustrated on Figure 7.5. To avoid exploring all possible states (i.e., expressions), the size of the search space can be constrained with three operations:

1. A systematic search to avoid redundant expressions (Rymon, 1993; Webb, 1995), Lines 5-6, Fig. 7.4. Each expression F_i conjoins several boolean features (or their complements), e.g., $F_i = x_1 \bar{x}_3 x_5$. Because conjunction is commutative, the search space is defined by avoiding any F_j that is identical to F_i except for the order in which features appear, e.g., $F_j = \bar{x}_3 x_5 x_1$ (Figure 7.5). The number of all possible new expressions grows exponentially with the depth of the search, but is limited by the number of combinations that remain still feasible as deeper subspaces are explored. In general, the number of possible expressions at depth k is $2^k \times \binom{n}{k}$, where n is the number of original features.

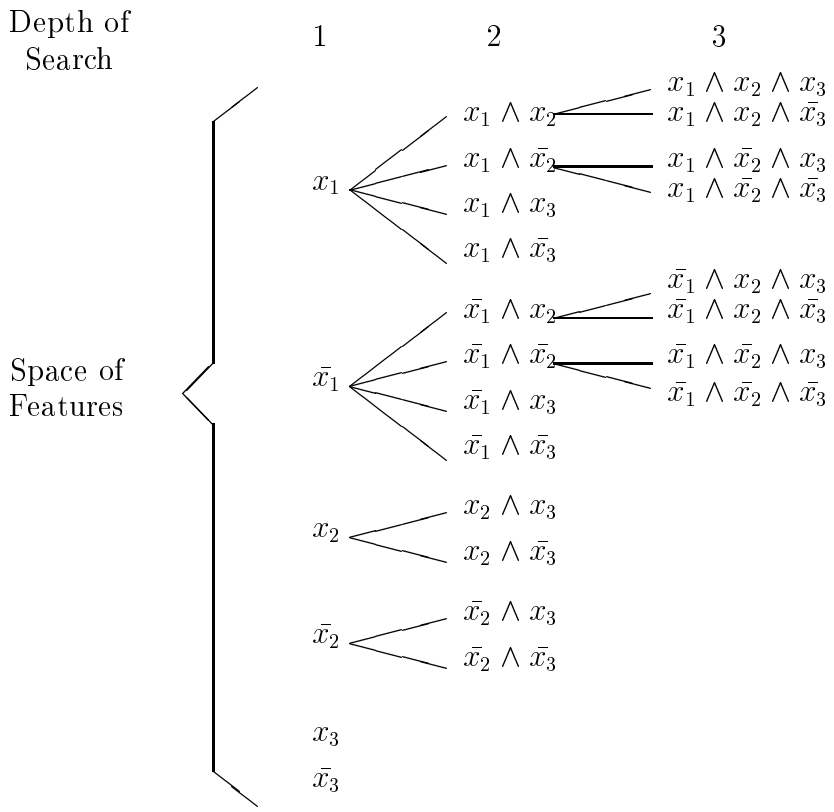


Figure 7.5: A systematic search in the space of new expressions or terms.

2. A pruning technique (Webb, 1995; Quinlan, 1995) (Line 7, Fig. 7.4). Define F_{best} as the best explored expression according to some impurity measure H (e.g., entropy), such that, for all explored F_i, $H(F_{\text{best}}) < H(F_i)$. As long as H is monotonic, F_i can be eliminated if the best value it can ever attain along its search path – according to H – is worse than F_{best}.

   The rationale behind this pruning technique is explained as follows. Denote (p_{F_i}, n_{F_i}) as the number of positive and negative examples covered by expression F_i respectively. The best performance F_i can ever attain is to conjoin until producing an expression having coverage-coordinates: $(p_{F_i}, 0)$ or $(0, n_{F_i})$ (because conjunction can only make the coverage more specific, i.e., conjunction is identical to set intersection). The elimination of unpromising expressions proceeds by deleting every F_i such that the performance values at $(p_{F_i}, 0)$ and $(0, n_{F_i})$ are both worse than the value of H at the coverage-coordinates of F_{best}.

3. A dynamic beam-width. At each depth of search, only the best expressions are selected to expand the search (Lines 3 and 10, Fig 7.4). Specifically, only

those expressions having a value of H within $\delta = \log_2(1 + \frac{1}{|T|})$ from the best combination at that level are retained; $|T|$ is the number of training examples. The value δ is simply a measure of the uncertainty that arises when the degree of impurity H is estimated on a sample of size $|T|$. Large training samples guarantee a confident estimate (i.e., reduce the uncertainty factor). Conversely, small training samples give large uncertainty factors, to compensate for the associated lack of statistical support.

The search above terminates when no more expressions are left on the beam, at which point the best explored expression, F_{best}, is output.

Analysis

The Search-Expression component above has several advantages and disadvantages:

- The depth and width of the beam search are set dynamically (operations 2 and 3 respectively). This is advantageous over a search limited by user-defined parameters, because without knowing the target concept in advance, the user has no means to provide good approximations to these limits (as for example in (Ragavan & Rendell, 1993)). Furthermore, overestimating these limits increases the computational cost significantly.

- Representing a concept through logical expressions adds interpretability to the output hypothesis: an important advantage of symbolic learning models over other inductive models (e.g., over a connectionist model). A desired characteristic of a learning algorithm is to preserve our ability to extract knowledge or discover patterns relevant to the domain under study.

- A search based on adding one feature at a time to previously defined expressions may experience difficulties in discovering complex feature interactions (Pérez, 1997). This is because the contribution of each feature to the target concept is hard to detect when multiple features combine in complex ways. Nevertheless, I expect real-world domains exhibiting a degree of complexity (Section 2.3) that can be tackled with this search mechanism.

- A second disadvantage is that a single logical expression exerts axis-parallel partitions over the instance space; for some domains (e.g., multiple numerical

Algorithm 7: Combining Multiple Expressions
Input: Number of iterations k, Training set T
Output: Linear combination of expressions H
COMBINE_EXPRESSIONS(k,T)
(1) Let D_1 assign equal weight to every example in T
(2) **foreach** $i = 1 \cdots k$
(3) Set current distribution to D_i
(4) Generate variant T^i from T according to D_i
(5) Apply SEARCH_EXPRESSION into T^i to generate
(6) expression F_i
(7) Let a_i measure the error of F_i on T
(8) **if** $a_i = 0$
(9) Exit-Loop
(10) Let D_{i+1} assign more weight to those examples
(11) incorrectly classified by F_i
(12) **end for**
(13) Let $H = \{a_i F_i\}$
(14) **return** H

Figure 7.6: Expressions are linearly combined following the *boosting* algorithm.

features), this may be undesirable. As explained in the next subsection, this is alleviated through the combination of multiple expressions.

Empirical tests assessing the contribution of this component to the entire mechanism are reported in the next chapter (Section 8.4).

Combining Expressions

HCL builds a hypothesis for the target concept through the linear combination of multiple logical expressions. The Combine_Expressions component is called at each new hierarchical level (Figure 7.2), and employs a multiple-classifier technique known as *boosting* (Freund & Schapire, 1996, 1995), described in Chapter 5. Section 5.3 explains how multiple expressions can be combined to improve classification at each node of a decision tree. The algorithm describing how expressions are combined is shown again in Figure 7.6. The idea is to use replicates of the training set to generate different expressions. Different from Section 5.3, each new level in *HCL* combines expressions from replicates of the whole training set, as opposed to subsamples covered by the current tree node. Each new replicate is generated by sampling with replacement the original training set under a probability distribution. The probability distribution changes for each new replicate, increasing the weight for those examples incorrectly classified by the previous expression. The goal is to combine expressions

to produce an accurate decision rule. Following I analyze the *boosting* algorithm and discuss its advantages and limitations.

Analysis

Generating a new hypothesis by combining logical expressions using *boosting* has several advantages and disadvantages:

- A single logical expression exerts axis-parallel partitions over the instance space. When a domain is characterized by many numeric features, or high feature interaction, this kind of partitioning becomes inappropriate. A linear combination of expressions is a more effective representation to delimit irregular space regions (Brodley & Utgoff, 1995; Murphy & Pazzani, 1991; Rendell, 1988).

- *boosting* is not only effective in reducing variance, but also improves the algorithm bias (see Section 7.3). The variance factor is reduced by voting over several expressions: it reduces the risk of relying on a single hypothesis that is not representative of the hypotheses obtained from different training samples. Bias reduction comes from several sources:

  - the mechanism is able to progressively focus on those examples harder to classify. By adopting a resampling scheme in which more weight is assigned to incorrectly classified examples, an increase in example density is observed on those space regions for which the underlying algorithm experiences more difficulty. This facilitates the job of delimiting regions comprising examples with similar class value.

  - the final hypothesis is obtained by reducing *predictive error*, estimated from the original training sample. As more samples are generated, the distribution of examples differs every time more from a uniform distribution over the training set T. Therefore, calculating error over T is a form of estimating the true error of the most recent hypothesis (Breiman, 1996a). This is one way to deal with data overfitting, because the generated expressions are not tailored to adjust to the original training sample: they are trained on a sample that differs from the original sample.

- As explained in Section 4.2, the design of an algorithm must ensure that every partial hypothesis is properly supported by the set of all available examples

(i.e., by the training set). Failing to accomplish this may result in the fragmentation problem (e.g., in decision-tree learners). *Boosting* is able to support each disjunctive hypothesis only *partially*, for the following reason. Once the first expression F_1 has been constructed, the resampling technique concentrates on those examples misclassified by F_1. Under the new distribution, a different training sample is created, which results on expression F_2. Assuming both F_1 and F_2 are part of the target concept, then the examples in the intersection of both expressions, $\mathrm{COV}(F_1) \cap \mathrm{COV}(F_2)$, have a high probability of not being present in the training sample from which F_2 is generated, because they were correctly classified by F_1. Thus, F_2 is left with less statistical support. Moreover, the process continues taking away examples lying in the intersection of partial hypotheses on each new iteration (i.e., on each new training sample).

Augmenting Operands for the Construction of New Expressions

After a hypothesis has been generated, a third component augments the number of possible operands to create new expressions at upper hierarchical levels. The introduction of intermediate concepts changes the description language so as to simplify the learning task for the underlying algorithm.

In *HCL*, a change of representation occurs when each of the combined expressions from Combine_Expressions is added into the set of original features and previously defined expressions. The place where this component is invoked in *HCL* is explained in Section 7.3 (Figure 7.9, line 11). The set of potential operands is increased at every new hierarchical level; new expressions build over these operands to bridge the gap between low-level features and highly abstract concept representations.

Analysis

The simplification in the learning process obtained by augmenting the instance space dimensionality with more informative expressions is conditioned on the distribution of examples over the instance space, which most current systems assume to be uniform, smooth, or locally invariant (Rendell, 1986). As an example, Figures 7.7a and 7.7b illustrate two instance spaces with different example distributions. Figure 7.7a denotes a simple example distribution where features are informative. On the other hand, primitive features produce irregularity throughout the instance space, as shown in Figure 7.7b. Irregular instance spaces are characterized by many single

class regions for which similarity-based learning methods (i.e., methods following a similarity-based approach, Section 2.5) are inadequate; the distribution of examples is not uniform and each single region must be identified and delimited. Augmenting the instance space dimensionality tends to merge dispersed regions into new hyperplanes. In the last case, an instance space characterized by an irregular distribution is gradually tamed by the introduction of new informative expressions. These expressions become operands for the construction of new expressions in subsequent layers.

The effect of this component is to augment the dimensionality of the instance space; the new dimensions should bring together examples of similar class value, ultimately smoothing the example distribution. As depicted in Figure 7.7c, adding an expression that captures important interactions that characterize the concept sampled in Figure 7.7b transforms the instance space by *merging dispersed regions* into a new dimension (Rendell & Seshu, 1990). The resulting space simplifies the distinction of regions with similar class value. This contrasts to learning models where the representation is fixed, and where no attempt is made to transform the (possibly) non-uniform space. An appropriate bias shift not only yields smaller, more concise hypotheses, but often also better hypothesis accuracy, due to the stronger support of partial concepts now represented in a more informative way (Section 4.2).

Adding more informative expressions as new operands exhibits at least two major disadvantages:

- An irregular distribution of examples in the instance space can only be tamed if the incorporated expressions are able to distinguish significant parts of the target concept. As explained in Section 2.7, inductive learning can be seen as a transformation process, in which the application of components simplifies classification. Adding new terms is a form of external transformation that augments the instance-space dimensionality. The effectiveness of constructive induction as a transformation process depends not so much in the addition of new terms, but in the quality of the terms being incorporated. Thus a key element to smooth the example distribution lies mainly in the feature-construction component.

- Another drawback is that, unless the potential new expressions are adequately constrained, the construction process often results in an exceedingly large space;

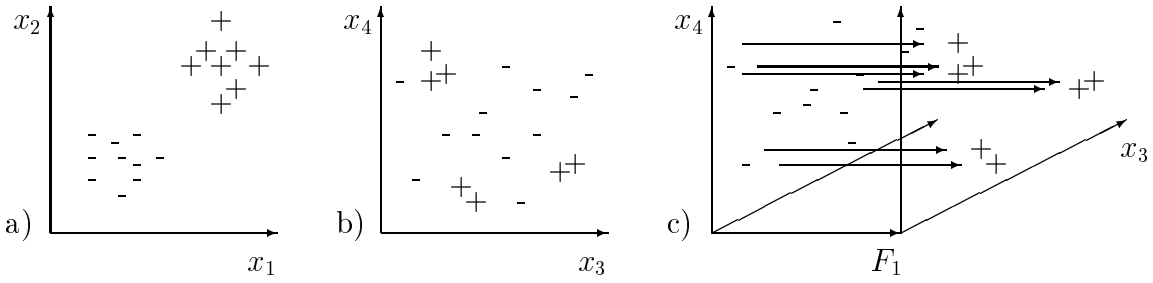


Figure 7.7: (a) and (b) denote instance spaces with regular and irregular distributions respectively. (c) denotes a transformation of the space in (b) where the inclusion of a new informative feature, F_1, simplifies the distinction between positive (+) and negative (-) examples.

in the general case the space of new expressions grows exponentially in the number of operands.

7.2 Implementing Adaptability

Previous sections analyze how a flexible representation is attained; this section explains how flexibility can be adapted to the right degree of complexity. *HCL* is an algorithm that generates increasingly more abstract hypotheses. But, when does the mechanism know that the right hypothesis has been found? When does it stop generating more hypotheses (i.e., hierarchical levels). As explained before (Section 6.2), augmenting the sophistication in the description of the final hypothesis may reach the point of data overfitting. Some domains are characterized by a simple representation, whereas other domains require complex expressions.

One way to implement this bias-detection component is to assess the performance of each hypothesis. *HCL* employs a resampling technique known as stratified 10-fold cross validation (Kohavi, 1995) to measure the predictive accuracy of each hypothesis. Figure 7.8a shows how adaptability is achieved with one component alone. The component detects when no additional improvement is observed, at which point the mechanism stops adding new levels. The hypothesis with highest predictive accuracy is selected for classification of unseen examples. Any learning algorithm with this capability is adaptable, because the complexity of the hypothesis is automatically adjusted according to the characteristics displayed by the domain under study. Figure 7.8b depicts this adaptation process as the search for a maximum on a function that maps hypotheses ordered by increasing complexity against predictive accuracy. The process stops when the curve shows no tendency to grow i.e., when the next

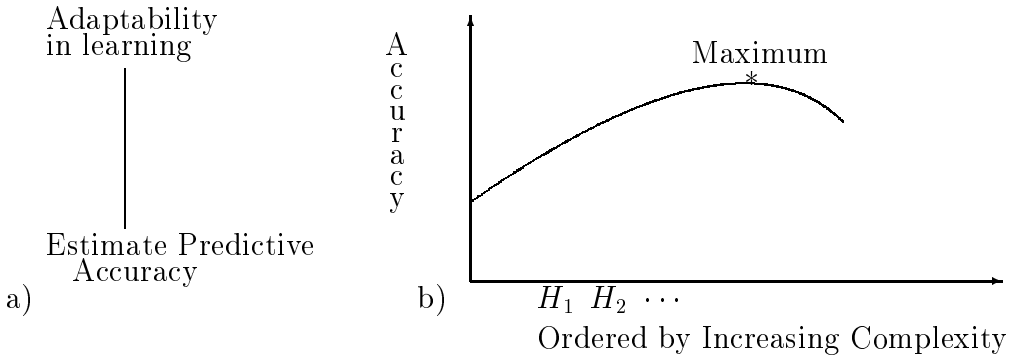


Figure 7.8: a) Adaptability in learning is attained by a single component estimating predictive accuracy at each level. b) This component is equivalent to a search for the right hypothesis that looks for a maximum on a function mapping complexity of the concept-language representation against predictive accuracy.

hypothesis (i.e., next level) shows a decrease in accuracy. Figure 7.9 illustrates the role of this component in *HCL*.

Analysis

Of the several known methods for accuracy estimation, one must aim for a method having low bias and variance. To explain this, let ϵ be the predicted error (rate) in classification —probability of incorrectly classifying a new instance sampled from the same distribution as the training set— of certain estimation method. Two factors affect the value of ϵ: the first factor specifies the degree to which ϵ differs from the true classification error (i.e., the error if all instances were known), also known as the *Bias* in error prediction. The second factor tells how much ϵ varies among different values, known as the *Variance* in error prediction. Estimation methods exhibit different behavior according to these two factors:

- Bootstrap methods (Bailey & Elkan, 1993; Efron & Tibshirani, 1993; Efron, 1983; Kohavi, 1995) are non-parametric maximum likelihood estimators that create new example sets by sampling with replacement the original training set. These methods have been reported to have low variability but large bias in their estimates (Bailey & Elkan, 1993; Kohavi, 1995). Thus, the estimate may be close to the true error, but repeated trials show high variability in the results.

- Repeated Subsampling (Weiss & Kulikowski, 1990; Kohavi, 1995) separates the dataset into a training set and a testing set; the final estimation averages the error in classification over the testing sets after the experiment is repeated multiple times. In this case, more instances used for training lower the bias of the estimate, but few examples on the testing side unavoidably increase the variance.

- The leave-one-out method (Kohavi, 1995; Shao, 1993; Zhang, 1992) separates one example at a time for testing, using the rest for training (averaging over all results). This method has been reported to be almost unbiased but causes unacceptably high variance. Moreover, the method is *inconsistent*, in that the estimate is not guaranteed to select the best model as the sample size approximates that of the entire population (Shao, 1993).

- Cross-validation methods (Number of folds k, $10 \leq k \leq 15$) divide the training set into k different folds. Iteratively, one fold is left out for testing while the others are used for training; the final estimation averages over the k estimates. (Breiman et al., 1984; Kohavi, 1995; Shao, 1993). In general, cross validation methods are approximately unbiased , although exhibiting high variance (they have been reported to perform better than leave-one-out for model selection (Breiman & Spector, 1992)). Nevertheless, a form of cross validation that keeps a similar proportion of class values on both the training and testing sets —known as stratified cross validation, reduces the variability of the standard approach while maintaining low bias (Kohavi, 1995).

As described above, selecting a method for accuracy estimation results in a trade-off between bias and variance. *HCL* employs stratified 10-fold cross validation to select the hypothesis with best predictive capabilities. Experimental evidence favors this method over other approaches.

7.3 The Final Arrangement

Now that each component has been identified and analyzed, we are in position to understand the mechanism behind *HCL*. Figure 7.9 provides a general view of the algorithm. The main idea is to keep generating new hypotheses that build upon

Algorithm 8: The *HCL* Algorithm
Input: Training set T
Output: Final Hypothesis H_f
HCL(T)
(1) $L_{\text{all}} \leftarrow$ original features
(2) $H_f \leftarrow$ NULL
(3) BestAcc $\leftarrow 0.0$
(4) **while** (**true**)
(5) Estimate accuracy of GENERATE_HYPOTHESIS(T,L_{all})
(6) Let Acc represent the estimated accuracy
(7) **if** (Acc > BestAcc)
(8) $H_f \leftarrow$ GENERATE-HYPOTHESIS(T,L_{all}) using
(9) all examples in T for training
(10) BestAcc $\leftarrow$ Acc
(11) $L_{\text{all}} \leftarrow L_{\text{all}} \cup \{f_i : f_i \in H_f\}$
(12) **else**
(13) Exit-Loop
(14) **end while**
(15) **return** H_f

Figure 7.9: The *HCL* algorithm.

original features and previously defined expressions until predicted accuracy starts to decline. Section 7.1 explains the mechanism behind Generate_Hypothesis, in which a new hypothesis is constructed through the linear combination of logical expressions. At each hierarchical level (i.e., each cycle in the while-loop), *HCL* estimates the accuracy of the hypothesis that would result from combining expressions that build over the current set of constructive operands (i.e., original features and previous expressions). If the estimated accuracy improves over the estimation of the hypothesis at the level below, a new hypothesis is generated by calling Generate_Hypothesis over the whole training set (since accuracy estimation was obtained by training on the smaller cross-validation partitions). Each of the combined expressions is then incorporated into the set of previously defined expressions to serve as operands for the construction of new expressions.

The Design of *HCL*

A different view of *HCL* is given in Fig 7.10 through a flow diagram depicting how components interrelate. An explicit account of the different components —and their interactions– leads to a better understanding of the design, and suggests on ways to improve the basic strategy. The design of *HCL* can be modified in two ways:

- By altering one or more of the components. As explained in Section 3.2, a learning algorithm can be altered without touching the basic strategy (i.e., by modifying the flexible and additional components only). For example, new expressions may be constructed using different feature-construction mechanisms; expressions can be combined using different multiple classifier techniques; different methods to estimate accuracy can be tried; etc. In all these cases the basic design remains unaltered: a hierarchical structure builds increasingly more abstract subconcepts; the process continues by estimating the predictive accuracy of the hypothesis generated at each new level.

- By accommodating components into different sequences. The basic strategy can be modified by deciding on different component arrangements, as a response to possible limitations and disadvantages. For example, Section 2.4 explains two domain characteristics for which most current algorithms are unprepared: high feature interaction, and the existence of patterns along separate regions of the instance space. The design of *HCL* aims at learning domains displaying different degrees of interaction (first characteristic), but ignores how to cope with patterns unfolding on separate regions. If we were to arm *HCL* to cope with the second characteristic, the basic design would require a totally new configuration.

7.4 Discussion

This chapter has covered the last two basic steps in creating a new approach to learning (Section 3.4): isolate individual components, and assemble components into a final sequence. We have seen the specific components helping to attain the two major functionalities in the design of *HCL*: flexibility (Section 7.1), and adaptability (Section 7.2). In addition, we have seen how components interact in the design (Section 7.3). From the perspective of a user or researcher interested in creating his own learning algorithm, however, little has been said about how components are selected to reach certain functionalities. What criteria can be followed to make the right choice? How can we decide on the right sequence to arrange the selected components? This section explains the rationale behind the implementation for *HCL*;

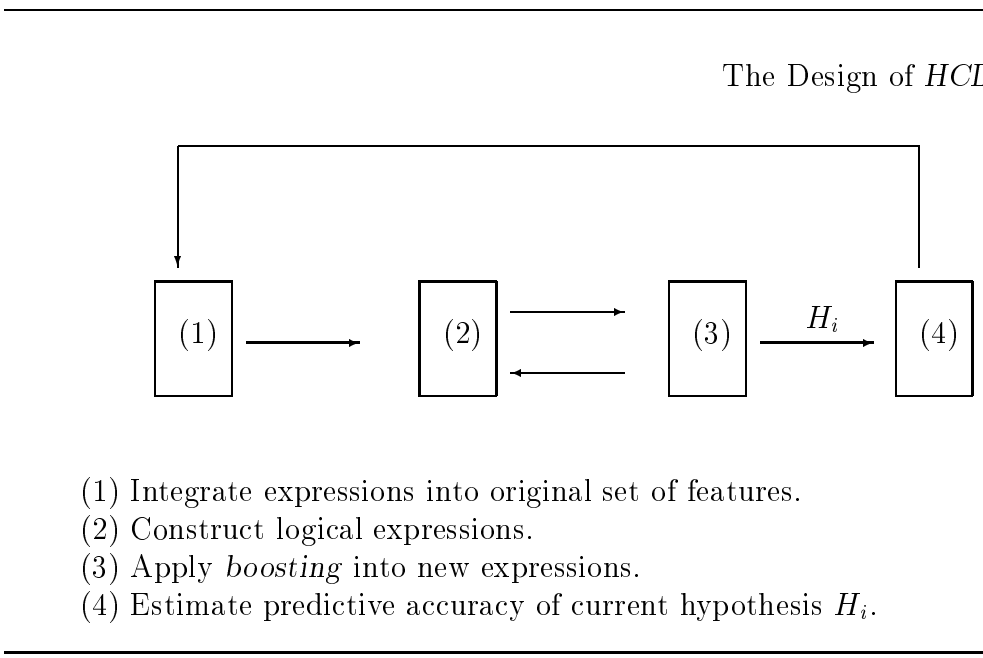


Figure 7.10: A flow diagram for the design of the *HCL* algorithm.

the goal is to give insight on how to select the right components once a goal and strategy are defined.

The Bias and Variance of the Estimate

HCL can be analyzed from a similar perspective to Section 7.2, where methods for accuracy estimation were distinguished by the bias and variance of the estimate. An analogous argument exists to distinguish between learning algorithms (e.g., between parametric and non-parametric learning models). In this case, we denote the bias of a learning algorithm L as the difference between the classification of the hypothesis (i.e., classifier) output by L, and the classification of the target concept[1]. High bias occurs when the induced model fails to approximate the data correctly; variance comes from the instability in L, as defined in Section 5.2, in the sense that small variations in the training set produce different hypotheses. Thus the error of a hypothesis is a combination of bias plus variance.

[1]Different approaches exist to decompose bias and variance on zero-one loss functions (Breiman, 1996c; Kohavi, 1996).

In general we are interested in non-parametric models, where no assumptions exists to model the data. The problem associated with the construction of non-parametric models is the size of the space of potential solutions. In a neural network, for example, increasing the number of neural nodes reduces bias, since higher complexity in the model allows approximation of the target function, but increases variance because of the size of the space of potential networks (Geman, Bienenstock, & Doursat, 1992). Similarly, k-nearest-neighbor algorithms reduce bias as k becomes smaller, since the local information around the example for which a class is to be predicted acquires more relevance; but variance increases, because the algorithm is now prone to producing different predictions stemming from possible variations around the example. The problem can be stated as follows: we can either select a model apriori and run the risk of high bias, or make no model assumptions and run the risk of high variance. How this and other problems are attacked in the mechanism of *HCL* is explained next.

The Rationale Behind *HCL*'s Mechanism

There are at least three concerns that arise in trying to build an algorithm like *HCL*: the bias-variance problem, how much interpretability is obtained in the final hypothesis, and the risk of data overfitting. The goal of this section is to show the rationale behind *HCL*'s mechanism, by explaining how each component fits into a strategy that attempts to solve the problems above. The idea is to exemplify the kind of criteria needed in selecting the right components while constructing a new learning algorithm.

The Bias-Variance Problem

With regard to the bias-variance problem (previous section), each hierarchical level in *HCL* produces a hypothesis where low variance is expected, because of the multiple-classifier technique (*boosting*), which tends to reduce the instability produced from small variations in the training set. Stability is attained by voting over multiple expressions. Nevertheless, the bias of the hypothesis at lower levels may be high, because of the simplicity in the representation of partial concepts. Augmenting the number of operands in the construction of new expressions increases the complexity

of the hypotheses at upper levels, which we expect to reduce the bias[2] (because more models exist to fit the data).

Interpretability

HCL outputs a hierarchical structure representing how different subconcepts interrelate. A matter of concern is if we could interpret the final hypothesis, including the dependencies among partial concepts below the level at which the final hypothesis exists. If the concept is complex, such that many levels are needed between the primitive features and the final hypothesis, we would at least like to identify relevant portions of the final structure. To that end, each node in the hierarchical structure of *HCL* is constructed using logical operators. Each node is an expression obtained from the logical combination of original features and previously defined expressions, and thus amenable to interpretation. Under difficult domains, this component may at least allow for partial identification of the target concept.

Overfitting the Training Data

A common problem in learning comes from measuring error reduction over the training data alone. Such technique is prone to underestimate the true error rate. A feature-construction algorithm tends to overestimate the complexity of the final hypothesis while searching for new terms (Section 6.4). To tackle this problem, *HCL* estimates the off-training accuracy of the most current hypothesis using a method known as stratified 10-fold cross validation. The final hypothesis shows the maximum reduction in predictive error among the evaluated hypotheses.

This section has shown the criteria used to select the components in *HCL*'s mechanism. The goal is been to show how the design of a new algorithm can be guided by goal functionalities; such goal functionalities may respond to common problems in inductive learning. The next chapter will demonstrate to what extent the mechanism in *HCL* is able to address the concerns mentioned above.

[2] Assuming the variance profile is not too steep (Schuurmans, Ungar, & Foster, 1997)

Chapter 8

A Functional Decomposition Analysis

This chapter evaluates *HCL* by isolating and testing its constituent components. The goal is to discern the contribution that each component (or group of components) makes to the entire mechanism. The results help to assess how well each component serves to attain the desired functionalities (e.g., flexibility in the representation, adaptability in the degree of representational complexity), and how each component assists in solving common problems in inductive learning (e.g., bias-variance problem, interpretability, data overfitting). Experimental data shows results in terms of multiple performance criteria, gathered by testing *HCL* over thousands of runs. Experiments are controlled to avoid undesirable sources of error; the class of domains under study comprises a broad range of artificial concepts (Appendix A). In a separate set of experiments (Chapter 9), *HCL* is compared with standard learning models on both artificial and real-world domains.

8.1 Introduction

HCL introduces two major functionalities in its design: flexibility in the concept-language representation, and adaptability in choosing the right amount of representational complexity (Section 6.2). From a functional-design view, one may ask how well these functionalities or goals are fulfilled by the current mechanism, which directs us to the following questions:

- Considering the general strategy and design, what is the expected performance of the algorithm?
  HCL is designed to vary the complexity of the final hypothesis. Therefore, the algorithm is expected to show good performance on those domains where a complex representation becomes indispensable (i.e., a simplicity assumption does not hold, Section 2.5). In other cases, the algorithm should perform similar to models assuming simplicity in their representation. Feature-construction

techniques are not always necessary, especially if the domain has low variation: a complex representation leads into data overfitting. *HCL* is designed to cope with this problem by detecting how much complexity is needed according to the characteristics displayed by the domain studied.

- What is the contribution of each component in *HCL*?
  To answer this question we must analyze *HCL* internally, to decide on the efficiency of its components. It may not always be true that each component is exerting the right operations to reach a specific functionality (Chapter 4). The contribution of each component (or group of components) in *HCL* is assessed here through a functional decomposition analysis. The goal is to identify limitations at the component level to gain insight into how to improve the current mechanism. Future work (Chapter 10) contemplates adjustments to those components where improvements are possible, and to replace certain components by others (i.e., to change the basic strategy).

About The Experiments

This chapter reports on controlled experiments designed to answer the questions above. Each reported value is the average of 50 runs. Graphs displaying learning curves show error bars denoting 95% confidence intervals. In every table, numbers enclosed in parentheses represent standard deviations. Experiments employ artificial domains displaying various degrees of concept variation ∇ (∇ explained with detail in Section 2.5). These domains comply with the design characteristics of *HCL* (Section 6.1): feature and class values are free of noise, a complete set of relevant features exists, no domain knowledge is available, and different degrees of feature interaction are allowed (feature interaction estimated through ∇). Artifical concepts are divided in two groups: 9-feature concepts and 12-feature concepts. Each of these groups covers a wide range of different values for ∇. The goal is to observe how performance is affected using different degrees of concept variation (i.e., feature interaction), and how the effect changes under different training-set sizes.

8.2 Flexibility

Flexibility in *HCL* is attained by increasing the complexity of the hypothesis at each new level of a hierarchical structure (Section 7.1). The components implementing

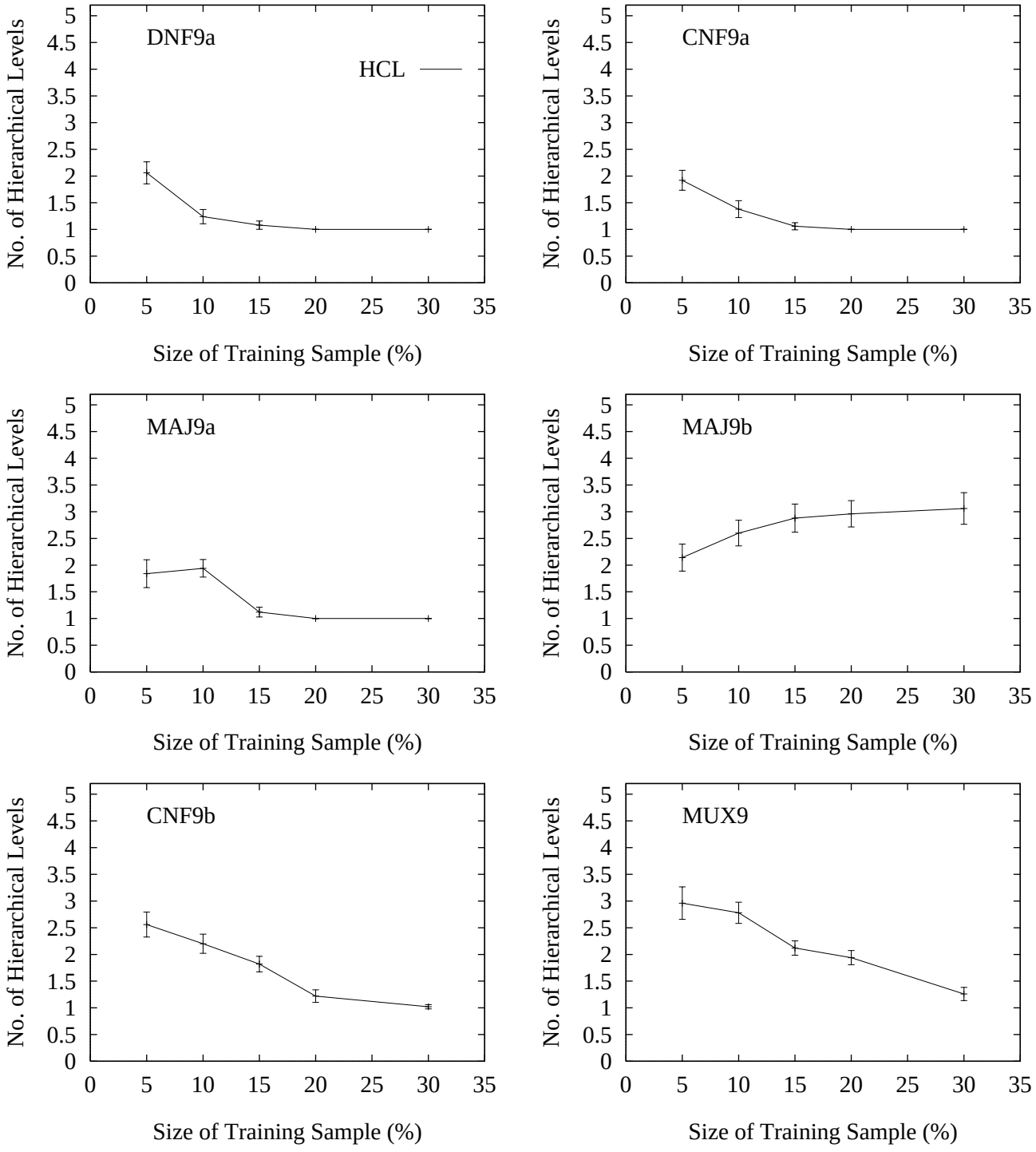


Figure 8.1: Training-set size vs. no. of hierarchical levels in *HCL*.

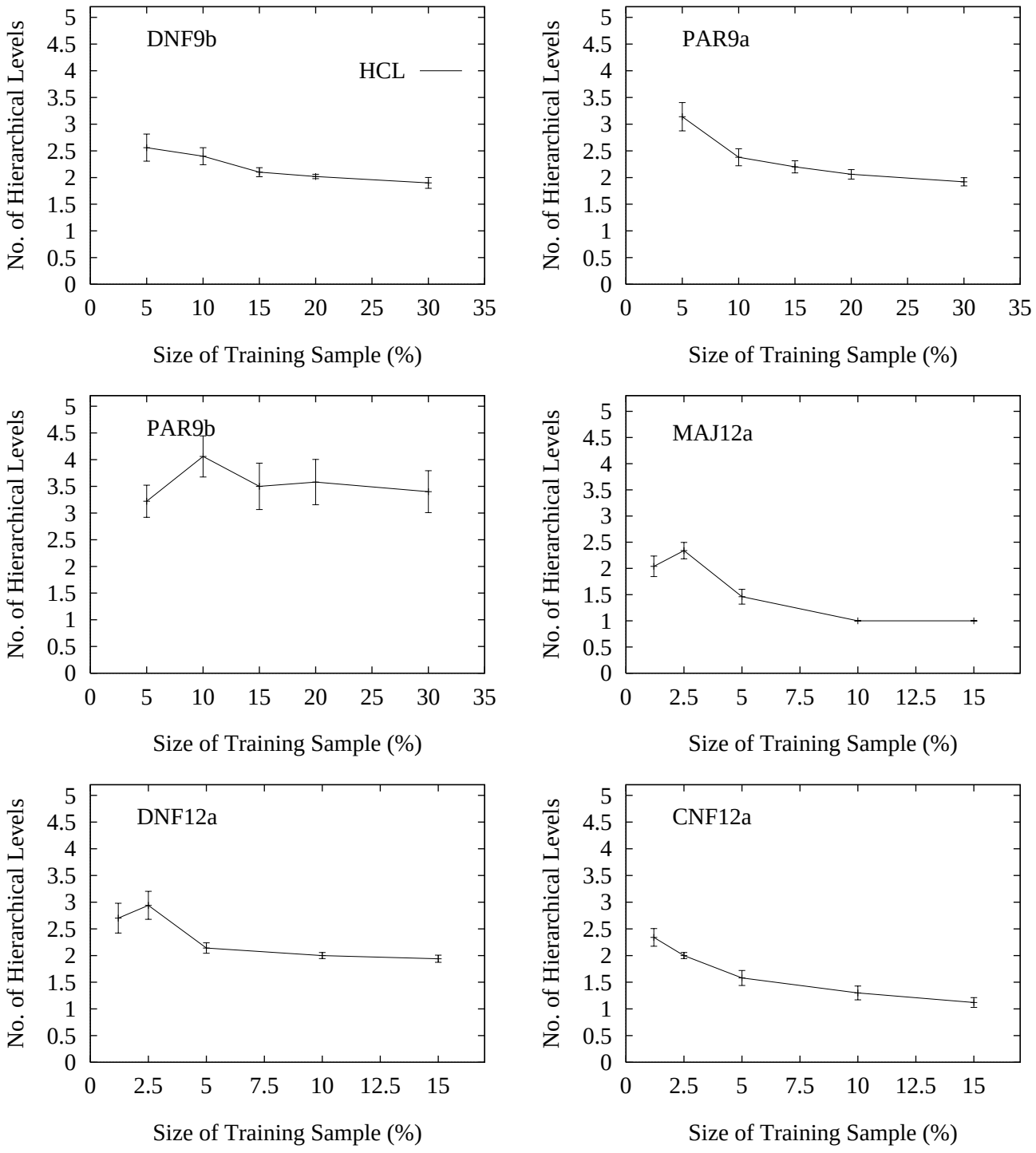


Figure 8.2: Training-set size vs. no. of hierarchical levels in *HCL*.

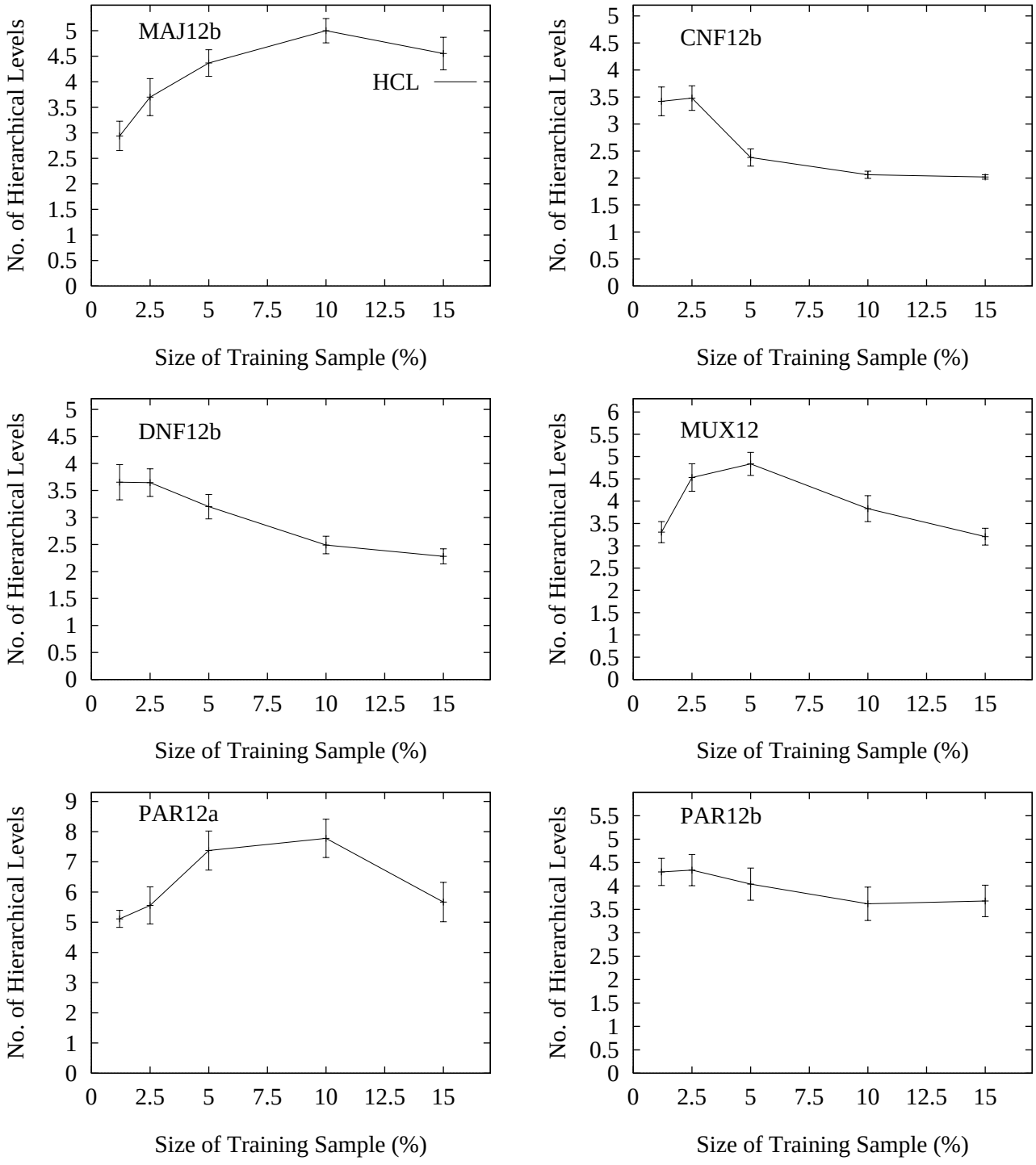


Figure 8.3: Training-set size vs. no. of hierarchical levels in *HCL*.

this functionality work as follows: create new logical expressions that build over previously defined features and expressions; expressions are then linearly combined to form a hypothesis at each new hierarchical level; finally, each expression is added into the pool of previous features and expressions to augment the dimensionality of the instance space.

To measure the effectiveness of the flexible representation introduced by *HCL*, Figures 8.1, 8.2, and 8.3 plot training-set size against the number of hierarchical levels constructed by *HCL* to reach the final hypothesis on each artificial concept. The goal is to assess the effectiveness of the combination of all components integrating this functionality. The concepts are shown on increasing variation (first for 9-feature concepts and then for 12-feature concepts). Plots indicate the level at which the final hypothesis is selected (the algorithm explores in fact one more level in which a decrease of predictive accuracy with respect to the hypothesis at the level below is detected (Section 7.2).

In general, the results show how small training sets require more hierarchical levels than large training sets. Most often the number of levels tends to stabilize between the range of 10% and 15% training-set size. *HCL* needs more intermediate terms to build a final hypothesis when few examples are available. This is to be expected, since small training sets give rise to a larger number of models that can fit the data. With less statistical evidence to support the target concept, the algorithm must look into many more combinations of expressions before producing a final hypothesis. Section 2.2 explains how a learning domain or situation is not only determined by the target concept itself. Other factors characterizing the domain are the size of the training set and the probability distribution from which the examples are drawn. Hence, the complexity of a domain partly depends on the size of the training set.

The behavior described above is not observed in all domains. For some functions the results show an increase in the number of levels as the training set is made larger (e.g., MAJ9b, MAJ12b, PAR9b, MUX12, and PAR12a). Concept variation in these cases is generally high ($\nabla > 20$). The results show how concepts with high variation need, in general, more levels than concepts with less variation, the effect being more evident with a larger set of primitive features. For small training samples a variation of less than 0.20 commonly results in few levels (approximately 2 hierarchical levels for 9-feature functions and 2.5 levels for 12-feature functions), whereas a variation greater

than 0.20 results in a larger number of levels (approximately 3 levels for 9-feature functions and 5 levels for 12-feature functions; PAR12a reaches almost 8 levels with a 10% training-set size). An explanation for this is that as the number of examples increases, the algorithm is able to discover the complex patterns characterizing these domains, constructing more levels on top of the hierarchical structure.

Overall, flexibility in *HCL*'s design effectively responds to circumstances where a change of representation is needed: high degree of variation and small training samples. The controlled nature of the domains under study leave these as the only potential sources of difficulty; *HCL* increases the number of hierarchical levels whenever these sources are present. Flexibility alone, however, is not enough to find the right degree of complexity in the representation (Section 6.2). The mechanism requires an adaptable component that can detect when enough levels have been constructed. The next section addresses adaptability in the design. Flexibility and adaptability are coupled in *HCL* as main design functionalities.

8.3 Adaptability

An adaptable system is able to vary the complexity in the concept-language representation while avoiding data overfitting. Adaptability in *HCL* is carried out by establishing a stopping criterion to the number of hierarchical levels constructed during learning (Section 7.2). The value of this functionality in *HCL* can be tested in two different ways: by measuring how well the stratified 10-fold cross-validation component approximates the true accuracy, and by comparing the performance of *HCL* with a modified version of the algorithm in which only one hierarchical level is built (called *HCL*-1-level). The goal here is to determine the gains obtained by incorporating adaptability in the algorithm design. Both tests are described next.

The Value of the Cross-Validation Component

The first set of experiments regarding adaptability compare three types of accuracy (corresponding to the final hypothesis output by *HCL*): training-set accuracy, the estimated accuracy by the 10-fold cross-validation component, and the true accuracy (i.e., the accuracy over the testing set). The purpose of this test is to determine if the cross-validation component is biased in estimating the true accuracy, and if so, under what conditions. The performance of this component is key to finding a

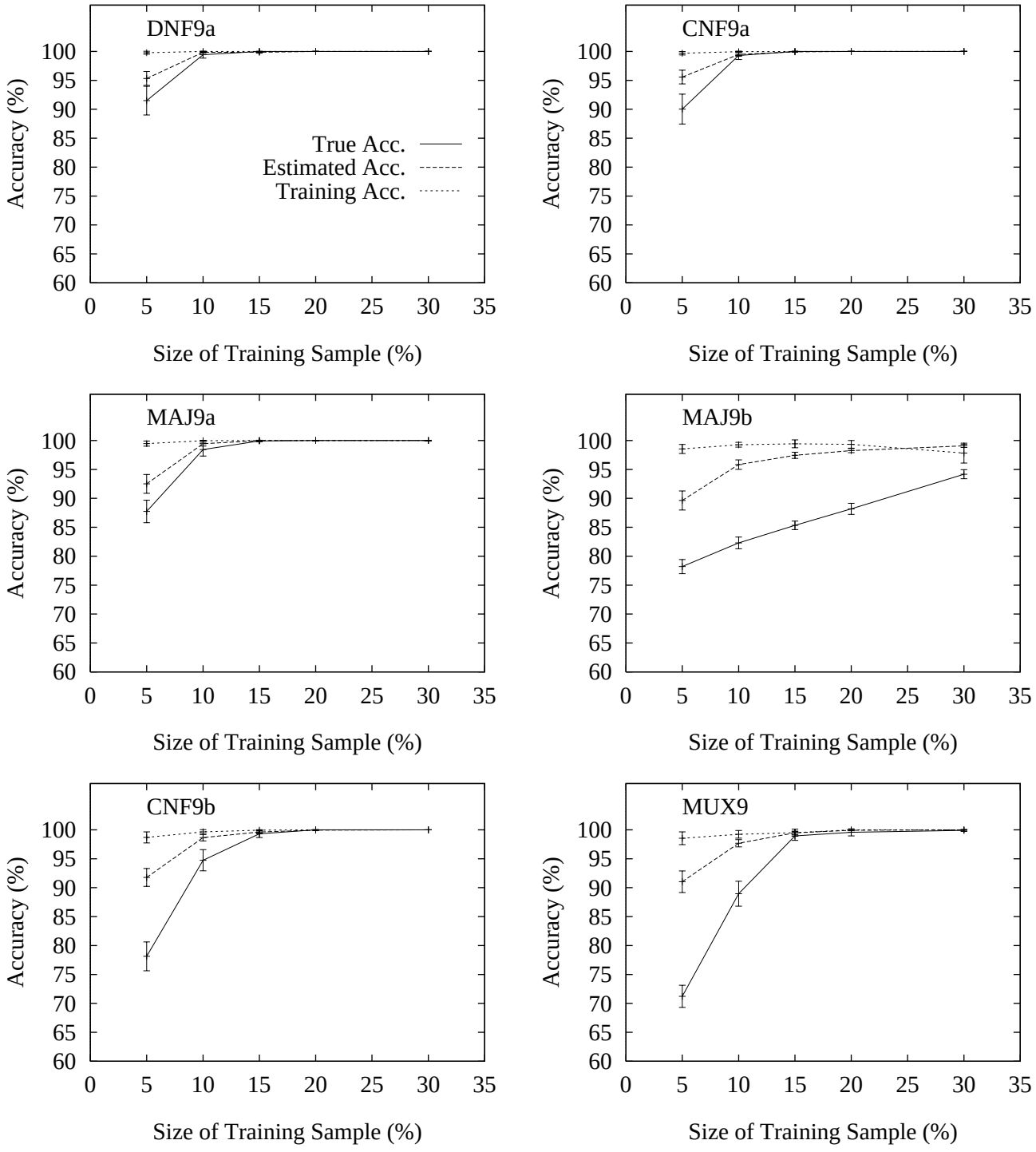


Figure 8.4: Training-set size vs. true accuracy, estimated accuracy, and training accuracy in *HCL*.

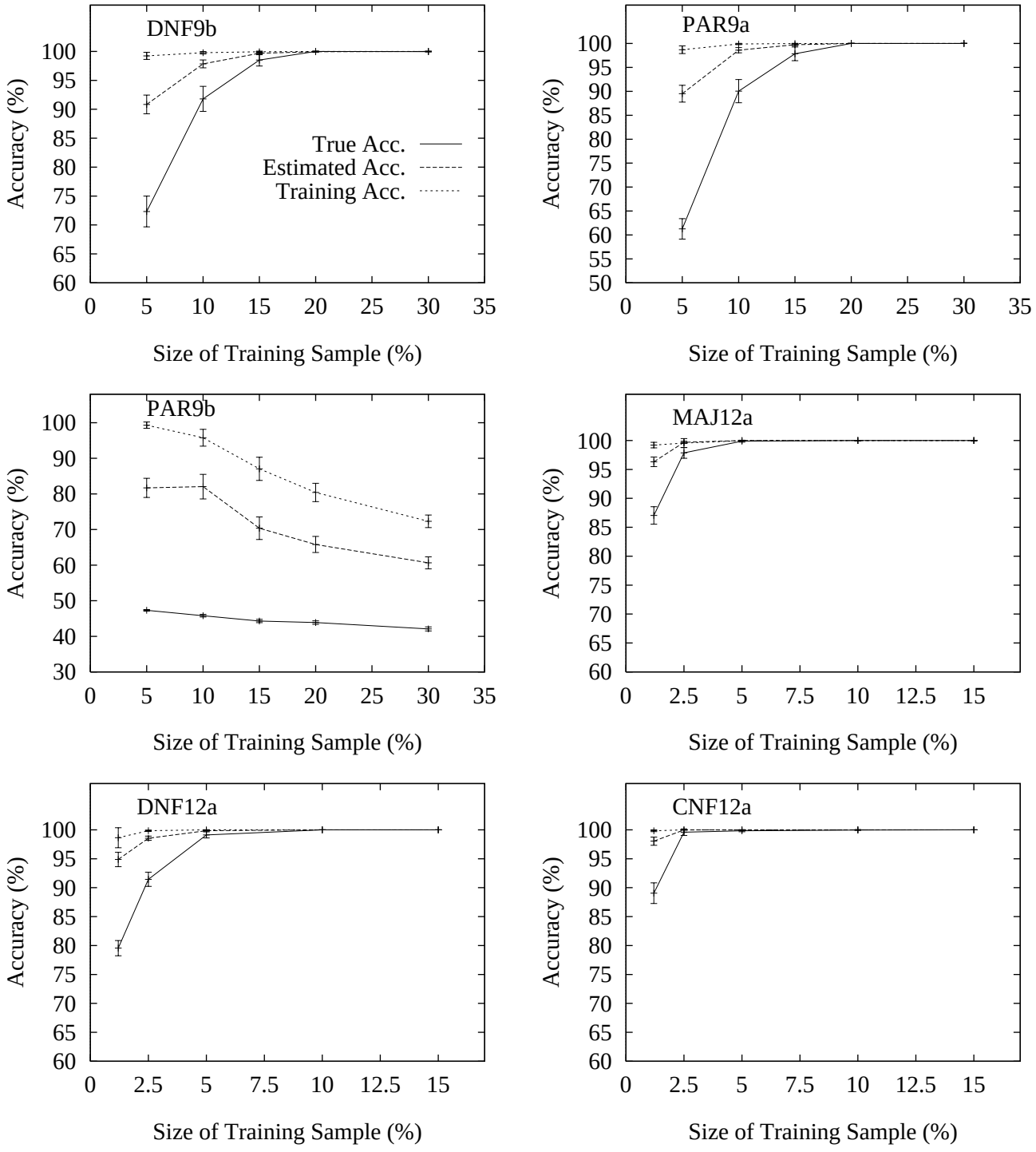


Figure 8.5: Training-set size vs. true accuracy, estimated accuracy, and training accuracy in *HCL*.

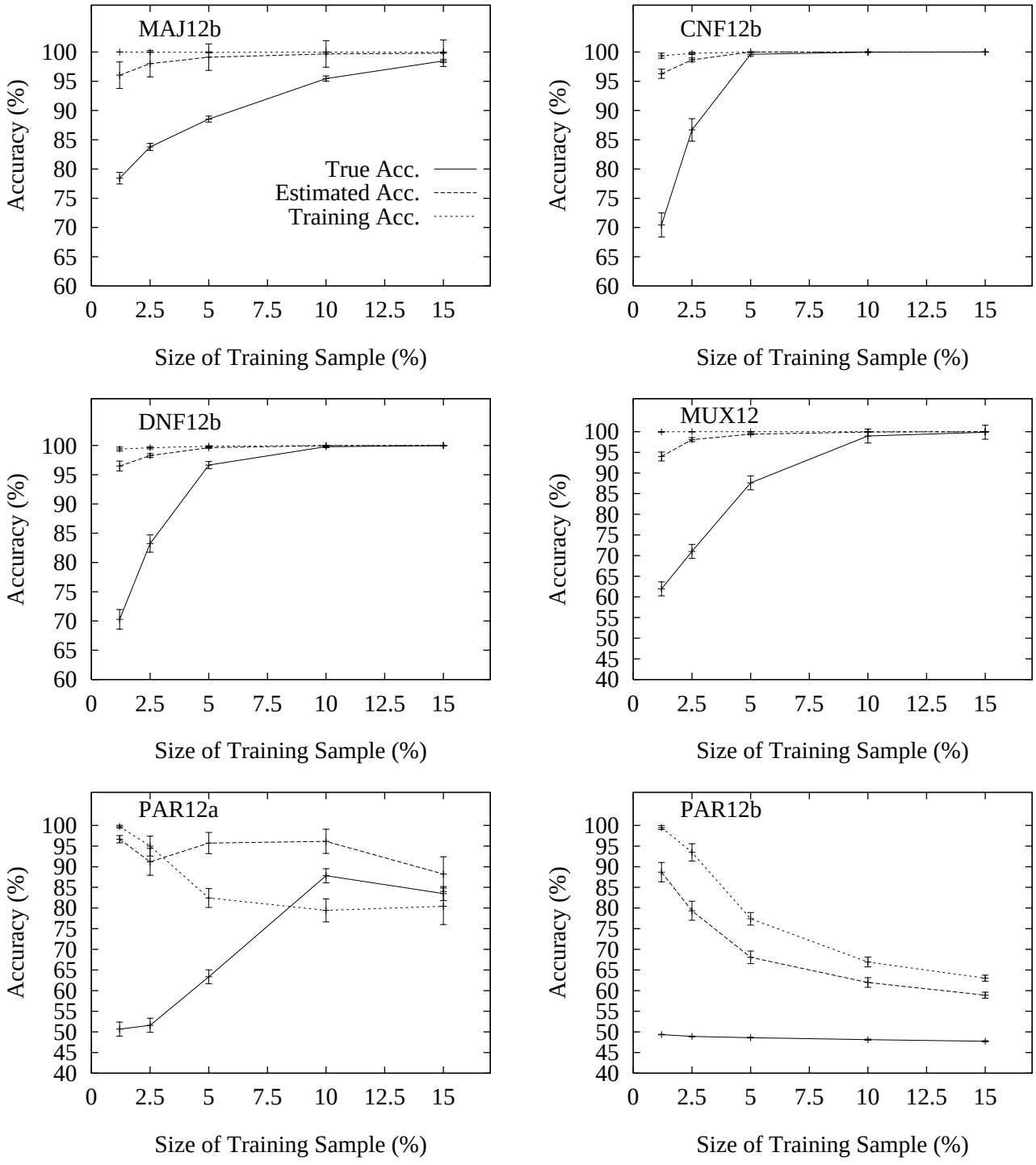


Figure 8.6: Training-set size vs. true accuracy, estimated accuracy, and training accuracy in *HCL*.

hypothesis with the right amount of complexity that has not yet been adjusted to all peculiarities in the training data (i.e., that retains accurate predictions over unseen examples).

Figures 8.4, 8.5, and 8.6 plot these accuracies for different training-set sizes over all artificial concepts. The experiments employ the same 18 artificial concepts used in the previous section. For low variation[1] ($\nabla < 20$) the difference between the estimated and the true accuracy is of approximately 5% in 9-feature functions, and between 10% and 25% in 12-feature functions[2]. Under high variation ($\nabla > 20$) the difference can be higher than 40% (e.g., parity). These differences decrease as the training set is made larger. The point at which the number of examples affects the estimation corresponds to approximately $100 - 200$ examples (except for MAJ9b, PAR9b, PAR12a, and PAR12b). It is interesting to observe the behavior of *HCL* for function PAR12a: the accuracy over the training set goes below the true and estimated accuracy; *HCL* continues building more hierarchical levels even at the expense of reducing accuracy over the training examples. After approximately 10% training-set size, the training accuracy goes below the true accuracy. Figure 9.3 shows how the result is a steep loss in performance.

In general, it is observed that the difference between the estimated accuracy and the true accuracy is more evident as variation increases, and it is not until perfect accuracy (100%) is achieved that all accuracies converge. Thus, under small training-set sizes, it can be concluded that the cross-validation component is highly biased, overestimating the true accuracy significantly. With few examples, the cross-validation component cannot find enough statistical support to validate the estimation. When the number of training examples is less than $100 - 200$ examples, the cross-validation component is expected to be a poor predictor of the true accuracy.

Table 8.1: Assessing *HCL* with only one hierarchical level. Numbers enclosed in parentheses rerpesent standard deviations.

Concept	∇ (%)	*HCL*-1-level 1 hierarchical level	*HCL*-standard multiple hierachical levels
DNF9*a*	17	99.40 (1.94)	99.43* (2.12)
CNF9*a*	17	99.37* (2.17)	99.30 (2.51)
MAJ9*a*	17	97.31 (3.84)	98.43* (3.96)
MAJ9*b*	21	86.27* (3.80)	82.30 (3.59)
CNF9*b*	22	90.93 (6.01)	94.74* (6.49)
MUX9	22	84.40 (4.59)	88.95* (7.60)
DNF9*b*	24	83.80 (5.92)	91.80* (7.64)
PAR9*a*	33	76.04 (9.84)	90.03* (8.59)
PAR9*b*	67	45.76 (0.99)	45.80* (1.32)
MAJ12*a*	13	99.68 (0.99)	99.89* (0.56)
DNF12*a*	15	95.21 (3.41)	99.12* (1.86)
CNF12*a*	15	99.64 (0.86)	99.83* (0.59)
MAJ12*b*	18	87.86 (1.98)	88.53* (1.79)
CNF12*b*	19	86.95 (5.10)	99.61* (1.04)
DNF12*b*	20	87.90 (5.07)	96.64* (2.19)
MUX12	21	71.50 (3.54)	87.63* (5.70)
PAR12*a*	33	52.20 (3.40)	63.35* (8.42)
PAR12*b*	67	49.11* (0.20)	48.61 (0.30)
ABS AVRG		82.96	87.44*

One vs Multiple Levels in *HCL*

The second set of experiments compare *HCL* with a modified version of the algorithm that stops the learning process after only one hierarchical level has been constructed (*HCL*-1-level). The purpose of this test is to measure the contribution made by a mechanism that explores more than one level of complexity dynamically.

Table 8.1 shows predicted accuracy for both systems using 10% and 5% training-set sizes for 9- and 12-feature functions respectively. The advantage of *HCL* over *HCL*-1-level is not evident in simple domains (i.e., domains with low variation, $\nabla < 20$). For high variation ($\nabla > 20$) the difference varies between 4% to 16% points accuracy. *HCL*-1-level has a significant advantage on MAJ9b only (approx.

[1] As a reminder, ∇ is normalized, so that a comparison can be made between training sets having different number of original features.

[2] Examples are distributed over a larger instance space in 12-feature functions than in 9-feature functions. Since more models can fit the data in the former group (under the same number of examples), effects are in general more evident in 12-feature functions than in 9-feature functions.

4% points), which shows *HCL* is not uniformly superior to *HCL*-1-level. The advantage of *HCL* over *HCL*-1-level is clear on the parity domain (PAR9a, PAR12a). An explanation may be that *HCL*-1-level is unable to discover all disjunctive terms with only one layer of abstraction. On average, *HCL* is around 7% points significantly above *HCL*-1-level.

Similar performance between *HCL* and *HCL*-1-level in simple domains ($\nabla < 20$) occurs because in those domains *HCL* builds only few levels of abstraction. Domains with high variation, in contrast, highlight the advantage attached to a mechanism that can vary the degree of representational complexity. Here *HCL* builds enough hierarchical levels to make the difference between both systems significant.

8.4 Feature Construction Component

Each hierarchical level in *HCL* is composed of a linear combination of logical expressions. Each expression is obtained from a beam-search over the space of literal combinations (setting the depth and width of the search space dynamically, Section 7.1). To measure the value of the feature-construction component, Table 8.2 shows statistics gathered by fixing the size of the training set at 10% and 5% for 9- and 12-feature functions respectively.

The first columns (2 and 3) in Table 8.2 show the absolute number of evaluated expressions and the number of eliminated expressions resulting —in average— from each call to the New_Expression module (Figure 7.4). The average is taken from the set of expressions at the hierarchical level where the final hypothesis was constructed. The number of evaluated and eliminated expressions varies between 12,000 for the simplest concept to 391,000 for the hardest concept. Thus, results show how simple concepts explore a smaller space than more difficult ones. The next column shows the ratio between the number of evaluated expressions and the number of eliminated expressions for each concept. I take this ratio as a measure of the effectiveness of the pruning process; it is an example of a local metric testing the effectiveness of a single component, as opposed to global criteria measuring performance over the entire algorithm (Section 3.4). The average ratio over all concepts is close to 1 (actual value is 1.21) which implies that for every evaluated expression, almost always another expression is discarded. The fraction of the search space explored is in fact reduced by a factor much smaller than half, because an expression that

Table 8.2: Results on feature construction and on the size of the search space. Numbers enclosed in parentheses represent standard deviations.

Concept	∇	No. of Expressions (potential features)			Search Space		
	(%)	Evaluated	Eliminated	ratio	Depth	Max. Depth	Max. Width
DNF9a	17	12873	12016	1.07	2.15	3.27	8.94
		(1578)	(1586)		(0.20)	(0.82)	(0.87)
CNF9a	17	12428	11763	1.06	2.10	3.21	8.63
		(1271)	(1294)		(0.25)	(0.58)	(1.14)
MAJ9a	17	16938	19424	0.87	2.51	3.78	9.22
		(2369)	(4013)		(0.68)	(0.62)	(1.23)
MAJ9b	21	65959	91627	0.72	2.34	3.51	8.79
		(30514)	(63623)		(0.67)	(0.86)	(1.06)
CNF9b	22	40128	36359	1.10	2.31	4.31	9.69
		(10671.4)	(16408)		(0.90)	(0.81)	(0.35)
MUX9	22	75939	90691	0.84	1.98	3.96	9.50
		(27944)	(62588)		(0.95)	(1.20)	(0.97)
DNF9b	24	62678	59771	1.05	2.71	4.31	9.42
		(22811.5)	(32763)		(1.07)	(1.05)	(1.11)
PAR9a	33	104568	85438	1.22	3.18	4.62	9.85
		(57970)	(62075)		(1.10)	(1.10)	(1.06)
PAR9b	67	146359	100684	1.45	2.82	4.88	10.31
		(72658)	(50051)		(0.44)	(1.11)	(3.35)
MAJ12a	13	34046	36675	0.92	2.30	3.66	8.24
		(6701)	(6623)		(0.64)	(0.46)	(0.66)
DNF12a	15	89663	67399	1.33	2.53	5.04	9.15
		(25121)	(60687)		(0.64)	(0.98)	(1.12)
CNF12a	15	32707	28500	1.14	2.97	3.63	9.42
		(3206)	(2130)		(0.24)	(0.75)	(0.71)
MAJ12b	18	115384	224512	0.51	2.68	4.38	8.20
		(36656)	(51206)		(0.56)	(1.13)	(1.10)
CNF12b	19	152288	105763	1.43	2.67	5.40	9.28
		(41778)	(27089)		(1.17)	(1.46)	(1.38)
DNF12b	20	178116	122324	1.46	2.64	5.36	8.81
		(63825)	(36335)		(0.89)	(1.52)	(1.68)
MUX12	21	334750	286748	1.17	2.99	5.18	9.22
		(101182)	(87148)		(0.72)	(1.37)	(1.49)
PAR12a	33	341032	166820	2.04	4.15	9.36	9.83
		(131141)	(72414)		(0.49)	(1.10)	(1.27)
PAR12b	67	391373	162257	2.41	4.37	9.48	9.73
		(143101)	(76339)		(0.53)	(1.29)	(1.01)
ABS AVRG		122620	94932	1.21	2.74	4.85	9.24

is eliminated automatically discards all other expressions that could potentially be derived from it (i.e., that can be obtained by conjoining more feature-values). Thus, the feature construction component is not only efficient in exploring the space of logical expressions, but capable of identifying the right expressions, as the advantage in accuracy gained when multiple levels are constructed is more evident on large space spaces (Table 8.1).

Columns 6, 7, and 8 in Table 8.2 report on the size of the search-space bounds (columns 7 and 8 show limits in depth and width respectively, whereas column 6 is the depth at which the best expression was found). Averaging over all concepts, the maximum depth explored is of 4.85, whereas the depth at which the best expression was found is of 2.74. This leaves only 44% of the maximum depth as unproductive search. The maximum width is of 9.24 in average, which compared to the total number of possible combinations, implies a drastic reduction in the number of candidate expressions contained within the beam. Both bounds (depth and width) take on small values over all functions, which means the size of the space explored is highly constrained.

In summary, the feature construction component is able to prune major portions of the space of logical expressions. The space is reduced by constraining the space bounds drastically, and by eliminating all those expressions that have no opportunity to improve over the best current expression. By limiting the size of the space explored, the mechanism becomes in position to identify the right expressions. This is true because the algorithm improves significantly in predictive accuracy (compared to the hypothesis obtained when only one hierarchical level is constructed, Table 8.1) as variation increases, which corresponds to increasingly larger sizes of the search space.

8.5 Conciseness

The last set of experiments in this chapter measure the effects of incorporating a pruning component in *HCL*. The purpose of these experiments is to determine if this component can retain the quality of the final hypothesis while improving conciseness. The pruning component is depicted in Figure 8.7; pruning is done over the final hypothesis exclusively. The component is divided in two parts: first, eliminate expressions in the final hypothesis while the accuracy over the training set is improved (or left the same); and second, eliminate literals on every expression left,

Table 8.3: A comparison of *HCL* with and without pruning. Numbers enclosed in parentheses represent standard deviations. A difference significant at the $p = 0.005$ level is marked with two asterisks.

Concept	∇	*HCL*			*HCL* with pruning		
	(%)	Accuracy	Expressions	Literals	Accuracy	Expressions	Literals
DNF9*a*	17	99.43	8.68	19.56	99.57*	2.28	4.91
		(2.12)	(4.43)	(10.31)	(1.80)	(1.06)	(1.77)
CNF9*a*	17	99.30*	7.30	16.34	99.27	2.14	4.54
		(2.51)	(4.93)	(11.37)	(2.55)	(1.22)	(1.97)
MAJ9*a*	17	98.43*	4.46	9.68	97.91	3.20	5.11
		(3.96)	(6.99)	(14.18)	(5.01)	(4.18)	(3.94)
MAJ9*b*	21	82.30**	16.00	38.38	80.09	10.11	14.20
		(3.59)	(12.96)	(34.28)	(3.71)	(6.68)	(9.14)
CNF9*b*	22	94.74*	3.64	9.32	93.38	3.19	6.16
		(6.49)	(6.67)	(17.74)	(7.51)	(4.71)	(6.73)
MUX9	22	88.95**	7.94	19.48	84.59	6.70	11.06
		(7.60)	(11.18)	(30.92)	(7.41)	(6.29)	(10.77)
DNF9*b*	24	91.80*	6.56	19.60	89.23	4.50	8.80
		(7.64)	(10.51)	(32.47)	(9.74)	(5.85)	(9.34)
PAR9*a*	33	90.03*	5.56	16.66	89.23	3.51	7.17
		(8.59)	(9.75)	(30.59)	(9.74)	(5.11)	(8.28)
PAR9*b*	67	45.80*	9.50	29.50	45.60	6.94	12.84
		(1.32)	(1.78)	(7.32)	(1.13)	(1.65)	(4.14)
MAJ12*a*	13	99.89	10.04	22.88	99.96*	6.36	9.94
		(0.56)	(3.63)	(7.89)	(0.23)	(2.40)	(3.44)
DNF12*a*	15	99.12*	6.86	24.62	99.08	4.20	8.94
		(1.86)	(5.26)	(19.38)	(1.91)	(3.06)	(5.59)
CNF12*a*	15	99.83	4.18	11.71	100.00**	2.12	5.44
		(0.59)	(3.76)	(10.24)	(0.00)	(1.55)	(3.04)
MAJ12*b*	18	88.53*	13.04	38.55	88.35	8.69	13.42
		(1.79)	(4.52)	(16.10)	(2.15)	(2.90)	(5.73)
CNF12*b*	19	99.61*	3.58	15.14	99.53	2.28	6.00
		(1.04)	(4.66)	(20.18)	(1.17)	(2.49)	(4.10)
DNF12*b*	20	96.64**	9.00	32.28	95.79	6.32	11.30
		(2.19)	(6.12)	(23.68)	(2.60)	(3.75)	(7.39)
MUX12	21	87.63*	11.61	39.46	87.24	7.78	12.25
		(5.70)	(5.73)	(22.06)	(5.88)	(3.44)	(6.97)
PAR12*a*	33	63.35**	14.62	45.12	57.43	7.90	21.62
		(8.42)	(0.70)	(9.18)	(8.44)	(1.41)	(6.02)
PAR12*b*	67	48.61*	10.00	48.04	48.52	7.56	20.68
		(0.30)	(0.00)	(5.87)	(0.26)	(1.38)	(65.65)
ABS AVRG		87.44*	8.47	25.35	86.37	5.32	10.24

while retaining or improving training-set accuracy.

Table 8.3 compares *HCL* with and without pruning when the size of the training set is 10% and 5% for 9- and 12-feature functions respectively. Each version of the algorithm (Standard vs Pruning) is characterized by three column-values: accuracy of the final hypothesis, number of expressions in the final hypothesis, and number of literals in the final hypothesis.

Regarding accuracy, the difference between *HCL* standard and *HCL* with pruning is insignificant on simple domains, but increases on difficult domains. For example, the difference in accuracy on concept MUX9 is about 4% (significant); the difference in accuracy on concept PAR12a is about 6% (significant). On average, *HCL* with pruning is only approximately 1% points below *HCL*. *HCL* standard tends to be more accurate, partly because the set of artificial domains are free of noise. Thus, *HCL* with pruning can gain in accuracy but at the expense of (in some cases) a significant loss of accuracy.

Regarding conciseness, *HCL* with pruning reduces the number of expressions of *HCL*. The reduction in average is about 35%. The reduction in the total number of literals (operand on each current expression or its negation; an operand can be an original feature or a previously defined expression) is approximately 60% on average. A preliminary analysis of both systems showed that *HCL* with pruning improves interpretability of the final hypothesis when the domain is not too complex.

8.6 Discussion

Results obtained from the experiments in this chapter give answer to the questions posed at the beginning section (Section 8.1):

- Considering the general strategy and design, what is the expected performance of the algorithm?
  HCL is expected to adapt to the right degree of representational complexity. The experiments in this analysis show how the algorithm increases the complexity of the final hypothesis by constructing more hierarchical levels. The need for more levels is triggered when concept variation is high, and when the size of the training set is small. These situations allow many models to fit the data, thus making the domain difficult to learn.

Algorithm 9: Prune Hypothesis
Input: Training Set T, Hypothesis H
Output: Pruned Hypothesis
PRUNE_HYPOTHESIS(T, H)
(1) **while** No of Expressions > 1
(2) **foreach** $F_i \in H$
(3) Let Acc_i be the accuracy in T when removing F_i from H
(4) **if** $\text{Acc}_i >$ Current-Best-Accuracy
(5) Current-Best-accuracy $= Acc_i$
(6) $\text{Worse}_F = F_i$
(7) **end for**
(8) Eliminate Worse_F from H
(9) **end while**
(10) **foreach** $F_i \in H$
(11) Eliminate literals in F_i while accuracy improves
(12) **end for**
(13) **return** H

Figure 8.7: Pruning the final hypothesis in *HCL*.

A comparison of *HCL* with a version of the algorithm where only one hierarchical level is constructed (Table 8.1) shows a significant improvement in accuracy under high variation (of up 16% points), and very similar behavior under small variation. Thus, *HCL* is able to adapt the representation as required by the domain under analysis.

Also, although concept variation cannot be computed on real-world domains where full class-membership information is unavailable, analyzing the number of levels constructed by *HCL* can throw insight into the degree of complexity displayed by the domain under study. *HCL* can then provide an objective measure to classify the difficulty of a domain.

- What is the contribution of each component in *HCL*?
  A functional decomposition analysis proved here instrumental to assess the contribution of the different components in *HCL*. For example, a problem detected during the construction of hierarchical levels in *HCL* is the poor estimation of the true accuracy output by the cross-validation component (Figures 8.4, 8.5, and 8.6). Better techniques to estimate true accuracy are needed, particularly when the training-set size is small (i.e., when the training set is approx. less than $100 - 200$ examples). On the other hand, adding new features into the

space of original and previous features (Augment-Dimensionality component) demonstrated to be efficient in changing the complexity of the hypothesis at each new level. Results in Figures 8.1, 8.2, and 8.3 show how *HCL* explores different levels of representation according to the characteristics of the domain. In general, more hierarchical levels are needed when the training-set size is small and when concept variation is high. The component in charge of searching in the space of logical expressions (Search-Expressions component) proved able to constrain the size of the search space while separating out the right expressions (Table 8.2). Also, the pruning component appeared, in general, capable of reducing the size of the final hypothesis with a small loss in predictive accuracy (Table 8.3). The next chapter describes further investigation of *HCL*'s performance, by comparing the algorithm with standard models on both artifical and real-world domains.

Chapter 9

An Experimental Comparison with Standard Models

In this chapter I show experiments with *HCL* to assess the efficiency of the entire algorithm, i.e., to assess the efficiency of the combination of all existing components. The goal is to determine how well *HCL* scales when compared to several standard learning models. The two main performance criteria are predictive accuracy and CPU time. This chapter is divided in two sections. The first section tests *HCL* over artificial domains (Appendix A) where experimental conditions are controlled, and no sources of difficult exist except high variation. *HCL* is compared to standard learning models to further analyze the efficiency of a flexible and adaptable mechanism (Section 6.2). The second part tests *HCL* over a set of real-world domains. Evaluating the algorithm here must take into account that each domain may exhibit characteristics lying outside the scope of the algorithm design (Section 6.1). For example, many domains are known to have noise in class and feature values. To correctly interpret the results, Section 9.2 categorizes the set of real-world domains by separating out possible sources of difficulty other than feature interaction.

9.1 Results on Artificial Domains

Using artificial concepts, the flexible and adaptable mechanism of *HCL* is expected to attack the feature interaction problem (only source of difficulty), by constructing the partial subconcepts lying between the primitive features and the final hypothesis. But, what should we expect of *HCL*'s performance when compared to standard models? Here, two scenarios may be found. First, if the concept is simple, such that the gap between the primitive features and the final hypothesis is small, then performance should be comparable to standard models. In this case *HCL* outputs a linear combination of expressions, each expression a logical function of primitive features (or simple expressions). On the other hand, if the representational gap is large, such

that many subconcepts must be discovered before the concept is well approximated, *HCL* is expected to improve performance. The goal of this section is to assess how close this expectation matches the empirical results.

The Algorithms Used For Comparison

Several learning algorithms are used as baselines for comparison: *C4.5*-trees (Quinlan, 1994), representing a standard decision-tree learner (both pruned and unpruned trees); *C4.5*-rules (Quinlan, 1994), representing a rule-based system; *DALI* (Section 5.2 (Vilalta & Rendell, 1997)) as a feature-construction decision-tree learner; and a k-nearest neighbor ($k = 10$).

The Experimental Methodology

Every reported result is the average over 50 runs. In every table, numbers enclosed in parentheses represent standard deviations. Experiments employ artificial domains displaying various degrees of concept variation ∇ (explained with detail in Section 2.5). Artifical concepts are divided in 9-feature concepts and 12-feature concepts, each group covering a wide range of different values for ∇ (similar to Chapter 8). CPU time is measured in seconds on a Sparc 10.

Predictive Accuracy

Figures 9.1, 9.2, and 9.3 show learning curves comparing *HCL* with the algorithms listed above. 95% confidence intervals using a two-sided t-student distribution encapsulate each curve-point, to measure the variability in the results. The following statements can be drawn from the information given by the learning curves:

Concepts With Low Feature Interaction. When concepts display low feature interaction, such that the hypothesis approximating the target concept is simple, all algorithms tend to do well, For example, in both MAJ9a ($\nabla = 17$) and MAJ12a ($\nabla = 13$), the difference in accuracy is minimal (except for k-nearest neighbor). In this group of concepts ($\nabla < 20$), high accuracy is achieved with few examples (10% and 5% training-set size on 9- and 12-feature functions respectively). In these cases, functionalities such as flexibility and adaptability in the concept-language representation produce no significant improvements. Better performance, however, is observed when the degree of feature interaction increases.

HCL **and** *DALI*. The advantage of *HCL* is less clear when the algorithm is compared to *DALI*. The small difference between these two algorithms can be explained

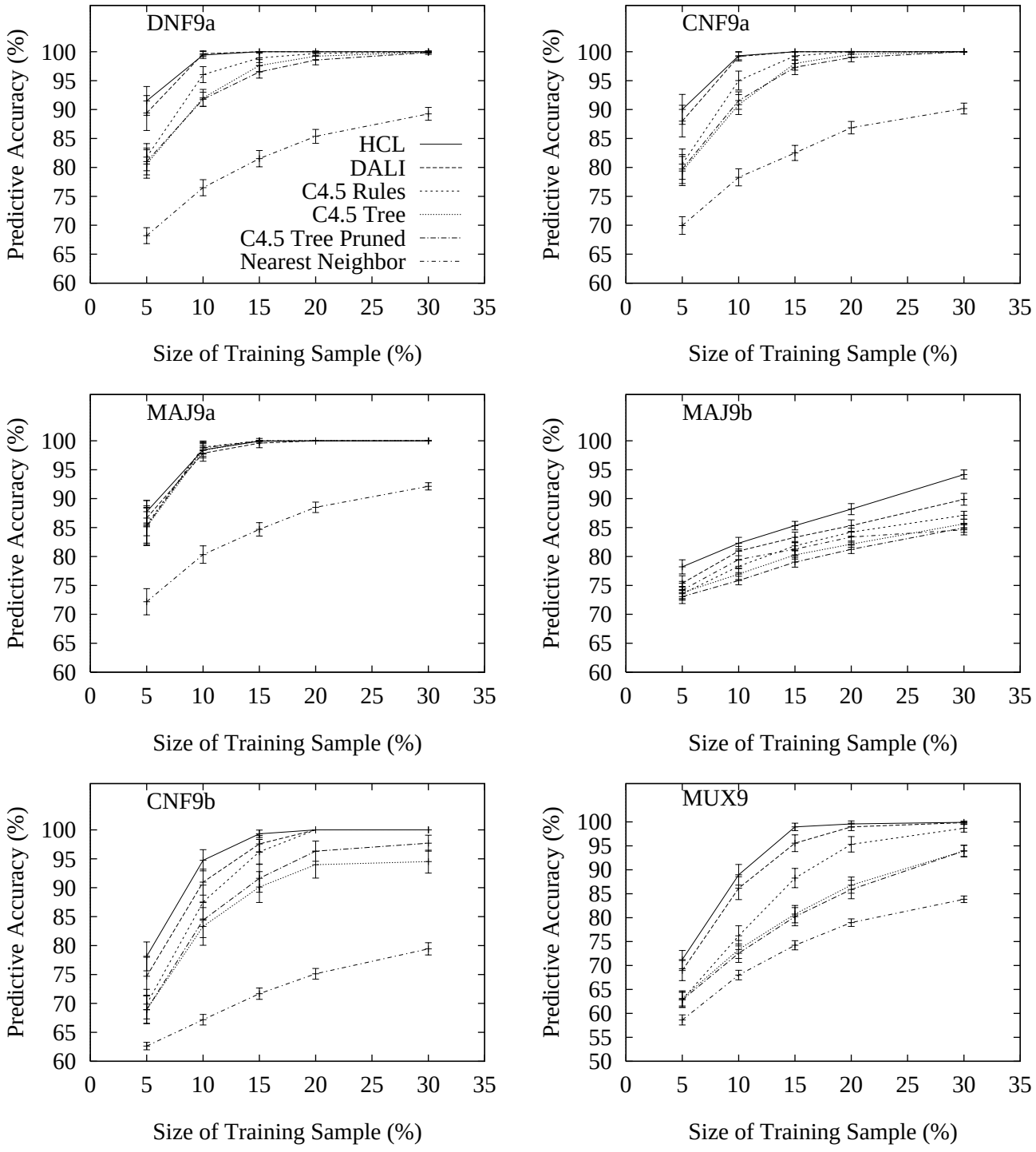


Figure 9.1: Training-set size vs. predictive accuracy comparing *HCL* with *DALI*, *C4.5*-rules, *C4.5*-trees (pruned and unpruned), and k-nearest-neighbor ($k = 10$).

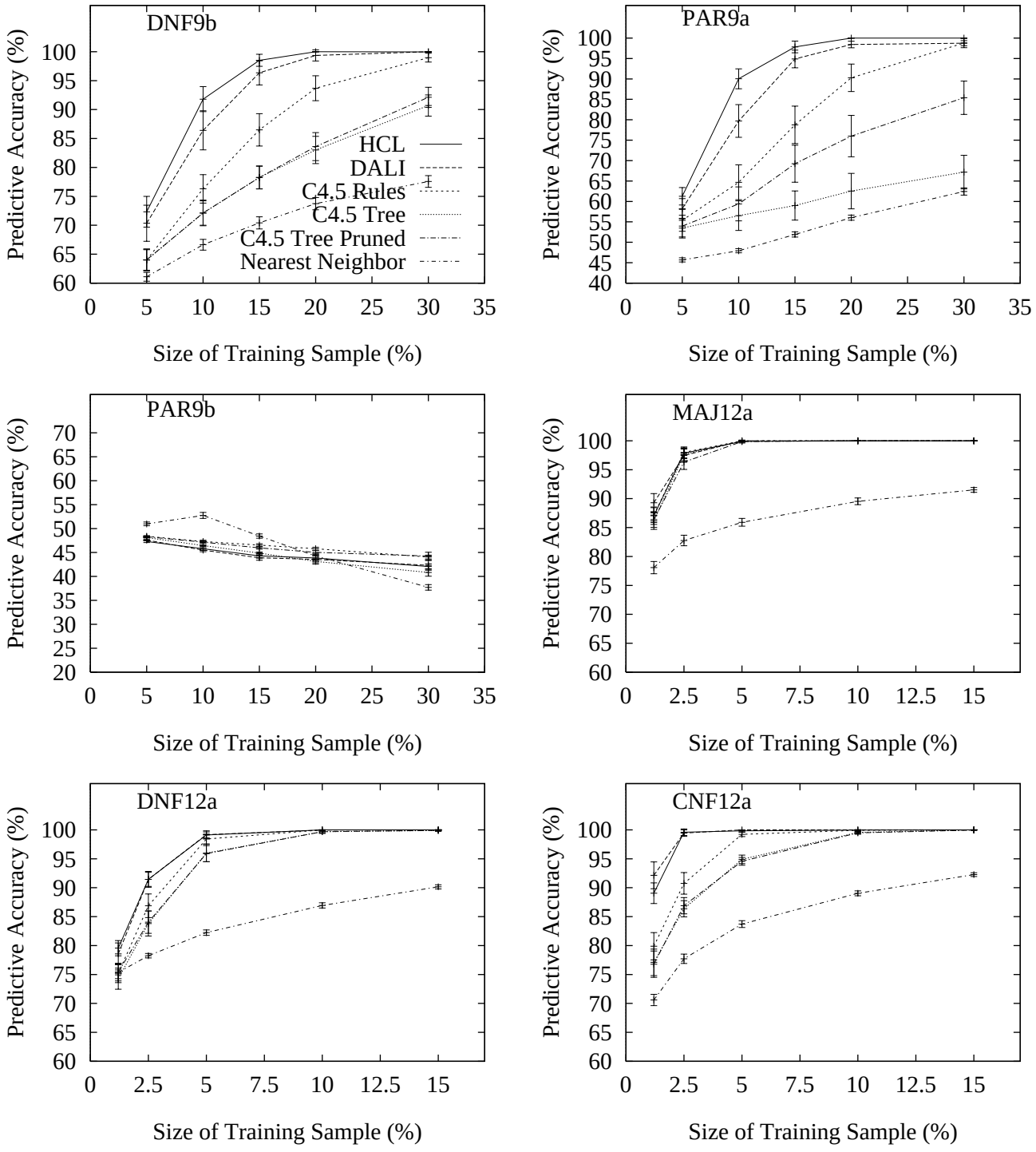


Figure 9.2: Training-set size vs. predictive accuracy comparing *HCL* with *DALI*, *C4.5*-rules, *C4.5*-trees (pruned and unpruned), and k-nearest-neighbor ($k = 10$).

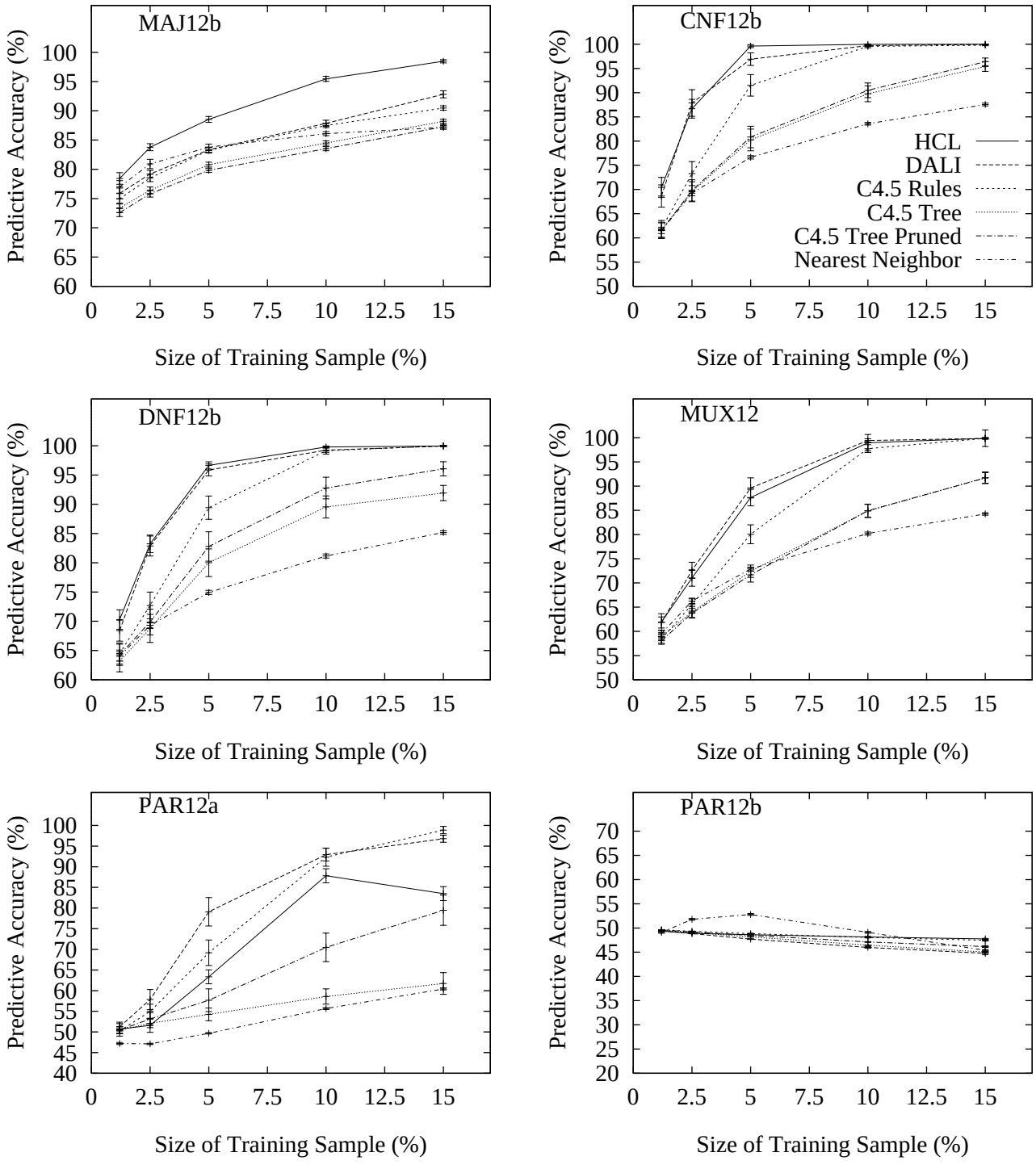


Figure 9.3: Training-set size vs. predictive accuracy comparing *HCL* with *DALI*, *C4.5*-rules, *C4.5*-trees (pruned and unpruned), and k-nearest-neighbor ($k = 10$).

as follows. Both *HCL* and *DALI* use the same feature-construction component to build new logical expressions. But whereas in *DALI* this component serves to produce a new splitting function at each node of a decision tree, *HCL* employs this component to yield a linear combination of new expressions. *DALI*, as a decision-tree learner, remains subject to the fragmentation problem (Section 4.2), which becomes disadvantageous as variation grows. The results show in general a slight advantage of *HCL* over *DALI*, precisely over those domains where ∇ is high (e.g., except for concepts PAR12a and MUX12, where *DALI* outperforms *HCL* significantly).

Estimating True Accuracy. For some concepts, we can observe (Figures 9.1, 9.2, and 9.3) how the advantage of *HCL* over other algorithms in terms of predictive accuracy tends to disappear under small training sets. For example, in concepts PAR9a, DNF9b, DNF12b, CNF12b, and PAR12a, the difference in accuracy for low training-set sizes varies significantly as more examples are used for training; in PAR9a, the difference between *HCL* is around 6% at 5% training-set size, and goes up to around 30% at 15% training-set size. The results obtained in Section 8.3 partly explain such results: By overestimating the true accuracy, *HCL* stops adding more levels on top of the hierarchical structure before the right degree of complexity is attained. Better results could potentially be achieved if a more accurate estimator were employed when few examples are available (e.g., bootstrap methods (Weiss & Kulikowski, 1990)).

Advantage of *HCL*. *HCL* shows a general advantage over all other algorithms (and across different training-set sizes), particularly when the concept exhibits high degree of variation, e.g., functions DNF{9,12}b, CNF{9,12}b, multiplexor MUX, and parity PAR (except for PAR12a). In general, the difference goes up to approximately 30% (CNF9b) and 40% (PAR9a) when compared to the k-nearest neighbor, and up to 10% (PAR9a) when compared to *DALI* (closest competitor). The advantage of *HCL* improving with increased variation is clear between MAJ9a and MAJ9b, and between MAJ12a and MAJ12b. Except for *DALI*, differences are in most cases significant.

Table 9.1 provides exact values for predictive accuracy when the training-set sizes are 10% for 9-feature functions and 5% for 12-feature functions. The sizes correspond to the point where the maximum difference among systems along the learning curves is generally observed. *HCL* performs significantly better than *C4.5*-trees (both pruned and unpruned) and k-nearest neighbor over most domains; the difference goes up to approximately 30% and 40% points respectively (PAR9a). The advantage of *HCL*

Table 9.1: Tests on predictive accuracy for artificial domains. Numbers enclosed in parentheses rerpresent standard deviations. A difference significant at the $p = 0.005$ level is marked with two asterisks.

Concept	∇ (%)	*C4.5*trees- unpruned	*C4.5*trees- pruned	*C4.5*rules	Nearest Neighbor	*DALI*	*HCL*
DNF9*a*	17	92.01 (5.19)	91.78 (4.34)	96.04 (4.90)	76.50 (4.90)	99.65* (1.73)	99.43 (2.12)
CNF9*a*	17	90.86 (6.08)	91.56 (5.33)	95.01 (5.78)	78.28 (5.20)	99.16 (2.65)	99.30* (2.51)
MAJ9*a*	17	98.35 (4.68)	98.77 (3.80)	98.90* (3.72)	80.33 (5.35)	97.84 (4.86)	98.43 (3.96)
MAJ9*b*	21	76.96 (3.29)	75.84 (2.58)	78.30 (3.84)	79.46 (5.25)	80.94 (2.84)	82.30** (3.59)
CNF9*b*	22	83.31 (11.52)	84.40 (10.70)	87.53 (10.32)	67.18 (3.23)	90.96 (7.95)	94.74** (6.49)
MUX9	22	73.35 (6.61)	72.57 (6.81)	76.20 (7.45)	68.01 (3.52)	86.16 (8.44)	88.95** (7.60)
DNF9*b*	24	72.11 (7.47)	72.13 (7.83)	76.29 (8.65)	66.64 (3.30)	86.41 (11.96)	91.80** (7.64)
PAR9*a*	33	56.52 (12.79)	59.40 (14.70)	64.69 (15.04)	47.94 (1.80)	79.70 (14.15)	90.03** (8.59)
PAR9*b*	67	46.46 (0.55)	47.17 (0.76)	47.30 (0.94)	52.75** (2.20)	45.50 (1.19)	45.80 (1.32)
MAJ12*a*	13	99.93 (0.46)	99.87 (0.91)	100.00* (0.00)	85.91 (2.37)	100.00* (0.00)	99.89 (0.56)
DNF12*a*	15	95.99 (5.30)	95.89 (4.92)	98.41 (2.99)	82.24 (1.67)	99.18* (2.30)	99.12 (1.86)
CNF12*a*	15	94.90 (2.58)	94.57 (2.37)	99.22 (1.31)	83.72 (2.08)	100.00** (0.00)	99.83 (0.59)
MAJ12*b*	18	80.79 (1.46)	79.82 (1.27)	83.23 (1.63)	83.79 (1.79)	83.33 (1.52)	88.53** (1.79)
CNF12*b*	19	80.24 (7.94)	80.83 (7.83)	91.49 (7.85)	76.65 (1.21)	96.92 (4.60)	99.61** (1.04)
DNF12*b*	20	80.01 (8.36)	82.79 (8.86)	89.41 (7.03)	74.92 (1.30)	95.87 (3.60)	96.64* (2.19)
MUX12	21	72.42 (4.49)	71.67 (5.25)	80.05 (6.93)	72.88 (1.38)	89.63* (7.38)	87.63 (5.70)
PAR12*a*	33	54.26 (5.43)	57.68 (9.70)	69.18 (10.89)	49.68 (0.58)	79.09** (12.19)	63.35 (8.42)
PAR12*b*	67	48.16 (0.10)	48.47 (0.19)	48.84 (0.26)	52.79** (0.66)	47.70 (0.25)	48.61 (0.30)
ABS AVRG		77.59	78.07	82.23	71.09	86.56	87.44*

over *DALI* and *C4.5*-rules is significant on domains with high variation only (more evident in *C4.5*-rules). The difference goes up to 10% when compared to *DALI*, and up to 25% when compared to *C4.5*-rules.

Functions PAR9b and PAR12b may at first glance seem an exception to the analysis above: all algorithms reflect a strong performance degradation (at the random-guessing level) over these functions. Nevertheless, such degradation is expected whenever a similarity-based bias is inappropriate (Section 2.6). Specifically, when the number of features in a parity concept exceed half of the total number of features, the probability that two neighbor examples (at Hamming distance one) are labeled differently surpasses 50%. Since most algorithms label new examples based on the class labels of the closest examples, these functions fall outside the class of domains that can be learned under a similarity-based assumption (Section 2.5).

Averaging over each column in Table 9.1 gives an advantage of less than 1% point to *HCL* (not significant) over its closest competitor (*DALI*), 5% points compared to *C4.5*rules, 9% points compared to *C4.5*-trees, and up to 16% points when compared to its worst competitor (k-nearest neighbor). If a distinction is made between simple and hard concepts, we observe that for simple concepts (i.e., $\nabla < 20$) this advantage tends to disappear. For hard concepts (i.e., $\nabla > 20$), the difference between *HCL* and *DALI* remains in the range of 1%, but increases when compared to the other systems. The difference increases to 7% when compared to *C4.5*rules and to 11% points compared to *C4.5*trees.

Discussion

Empirical results displayed in Figures 9.1, 9.2, 9.3, and Table 9.1 answer the question posed at the beginning of the section: what should we expect of *HCL*'s performance when compared to standard models? Under low variation, a simple representation in the hypothesis can match the concept structure, and the difference between *HCL* and other systems is small (the advantage of *HCL*, however, remains significant in many of these concepts). But when the simplicity assumption is incorrect because of high variation, *HCL* shows a significant advantage in predictive accuracy over standard models. These conclusions agree with the results comparing *HCL* with a modified version of the algorithm where only one hierarchical level is constructed (Section 8.3, Table 8.1). Both systems perform similarly under low variation, but *HCL* shows significant gains in predictive accuracy when the final structure is composed of more

than one hierarchical level.

The results above come from an idealized scenario. An assumption is made that no sources of difficulty other than high feature interaction exist. Section 9.2 analyzes other situations where additional factors may affect accurate predictions.

CPU **Time**

Table 9.1 displays CPU time, measured in seconds, for all learning algorithms. The fastest algorithm is *C4.5*-trees —on average, a single run takes less than one second. Both *DALI* and *Nearest-Neighbor* never go beyond one order of magnitude above *C4.5*-trees. *DALI* runs in times comparable to standard models despite the costly feature construction component. The mechanism behind *HCL* is extremely costly: the time spent for a single run in the cases shown can go up to two orders of magnitude above *C4.5*-trees.

Table 9.1 also shows how CPU time increases as variation grows (consistently over all algorithms). This is because the higher the degree of dissimilarity of class values among neighboring examples (Section 4.2), the more single-class regions must be delineated throughout the instance space. Thus, variation (∇) is a good estimator of the complexity required by the final hypothesis.

In conclusion, CPU time is high. The current implementation cannot satisfy the demands of most practical applications. Moreover, with the advent of an increase in *data mining* (Weiss & Indurkhya, 1998) software intended to discover relevant patterns buried under millions of records, it becomes even more important to create machine-learning tools exhibiting short response times.

9.2 Results on Real-World Domains

Real-world domains introduce sources of difficulty for which *HCL* is not designed. For example, noise has detrimental effects during learning, particularly when the mechanism augments the dimensionality of the instance space with newly constructed features (Vilalta, 1994). In other situations the set of features may be insufficient to approximate the target concept (i.e., the domain may exhibit representational inadequacy). In this case, the features describing the examples fall short from capturing relevant portions of the target concept. What then can we expect of *HCL*'s performance in real-world domains? The following analysis responds to this question.

Table 9.2: Results on CPU time (seconds) for artificial domains. Numbers enclosed in parentheses rerpresent standard deviations. A difference significant at the $p = 0.005$ level is marked with two asterisks.

Concept	∇ (%)	*C4.5*trees- unpruned	*C4.5*trees- pruned	*C4.5*rules	Nearest Neighbor	*DALI*	*HCL*
DNF9*a*	17	0.15** (0.05)	0.15** (0.05)	0.17 (0.04)	0.60 (0.03)	0.27 (0.05)	5.89 (0.58)
CNF9*a*	17	0.16** (0.05)	0.16** (0.05)	0.16* (0.05)	0.62 (0.04)	0.29 (0.05)	5.73 (0.48)
MAJ9*a*	17	0.15** (0.05)	0.15** (0.05)	0.18 (0.04)	0.64 (0.05)	0.27 (0.04)	6.93 (0.94)
MAJ9*b*	21	0.16** (0.05)	0.16** (0.05)	0.18 (0.04)	0.62 (0.04)	0.31 (0.04)	47.54 (27.00)
CNF9*b*	22	0.15** (0.05)	0.15** (0.05)	0.18 (0.04)	0.62 (0.05)	0.32 (0.05)	15.34 (5.68)
MUX9	22	0.17** (0.05)	0.17** (0.05)	0.19 (0.04)	0.63 (0.05)	0.36 (0.06)	37.45 (21.89)
DNF9*b*	24	0.15** (0.05)	0.15** (0.05)	0.19 (0.03)	0.63 (0.05)	0.33 (0.05)	27.38 (15.28)
PAR9*a*	33	0.16** (0.05)	0.16** (0.05)	0.21 (0.04)	0.62 (0.04)	0.42 (0.06)	40.33 (28.78)
PAR9*b*	67	0.17** (0.05)	0.17** (0.05)	0.19 (0.05)	0.62 (0.04)	0.66 (0.15)	60.00 (23.00)
MAJ12*a*	13	0.55** (0.05)	0.55** (0.05)	0.70 (0.05)	11.55 (0.08)	2.98 (0.12)	25.91 (4.80)
DNF12*a*	15	0.55** (0.05)	0.55** (0.05)	1.11 (0.33)	11.45 (0.09)	3.34 (0.19)	55.59 (13.89)
CNF12*a*	15	0.56** (0.05)	0.56** (0.05)	1.31 (0.40)	10.26 (3.00)	2.90 (0.13)	26.31 (0.78)
MAJ12*b*	18	0.61** (0.03)	0.61** (0.03)	1.16 (0.27)	11.54 (0.08)	3.90 (0.26)	219.13 (45.73)
CNF12*b*	19	0.58** (0.05)	0.58** (0.05)	1.18 (0.37)	8.60 (4.16)	3.54 (0.37)	76.29 (36.07)
DNF12*b*	20	0.59** (0.04)	0.59** (0.04)	1.16 (0.25)	11.48 (0.11)	3.69 (0.31)	156.32 (46.08)
MUX12	21	0.61** (0.03)	0.61** (0.03)	1.21 (0.09)	11.54 (0.08)	4.54 (0.36)	555.31 (523.00)
PAR12*a*	33	0.64** (0.04)	0.64** (0.04)	1.33 (0.24)	11.55 (0.10)	6.13 (1.06)	165.75 (33.0)
PAR12*b*	67	0.63** (0.05)	0.63** (0.05)	1.24 (0.12)	11.56 (0.12)	8.69 (1.37)	175.00 (58.3)
ABS AVRG		00.37	00.37	00.67	5.84	2.38	95.01

Table 9.3 illustrates the expectations for the performance of *HCL* when applied to different classes of domains. The columns correspond to different sources of difficulty (an entry value of 1 means the source is present, a value of 0 means the source is absent, a value of X means the source can be either present or absent). The first row belongs to a class of domains where no sources of difficulty are present. From the results obtained on artificial domains (Section 9.1), we expect *HCL* to perform similarly to standard models, since not much complexity is required in representing the final hypothesis (i.e., variation is low). The second row assumes the only source of difficulty is a low-level representation of primitive features, such that many intermediate terms must be discovered before a final hypothesis is output (i.e., feature interaction, and thus concept variation, is high). Here *HCL* is expected to outperform standard models by dynamically adapting to the right degree of complexity in the representation. On the other hand, whenever noise or representational inadequacy are present, we expect *HCL* to perform below average, as feature-construction components are particularly sensitive to such sources of difficulty. The goal next is to observe if the above expectations agree with experimental data.

Interpretation of Results

The next sections report on experimental data showing the performance of *HCL* on different types of domains. Results show learning curves for all real-world domains using 25%, 50%, and 75% of the examples available for training and the rest for testing. Each reported value is the average over 50 runs; error bars denote 95% confidence intervals. The same algorithms in Section 9.1 are used here as baselines for comparison. To facilitate interpretation of results, the y-axis on each graph (i.e., predictive accuracy) was scaled to focus on the range of values where all algorithms group together. Thus, different graphs show different value ranges on the y-axis. The nearest-neighbor algorithm was excluded from these experiments because of its low performance.

Real-world domains can be obtained from the UCI repository on machine learning,

Table 9.3: The expectations on *HCL*'s performance for real-world domains.

Feature Interaction	Noise	Representational Inadequacy	Expected Performance
0	0	0	Similar to Std. Models
X	X	1	Below Average
X	1	X	Below Average
1	0	0	Above Average

except for the star-cluster domain, (obtained from (Andrews & Herzberg, 1985)). Because of *HCL*'s expensive mechanism, an upper bound on the size of the training set was set to 1000 examples[1].

Domains Exhibiting Relatively Little Difficulty

The first group of domains[2] in this analysis is presumed to contain no sources of difficulty. The domains are simple and most standard models are expected to perform well over this group (first row, Table 9.1). Datasets include the mushroom domain, where a mushroom is classified as poisonous or edible based on direct characteristics of the plant, such as shape, color, odor, etc. The documentation for this domain includes four simple logical rules covering most of the examples. In the thyroid-gland-data domain the goal is to predict the state of the thyroid gland. The two datasets classify a patient as having hyperthyroidism (or not), or as having hypothyroidism (or not). The examples come from a complete medical diagnosis (all information is relevant). Other examples are the star cluster domain predicting star-cluster membership based on spatial coordinates, relative motion and luminosity (complete and relevant features exist). In the (breast) cancer domain the class label differentiates among patients having benign or malignant cancer. Previous studies show only three parallel hyperplanes are consistent with 67% of the examples[3] (Wolberg & Mangasarian, 1990). In general, the features describing the examples in these domains are highly correlated with the concept being learned; searching for complex interactions results in data overfitting.

Figure 9.4 compares predictive accuracy among the different algorithms over these group of domains. With enough examples, *HCL* performs well on the new-thyroid and zoo domains. In general, all systems perform similarly (except for *C4.5*-trees pruned on the mushroom domain). The range over the y axis (i.e., accuracy) is in all cases above 90%. Table 9.4 reports on exact values (predictive accuracy) when the

[1] If the training set T was larger than this value, a new set was created (for each of the 50 executions) by sampling T with replacement.

[2] Real-world domains are classified here differently than Chapter 5. The classification in this chapter is based on the kind of sources of difficulty present on every domain. The classification in Chapter 5 is, in contrast, based on what domains best match a local or global application of multiple-classifier techniques in decision-tree induction.

[3] The database had less examples by the time these studies were made.

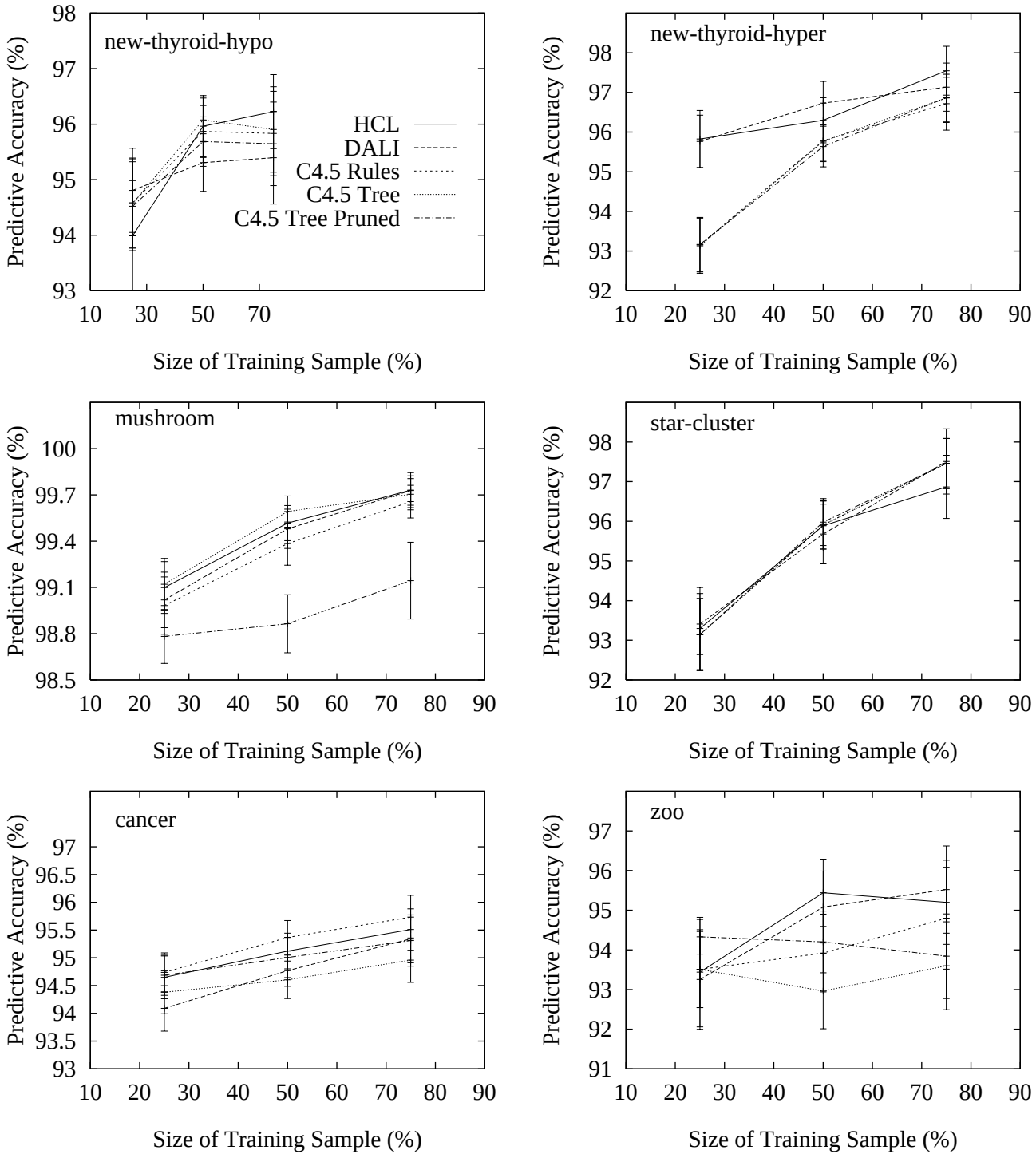


Figure 9.4: Training-set size vs. predictive accuracy comparing *HCL* with *DALI*, *C4.5*-rules, and *C4.5*-trees (pruned and unpruned). Domains are presumed simple, with no sources of difficulty present.

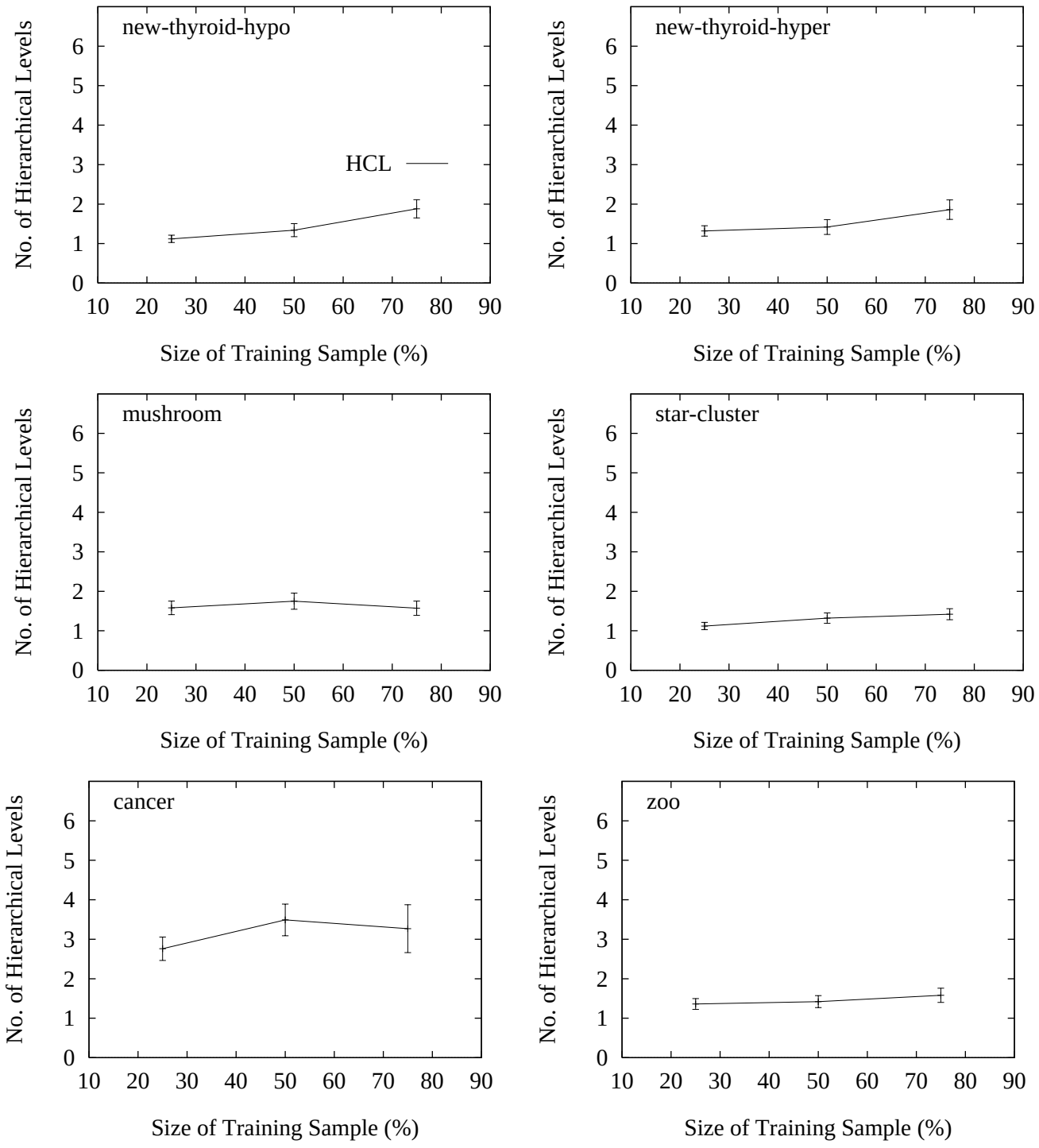


Figure 9.5: Training-set size vs. no of hierarchical levels in *HCL* using simple domains.

Table 9.4: Tests on predictive accuracy for real-world domains. Numbers enclosed in parentheses rerpresent standard deviations.

Concept	*C4.5*trees-		*C4.5*rules	*DALI*	*HCL*	
	unpruned	pruned			pruned	standard
Simple Domains						
cancer	94.60 (1.19)	95.00 (1.27)	95.36* (1.07)	94.76 (0.99)	95.28 (0.99)	95.12 (1.13)
mushroom	99.59* (0.35)	98.86 (0.66)	99.38 (0.49)	99.48 (0.44)	99.54 (0.44)	99.51 (0.40)
new-thyroid-hyper	95.76 (1.81)	95.63 (1.81)	95.78 (1.75)	96.72* (1.93)	96.02 (2.15)	96.29 (2.01)
new-thyroid-hypo	96.07* (1.39)	95.68 (1.57)	95.86 (1.65)	95.30 (1.83)	95.90 (1.96)	95.96 (1.94)
star-cluster	95.90 (2.14)	95.98 (2.09)	95.90 (2.14)	95.67 (2.66)	96.38* (2.29)	95.88 (2.25)
zoo*	92.96 (3.35)	94.2 (2.74)	93.92 (3.46)	95.08 (3.20)	95.67* (2.74)	95.44 (2.99)
AVRG	95.81	95.89	96.03	96.17	96.47*	96.37

training size is fixed at 50%. On average, the accuracy of all systems lies in a range of 1% points, with no significant differences.

Figure 9.5 illustrates the number of hierarchical levels reached by *HCL* for each of these domains. The number of levels never goes above two, and stays close to one most of the time (except for the cancer domain lying between two and three hierarchical levels). This validates the claim that the concepts are simple, with few intermediate concepts lying between the primitive features and the final concept.

Domains Affected by Various Sources of Difficulty

In the second group, each domain exhibits one of several sources of difficulty. Information about the nature of the concept and the methodology employed to collect the examples indicates each domain in this group cannot be considered simple (group one), or having feature interaction as the only source of difficulty (group three).

Domains in this group include medical datasets such as heart disease, where the class label stands for the diagnosis of coronary artery in the patient. The original dataset was collected by characterizing each patient with 76 features, only 14 of which are contained in the dataset: relevant information may be missing. In the liver disorder domain (bupa), the features are mostly blood tests believed to relate to excessive alcohol consumption. Documentation suggests there is uncertainty with regard to the set of features being complete: representational inadequacy may exist. In the lymphography domain, each class label reflects a particular diagnosis. Apparently the

Table 9.5: Tests on predictive accuracy for real-world domains.

Concept	*C4.5*trees-		*C4.5*rules	*DALI*	*HCL*	
	unpruned	pruned			pruned	standard
Various Sources of Difficulty						
heart	73.55 (3.54)	73.97 (3.69)	75.93* (3.45)	73.71 (3.49)	75.26 (3.95)	75.86 (3.63)
credit	70.77 (5.34)	71.29 (4.45)	71.61* (4.90)	68.54 (4.81)	68.72 (4.87)	69.70 (5.05)
bankruptcy	68.83 (5.70)	68.86 (5.59)	69.43* (5.60)	60.96 (5.41)	64.48 (5.12)	64.63 (5.00)
bupa	62.60 (4.45)	63.06* (4.37)	62.92 (4.33)	61.01 (3.07)	61.37 (3.44)	61.76 (3.38)
hepatitis	78.14 (5.09)	79.25 (4.88)	78.75* (4.85)	76.17 (5.38)	78.75* (4.78)	78.60 (5.06)
lymphography2	77.72 (4.43)	78.45 (4.95)	79.59 (4.41)	81.59* (4.69)	80.00 (4.66)	80.56 (4.28)
lymphography3	76.94 (4.32)	77.18 (4.80)	78.86 (4.07)	79.75 (4.98)	80.48 (4.97)	81.45* (4.66)
diabetes	72.79 (2.21)	72.97 (2.19)	73.53* (2.23)	68.73 (2.59)	71.70 (2.28)	72.20 (2.11)
AVRG	72.67	73.13	73.83*	71.31	72.60	73.10

domain contains complex patterns, and the set of features is known to be complete. Not much certainty, however, exists in the quality of the diagnosis: the value for each diagnosis was not verified (testing of physicians was not performed). A specialist estimated internists diagnose correctly only 60% of the times, and specialists 85% of the time (Michalski et al., 1986). This evidence points to noise in the class values. In the credit domain, examples represent instances of credit-card applications. Although features seem complete and relevant, the decision to give credit or not is often subject to criteria not reflected by the features.

Figures 9.6 and 9.7 show learning curves for the third group of domains. *HCL* performs well on the Lymphography domain (outperformed only by *DALI* as the training-set size increases). In other domains, *HCL*'s performance is either around average (e.g., heart, hepatitis), or slightly below average (e.g., bankruptcy, bupa, credit, diabetes). Table 9.5 shows exact values on predictive accuracy for this group (training-set size fixed at 50%). Accuracy lies in the range $70\% - 80\%$ points. The advantage of *HCL* over *C4.5*-trees on Lymphography is of approximately 3%, even when class values are possibly noisy. *HCL* with pruning performs very similar to *HCL* standard (difference within 1% points). Overall, *HCL* performs similar to other algorithms with no significant differences. The results disagree with original expectations predicting *HCL*'s performance below average on this group of domains (Table 9.3).

Even under the influence of various sources of difficulty, *HCL*'s mechanism is able to maintain average performance with respect to other standard learning models.

Figures 9.7 and 9.8 show the number of hierarchical levels reached by *HCL* to output the final hypothesis. The range extends between $2.5 - 4.5$ levels (except for Hepatitis, slightly below that range). Here, the reason for *HCL* exploring more than one hierarchical level may respond to sources of difficulty other than high interaction.

How various sources of difficulty affect *HCL*'s mechanism

HCL is not armed to cope with some of the difficulties mentioned above. The design of the algorithm is guided by functionalities intended to discover the structure of the target concept by finding dependencies among partial subconcepts (Section 6.2). The functionalities assume the partial concepts can be constructed, although simplicity in the representation may be unnecessary. A new arrangement of components (i.e., a new design) is needed to reach additional goals. Next I explain what sources of difficulty are outside the scope of the design of *HCL*, and how the mechanism is affected by those sources.

- **Noise in Feature and Class Values**.
  When class and/or feature values are perturbed by noise, constructing new expressions covering examples of the same class becomes difficult (the perturbed values increase the irregularity in the distribution of class values). As a consequence, the quality of the new expression (as an operand to construct new features) is diminished; the effect gets progressively worse as more expressions are constructed. Moreover, augmenting the dimensionality of the instance space with poor-quality expressions fails to smooth the original distribution of class labels (Section 7.1).

- **Representational Inadequacy**.
  HCL continues building new hierarchical levels as long as the accuracy of the hypothesis at the current level improves over the hypothesis at the level below. An incomplete set of features implies part of the structure of the target concept cannot be discovered, because no information exists to explain the missing terms. In such situations, *HCL* is expected to keep constructing new terms until no gain in predictive accuracy is observed. But since important operands are missing during the construction of logical expressions, high-level

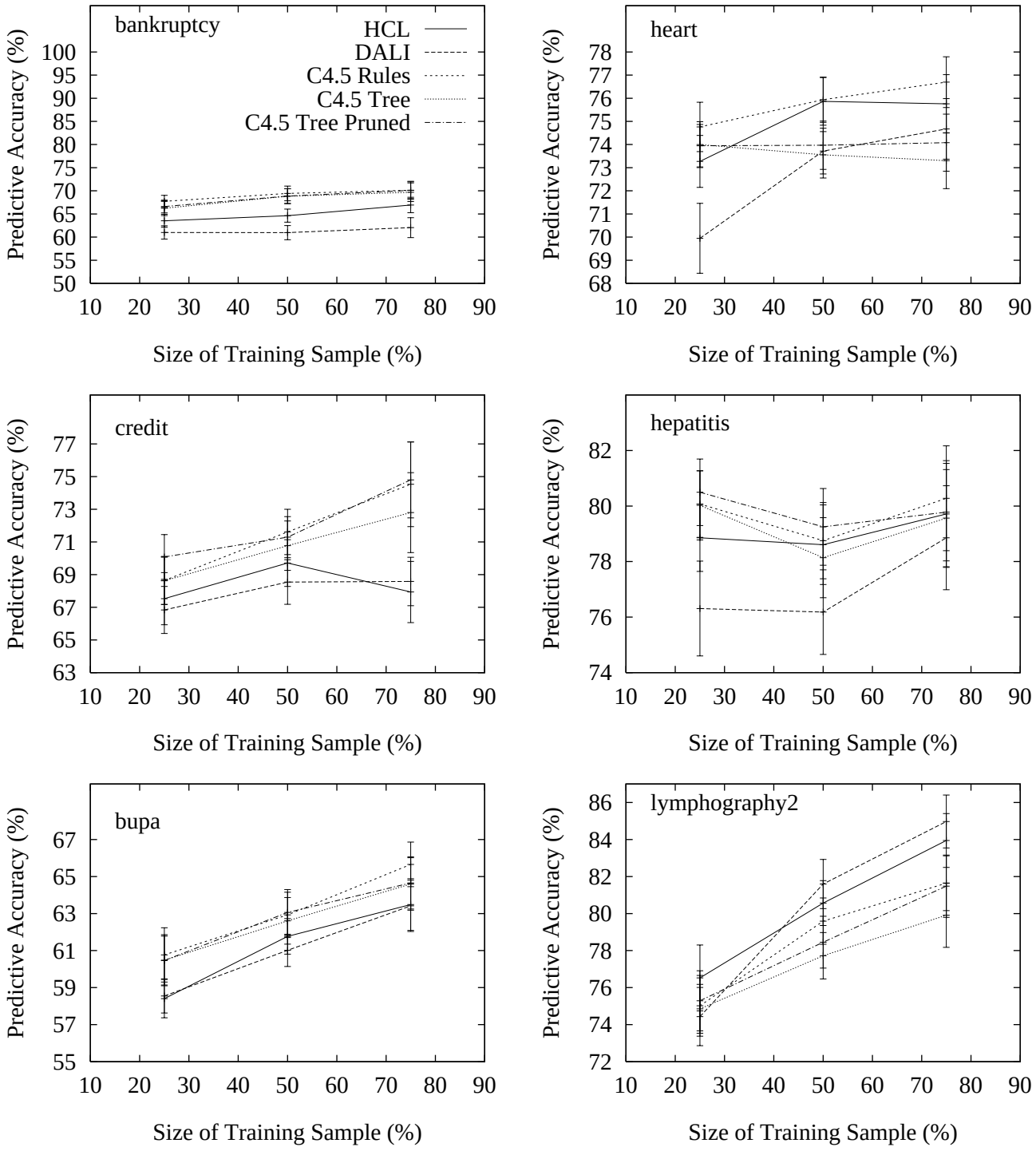


Figure 9.6: Training-set size vs. predictive accuracy comparing *HCL* with *DALI*, *C4.5*-rules, and *C4.5*-trees (pruned and unpruned). Domains might contain various sources of difficulty.

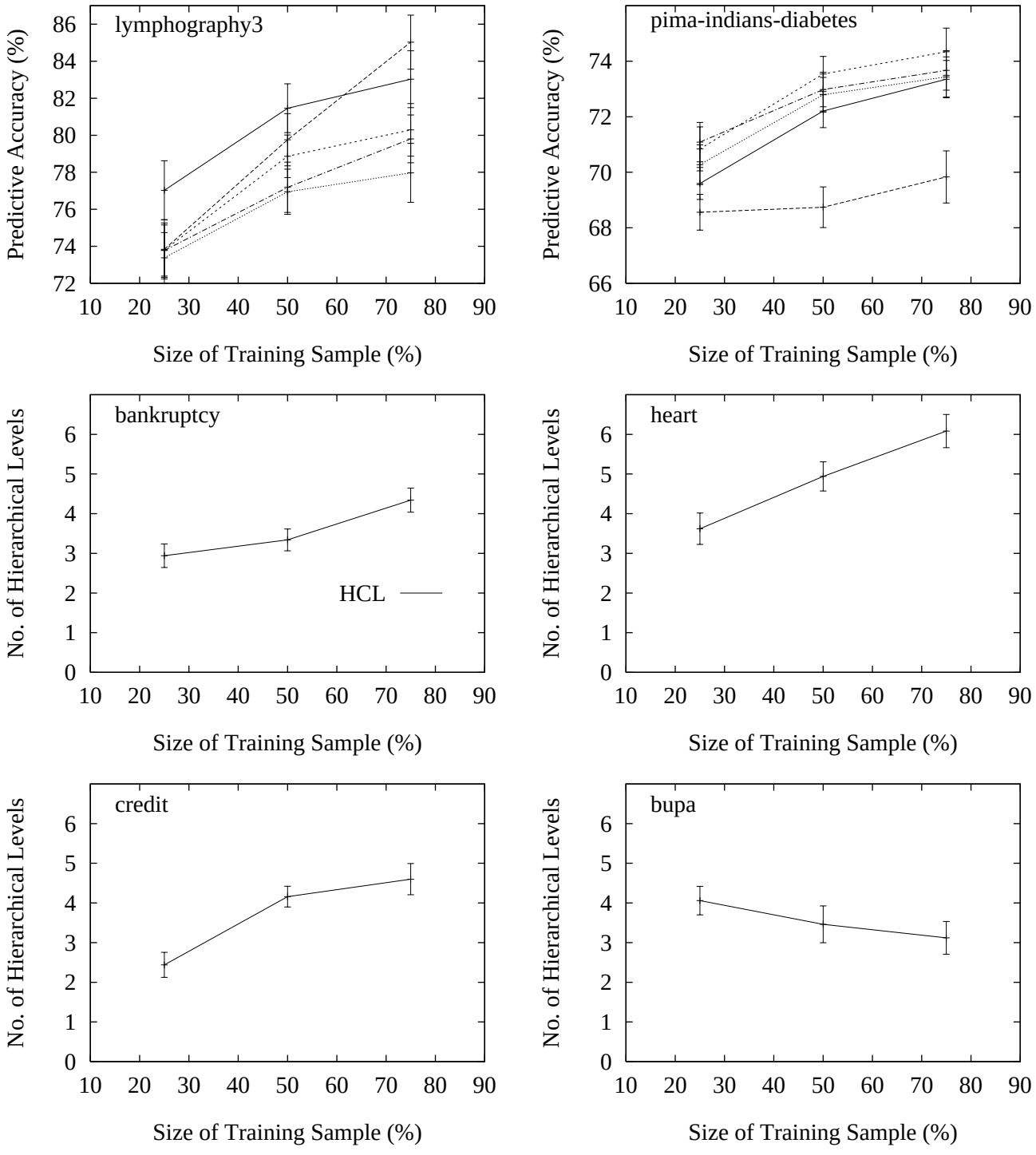


Figure 9.7: First two graphs: training-set size vs. predictive accuracy comparing *HCL* with *DALI*, *C4.5*-rules, and *C4.5*-trees. Last four graphs: Training-set size vs. no of hierarchical levels in *HCL*. Domains might contain various sources of difficulty.

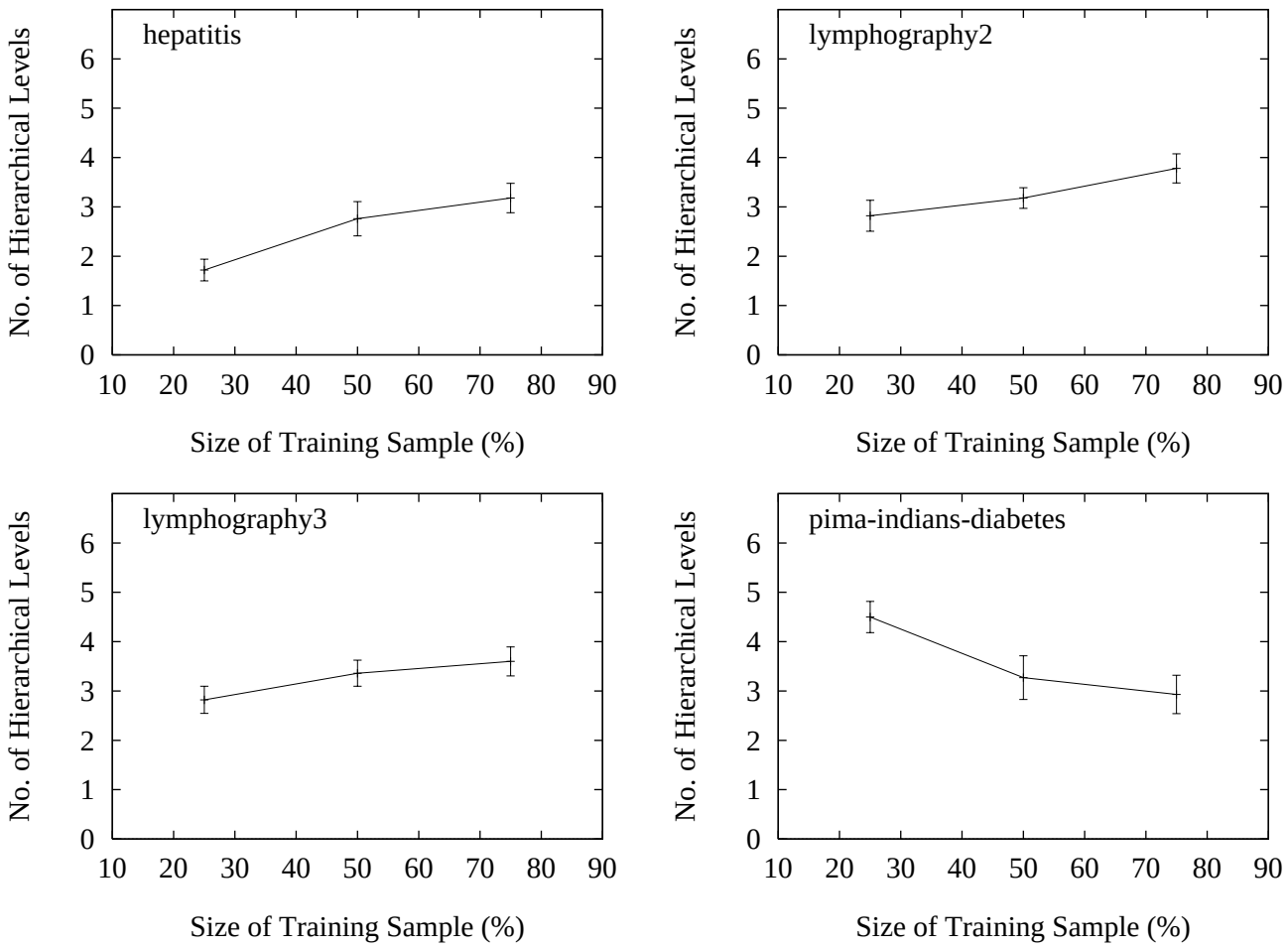


Figure 9.8: Training-set size vs. no of hierarchical levels in *HCL*. Domains might contain various sources of difficulty.

terms might lead to data overfitting. An example of a domain potentially affected by this problem is liver disorder (bupa), where *HCL* performs below average (Figure 9.6). The difference is significant with respect to *C4.5*trees and *C4.5*rules by more than 1%; performance is only similar to *DALI*.

- **Few examples (cross validation component)**
  Section 7.1 explains how adaptability in *HCL*'s design is implemented by incorporating a 10-fold cross-validation component that estimates the accuracy of the hypothesis at the current hierarchical level. A functional decomposition analysis revealed this component being sensitive to small-training sets; the component overestimates true accuracy significantly (Section 7.2). An analysis of the size of the training samples used in these experiments shows that, in average, the second group of domains (domains exhibiting various sources of difficulty) is of approximately 260 examples, compared to 400 examples in the first group (simple domains), and 600 examples in the third group (domains having high feature interaction only). Hence, the size of each training set in the second group of domains reported in Table 9.5, is in average of approximately 130 examples (training sets are fixed at 50% size). If we exclude the diabetes domain, the average is reduced to 170 examples, leaving only 85 examples (in average) for training. In these cases, the cross-validation component is expected to overestimate the true accuracy of the hypothesis at each hierarchical level (Figures 9.6).

- **Overfitting the examples used for accuracy estimation**
  Overfitting occurs when the hypothesis output by a learning algorithm is tailored to the training examples; as a result, accuracy over the testing set is poor. In *HCL* a similar situation is observed when examples in the training set are used for estimating predictive accuracy. Specifically, a mechanism that guides the learning process by searching a maximum in predictive accuracy, may overfit the examples used for such estimations. In those cases, accuracy on the training set starts to decrease, as long as predictive accuracy denotes a significant improvement. The result is a loss of accuracy over the training set. This may, to certain extent, be desirable (e.g., noisy domains). Nevertheless, a steep degradation of performance on the training set may damage prediction

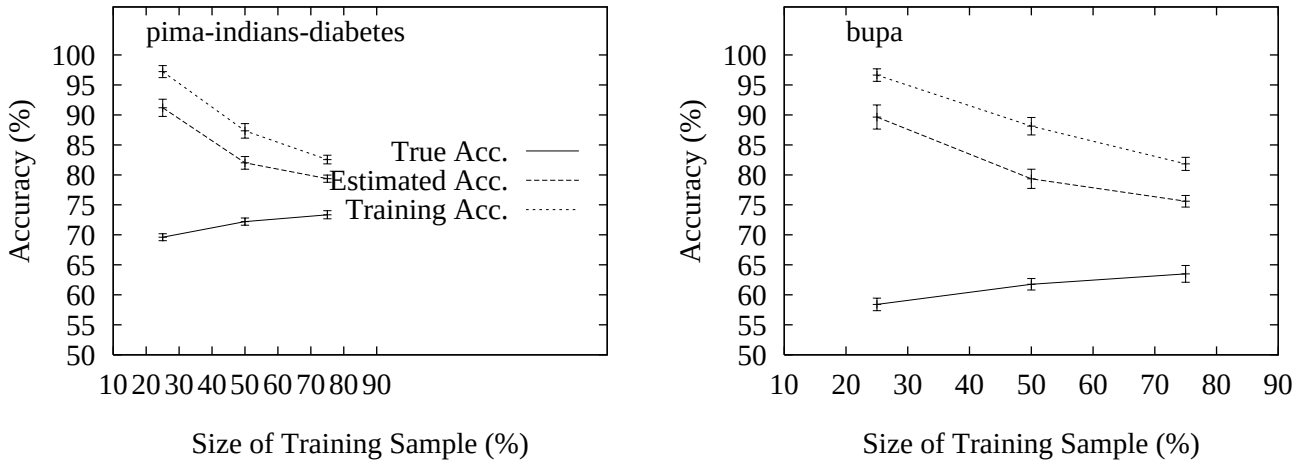


Figure 9.9: Training-set size vs. training, estimated, and true accuracy. In these domains, training and estimated accuracies decrease as the number of examples grows larger.

on unseen cases. Figure 9.9 shows two domains where this phenomenon is evident. In both the diabetes and liver disorder (bupa) domains, accuracy over the training set decreases as the training set is made larger. Predictive accuracy does not increase rapidly with more training examples. A similar situation occurs for the heart and credit domain (Figures 9.6), although the reduction in training accuracy on these domains is less than 1% (imperceptible in the graphs); in both cases predictive accuracy declines as the size of the training set increases.

Domains with High Feature Interaction as the Primary Source of Difficulty
The third group of domains can be categorized as difficult for most current learning models because of the expected high degree of feature interaction present (second row in Table 9.3). Information describing the domains in this group suggest high interaction does not combine with other sources of difficulty. This group includes board-game domains such as chess-end game (king-rook-vs-king-pawn) and tic-tac-toe, protein folding prediction, and others (e.g., ionosphere). One distinctive characteristic of these domains is that the set of primitive features is complete, in the sense that all the information required to discover the nature of the target concept is available, albeit in a low-level form (Section 2.4). Also, many intermediate terms must be constructed to find a good concept approximation. Section 2.5 refers to this group

as unfamiliar structured concepts, because current learning techniques experience difficulty in finding good approximations, despite the inherent concept structure.

For example, in board-game domains, features describing board configurations are the only available information to determine win or loss. As explained in Section 2.6, the rules of these games are well defined, and the examples comprise all relevant information. This suggests a concise representation can be obtained if we are able to discover the partial concepts lying between the primitive features and the target concept. In chess-end-game (king-rook versus king-knight), the number of possible states is of approximately 10^6, but new features can bring up high-level information about the game, reducing the number of possible states to a few hundred (Quinlan, 1983). Tic-Tac-Toe differs from chess-end game in that this domain comprises few entries enabling the analysis of all legal states; the game comprises 958 legal states, the concept of win can be described as a DNF expression of eight terms, each term conjoining only three literals (Matheus, 1989).

Another example is to use sequences of nucleotides to identify promoter instances. The problem is similar to the open problem in molecular biology, where we wish predict the secondary structure of a protein from sequences of aminoacids (primary sequence). As explained in Section sc:exs-domains, the information necessary to encode proteins originates on a gene-alphabet of only four letters and a language to decode that information. Secondary-structure prediction is governed by rules achieving a high degree of compression when compared to an extensional definition based on linear sequences of aminoacids.

In the ionosphere domain, inferring conditions of the ionosphere with complex numbers obtained from electromagnetic signals requires finding the (intricate) relations among those signals. But the concept can be stated in simple terms once those relations are unveiled.

Figure 9.10 reports on experiments measuring predictive accuracy over this group of domains. In the promoters, ionosphere, and kr-vs-kp domains, the advantage of *HCL* is significant. The advantage is particularly clear in the chess-end game domain. The advantage of *HCL* on the promoters domain is less evident than chess-end game because of the large standard deviations. On the tic-tac-toe domain, *DALI* and *C4.5*-rules perform similarly to *HCL*, possibly because in this domain the search for partial terms becomes a feasible task when compared to chess-end game, enabling

Table 9.6: Tests on predictive accuracy for real-world domains. Numbers enclosed in parentheses rerpresent standard deviations. A difference significant at the $p = 0.005$ level is marked with two asterisks.

Concept	*C4.5*trees-		*C4.5*rules	*DALI*	*HCL*	
	unpruned	pruned			pruned	standard
High Interaction						
ionosphere	89.63 (2.06)	89.50 (2.11)	89.80 (1.96)	89.24 (2.45)	88.34 (3.16)	90.40* (2.23)
kr-vs-kp	97.25 (0.85)	97.13 (0.96)	97.28 (0.83)	96.93 (1.05)	97.34 (0.89)	99.11** (0.23)
promoters	72.83 (6.16)	74.08 (4.52)	76.49 (6.63)	76.90 (7.19)	76.35 (6.26)	79.39** (5.24)
tic-tac-toe	82.06 (2.58)	80.26 (2.61)	93.92 (2.71)	94.54* (2.37)	93.31 (1.62)	93.51 (1.61)
AVRG	85.44	85.24	89.37	89.40	88.84	90.60*

other learning models to reach good results. Table 9.6 shows exact values for predictive accuracy at 50% training-set size. *HCL* with pruning denotes a disadvantage (significant) compared to *HCL* standard; further investigation is required to achieve improved conciseness without performance degradation. The difference with respect to *DALI* and *C4.5*-rules is of approximately 1% points (significant). On average, *HCL* outperforms *C4.5*-trees significantly by 5% points.

Figure 9.11 shows the number of hierarchical levels reached by *HCL* to output the final hypothesis. The greatest number of levels is seen on the board-game domains (tic-tac-toe: 5-6; kr-vs-kp: 4-5). The promoters and ionosphere domains show less number of levels (2-3). The numbers of levels reached by *HCL* in this group contrasts with the number of levels in the first group (i.e., simple domains). In this second group, *HCL* requires to build more intermediate terms to unveil the structure of the target concept. The construction of increasingly more complex terms at each new level is the result of a flexible and adaptive mechanism that responds in different ways according to the difficulty of the domain under study. The number of hierarchical levels reached by *HCL* can then be taken as a rough measure of the degree of complexity displayed by the domain under study. Previous sections experiment with artificial concepts where function complexity could be estimated (through concept variation ∇). This is possible when full class-membership information is available. Since this information cannot be obtained in real-world domains, the difficulty of the target concept can be assessed by relying on this metric.

In summary, empirical results show how domains in which feature interaction is

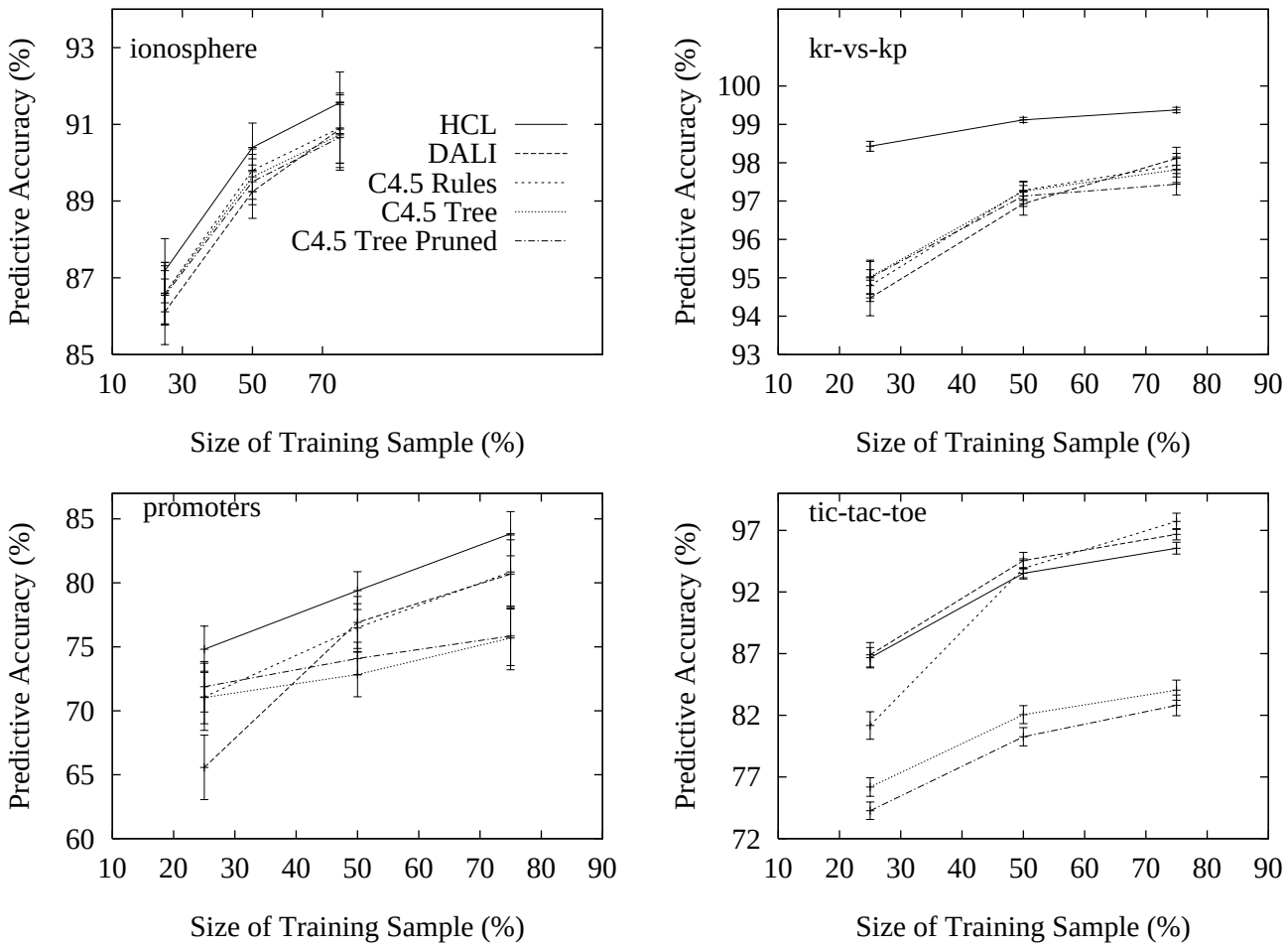


Figure 9.10: Training-set size vs. predictive accuracy comparing *HCL* with *DALI*, *C4.5*-rules, and *C4.5*-trees (pruned and unpruned). Domains are characterized by having high feature interaction.

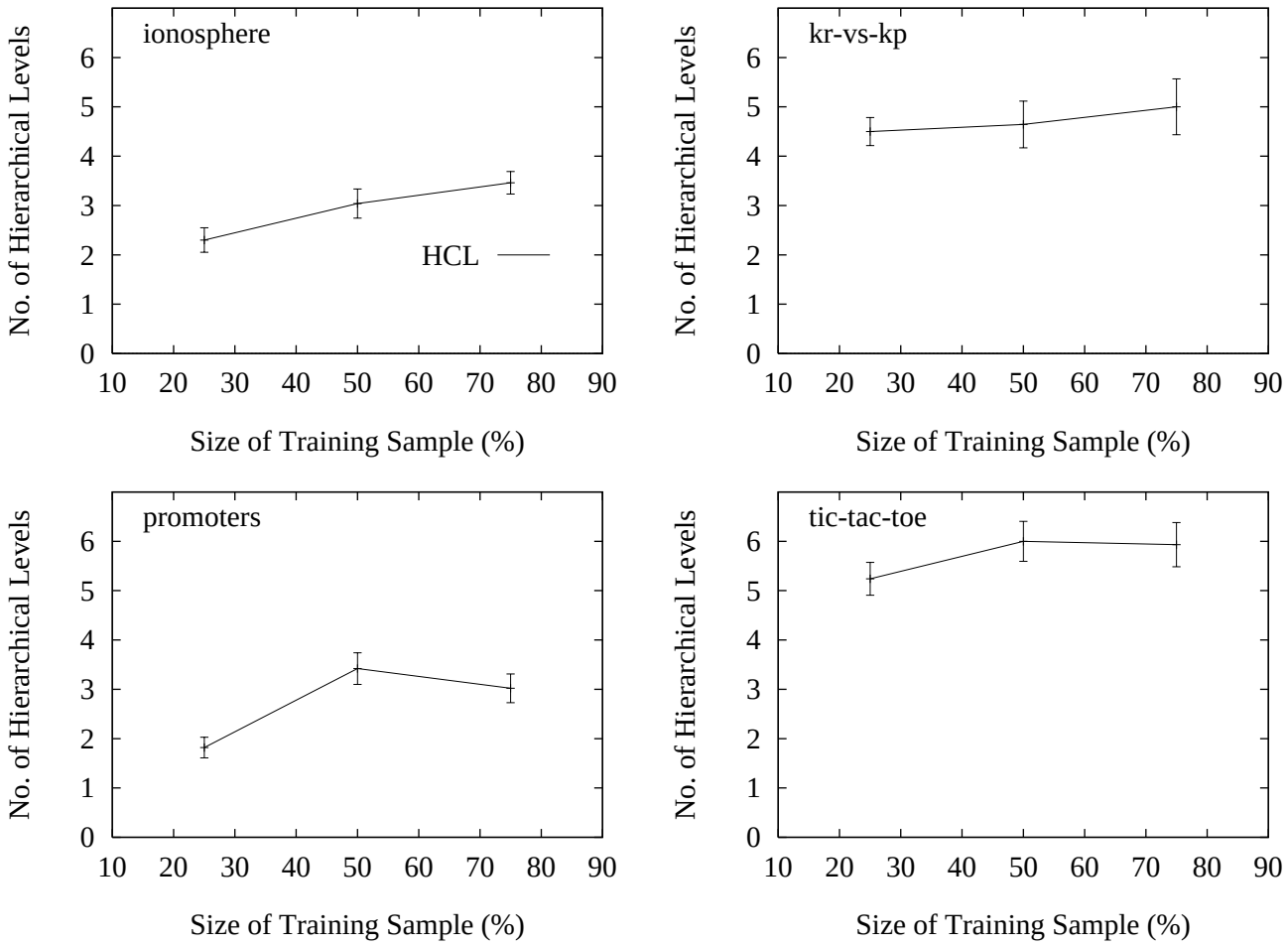


Figure 9.11: Training-set size vs. no of hierarchical levels in *HCL*. Domains are characterized by having high feature interaction.

the main source of learning difficulty match the design of *HCL*. When many intermediate terms are necessary to construct accurate approximations, *HCL* tends to outperform standard models significantly. Empirical results match the expectations for *HCL*'s performance over this group of domains (above-average, Table 9.3).

HCL shows improved performance on these domains because the mechanism is able to explore increasingly more complex representations through its flexible and adaptable mechanism;

HCL and the Development of New Learning Algorithms

The reason for the good performance of *HCL* in this group of domains lies specifically on its flexible and adaptable mechanism. *HCL* is able to adjust the complexity of the concept representation according to the domain under study. Since the nature of the domain is rarely known in advance, no assumptions can be made about the appropriateness of the language representation before learning starts. To tackle this problem, *HCL* explores different spaces of representation (one on each hierarchical level) until a hypothesis is found displaying best predictive accuracy. In contrast, most learning models assume a fixed concept-language representation (Section 2.5), restricting in this way the size of the class of domains where good performance can be obtained.

Section 2.7 explains how the development of new algorithms can be guided by the construction of operators that can transform unfamiliar structured concepts (i.e., difficult concepts) into simple concepts. Unfamiliar structured concepts are those where a concise representation is attainable (Kolmogorov complexity is low), despite most current techniques failing to yield good estimations (because of the low-level representation characterizing the examples). Figure 9.12 (a reproduction of Figure 2.6) depicts this idea more clearly. $\mathcal{S}^*_{\text{difficult}}$ represents precisely those unfamiliar structured concepts lying outside the applicability of current learning techniques, but within the set of structured concepts. The idea behind *HCL* is to progressively transform the learning problem until current learning techniques can be applied. If the concept lies within $\mathcal{S}^*_{\text{simple}}$ little needs to be done, and the problem is left unaltered. But if the problem lies within $\mathcal{S}^*_{\text{difficult}}$, *HCL* attempts to use external transformation operators (e.g., augment dimensionality of the instance space using logical expressions), and internal transformation operators (e.g., search for linear combination of expressions to delimit single class regions over the instance space), in order to simplify the learn-

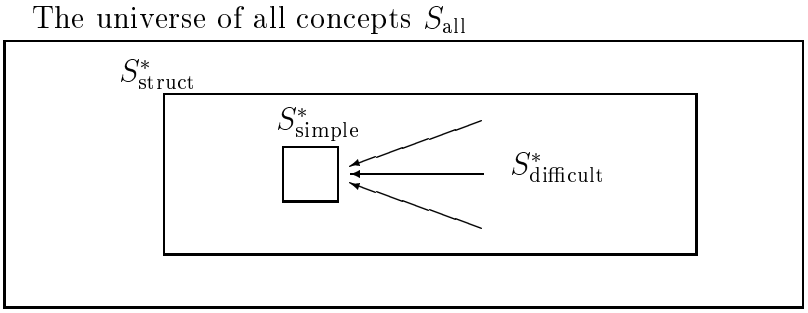


Figure 9.12: The set S^*_{struct} of structured concepts can be divided into two subsets: S^*_{simple} and $S^*_{\text{difficult}}$. A transformation process can modify the latter subset into a form similar to the former subset.

ing problem. Thus, *HCL* is an attempt to transform the learning problem to extend our coverage over the set of all structured domains (Section 2.4).

9.3 Conclusions

The beginning of this section set expectations for *HCL*'s performance on different scenarios (Table 9.3). Results over real-world domains confirm, in general, the major claims:

- When a simple representation is sufficient to uncover most of the structure of the target concept (i.e., no sources of difficulty appear to be present), *HCL*'s performance is similar to other standard models. In this group of domains, *HCL*'s adaptive mechanism stops building new levels at an early stage, because of a decrease of predictive accuracy detected by using more complex representations. Results using real-world domains in this group show *HCL* exploring few hierarchical levels (between one and two; Figure 9.5) to reach the final hypothesis. Results on predictive accuracy (Figure 9.4) show similar performance among the tested algorithms; average values are above 90% and within a range of 1% points (Table 9.4, last row).

- Domains exhibiting sources of difficulty other than high feature interaction (e.g., noise, representational inadequacy, small training samples) affect *HCL*'s performance. Documentation on these domains suggests there is either uncertainty regarding completeness of the feature set (e.g., heart disease, liver disorder), or

reason to believe feature and class values are corrupted by noise (e.g., lymphography). Results on real-world domains for this group (Figures 9.6 and 9.7) show *HCL*'s predictive accuracy around average (e.g., heart, hepatitis), and only in few cases below average (e.g., liver disorder). Plots indicating the number of hierarchical levels reached by *HCL* show values in the range $2.5 - 4.5$: *HCL* looks for complex expressions as a response to sources of difficulty introduced by the domain, which sometimes leads to a degradation in performance. In general, *HCL* is able to maintain performance levels similar to other models. This disagrees with original expectations predicting *HCL* being unable to learn on these domains (Table 9.3).

- Domains characterized by high feature interaction (only source of difficulty) match the design of *HCL*. An assumption is made that the features convey enough information to discover the structure of the target concept, despite the initial representation being too primitive. Flexibility in the design enables to vary the concept-language representation by exploring increasingly more complex hypothesis; adaptability helps in knowing when enough levels have been constructed. Results on this group denote *HCL* outperforming standard models significantly (e.g., chess-end game, promoters, ionosphere; Figure 9.10). The advantage of *HCL* goes up to 5% points when compared to *C4.5*-trees. The number of hierarchical levels reached by *HCL* in this kind of domains is high (e.g., kr-vs-kp: 4-5, tic-tac-toe: 5-6). The reason for *HCL*'s good performance lies on its ability to explore different spaces of representation according to the domain under study. Most current algorithms assume a fixed representation, thus restricting the kinds of concepts over which positive performance may be attained.

Results above agree with tests on artifical domains (Section 9.1). *HCL* is able to vary the complexity of the representation according to the difficulty of the domain, which on an artificial setting can be determined through concept variation (∇). On simple domains (low ∇), all models tend to perform similarly; a simple representation suffices to produce accurate estimations. On unfamiliar structured (i.e., difficult, high ∇) domains, *HCL* denotes a significant advantage over standard models, because of its dynamic search for complex representations.

Similar results were also derived from the functional decomposition analysis of *HCL* in Chapter 8. By assessing the different components in *HCL* we could observe how the flexible and adaptable mechanism in the algorithm is able to construct more hierarchical levels as a response to high variation and small training samples. In those situations many more models can fit the data, making learning a more difficult enterprise.

On real-world domains these results are evident when a distinction is made between simple and difficult domains (in addition, I analyze those cases where other sources of difficulty besides feature interaction are present). Despite the fact that no measure exists to classify real-world domains according to their inherent difficulty, background information about each domain proved useful to underline the importance of a learning mechanism that varies the representation dynamically. On the next chapter I summarize this thesis and propose new design paths for the improvement of *HCL*'s mechanism.

Part III

Conclusions

Chapter 10

Summary and Conclusions

This chapter is a summary of the two major parts of this thesis: a methodology for the development of new inductive learning algorithms, and the implementation and testing of *HCL* —an algorithm introducing flexibility and adaptability in the concept language representation. Each section adds some discussion and concluding statements. The last section describes future work aimed at improving/redefining the strategy behind *HCL*.

10.1 On the Development of Learning Algorithms

Chapter 1 explains how most research in machine learning has focused on creating new algorithms stemming from refinements to existing learning models (e.g., neural nets, decision trees, instance-based learners, rule-based systems, bayesian estimators). This is convenient when a model $\mathcal{M}$ already exists exhibiting satisfactory performance over the class of domains of study $\mathcal{S}$. It may occur, however, that no model over $\mathcal{S}$ performs adequately; the definition of a new learning model is then necessary.

As I argued in Chapter 1, the construction of a new learning model cannot rely on assumptions common to a process based on refinements alone: that a global evaluation metric is enough to guide the development process (e.g., a comparison based on predictive accuracy alone), and that the strategy of the model is always correct (inherent limitations in the strategy may exist). When a model is defined from scratch, I recommend that we should adopt a functional view. This means we must first define the set of functionalities that the new algorithm will perform. A functionality is defined here as an operation we set as a goal for the new algorithm to carry out. It responds to the question: what are we striving to attain with the new algorithm? (e.g., to cope with feature and class noise? to generate hypotheses amenable to interpretation and analysis? to elucidate feature interaction effectively? etc.). After implementation, a functional decomposition analysis can assess how well

the different components in the algorithm are able to carry out such functionalities.

Discussion

Section 1.6 explains how the development of new learning algorithms (based on the attainment of goal functionalities) gives rise to several questions, the answer to some of which appears more clear from this thesis. A summary is given next:

- Precisely which and how many functionalities must be implemented before significant improvement is observed over standard learning models?
  To answer this question we must consider the class of domains under study $\mathcal{S}$. If the bias of a standard model already matches the characteristics of $\mathcal{S}$, the new methodology becomes superfluous. On the other hand, we may find that $\mathcal{S}$ is difficult to learn by most current models. Improvement over standard models is expected when we are able to identify the most relevant sources of difficulty $\{P_i\}$ present in $\mathcal{S}$, and to arm the new algorithm with precisely those functionalities that can deal with each P_i.

- How can we avoid the limitations attached to current models within the new design?
  This question highlights the importance of a functional decomposition analysis. By assessing the individual contribution of each component in a learning algorithm, we become able to identify possible deficiencies and limitations. We can then propose a new model explicitly designed to improve previous strategies.

- Under what conditions is the construction of a new model necessary?
  In conclusion, the proposed methodology is useful when the following conditions are met: 1) no model performs satisfactorily over $\mathcal{S}$, and 2) the new strategy is explicitly designed to attack limitations existing in current models.

Fundamental Issues and The Goal of Inductive Learning

Chapter 2 shows how no algorithm can be deemed superior to any other algorithm unless we specify the class of domains over which the comparison is taking place. I argued that the role of inductive learning is to learn over all *structured domains*, where a highly compressed representation is always attainable. This implies poor performance may result somewhere outside this set (e.g., random concepts). This chapter provides a characterization of all structured domains using a measure of

complexity known as Kolmogorov complexity (Section 2.4). Specifically, for a fixed concept C, I denote the complexity of C as

$$K_{D,m}(C) = E(K_{C,D,m}(T)) = \sum_{\text{all } T_i \text{ of size } m} P(T_i)\ K_{C,D,m}(T_i) \qquad (10.1)$$

where, $K_{D,m}(C)$ is the expected value of the Kolmogorov complexity (i.e., degree of randomness) of training set T (conditioned on C), for a fixed size m, and a fixed distribution D over which the examples of T are drawn (Section 2.3). This definition serves to define the set of all structured domains, S^*_{struct}, as $\{C \mid Y(C|K(C)) > 1-\gamma\}$, where γ is user defined, and $Y(C|K(C))$ is the probability of correctly approximating C conditioned on $K(C)$.

Section 2.4 explains how common assumptions followed during the design of learning algorithms (simplicity and similarity) make a certain class of concepts hard to learn, even when their Kolmogorov complexity is low. In this class of domains, lack of expertise in crafting new features results in a low-level representation: information is missing to conform a representation explicitly showing the interactions among primitive features. A low-level feature representation makes learning hard, because it produces high variation in the distribution of examples along the instance space: examples lying close to each other often disagree in class value (Section 2.4). A different way of saying this is that the instance space is characterized be having many disjuncts or peaks (Rendell & Cho, 1990), that the distribution of examples is highly irregular, that the concept is complex, difficult, or displays high feature interaction.

Current algorithms assume primitive features are highly representative of the target concept; their design disregards the importance of implementing operations to bridge the gap between the representation of the primitive features and the representation of the target concept. This all means we have been able to learn only a fraction of all structured domains. Unfamiliar structured domains, in which finding intermediate concepts is necessary to yield good concept approximations, are exemplified by parity, board-game domains, protein folding prediction, and others (Section 2.6).

Discussion

Section 2.7 explains how one possible way to learn unfamiliar structured domains is to transform these domains into a form that matches current learning techniques. For such difficult domains, one kind of transformation consists of building the terms

necessary to increase the complexity of the feature representation until it matches the complexity of the target concept. This is one approach to extend our coverage over domains that denote structure and regularity. In contrast, algorithms that use fixed representations can exhibit satisfactory performance only on a limited class of domains.

In summary, the development of transformation techniques is ultimately hampered by the concept-language representation. High level transformation operators may be necessary to learn relevant patterns (e.g., win in board-game domains, predict secondary structure in proteins, etc.). Thus, independently of the sophistication attached to our learning algorithm, the concept-language representation limits the space of possible patterns. Since such a space may omit important concepts, it is important to investigate representations necessary to extend our coverage over the set of all structured domains.

Basic Steps for the Development of New Learning Algorithms

After all background information is explained (Chapters 1 and 2), Chapter 3 outlines basic steps for the development of new learning algorithms. The basic steps are summarized as follows:

1. Analyze the class $\mathcal{S}$ of domains of study. The idea is to assign one metric to each property or characteristic P (e.g., noise, feature interaction) present on each domain in $\mathcal{S}$.

2. Based on the characteristics of the class of domains under study, the next step is to define a set of goals and a design strategy. This step essentially consists of enumerating the set of functionalities that we intend to accomplish in the new algorithm (e.g., using a specific concept-language representation, adapting the learning bias dynamically, producing interpretable hypotheses, etc.).

3. Build, isolate, and test components. This step deals with the implementation of all component necessary to reach the specified functionalities. It also includes the design of experiments to verify that each component operates correctly.

4. The assembling process. After implementation we need to arrange all components into a specific sequence: a sequence matching the algorithm's basic design.

Discussion

Despite being general, the basic steps outlined in Chapter 3 convey important ideas: 1) the set of goal functionalities must be made clear from the beginning, and the design and implementation must be oriented to accomplish such functionalities; 2) much can be learned from a functional decomposition analysis; by analyzing each component individually we can detect possible deficiencies; 3) it is important to propose a design in which all components are made explicit, and thus amenable to modification or replacement. Regarding the last idea, Section 3.2 explains how most current learning models can be separated into fixed, flexible, and other components, and that only the last two kinds are subject to modification. In contrast, this thesis underlines the importance of making each component accessible, in order to redesign the basic strategy whenever necessary, even if that entails a new approach to learning.

Case Studies Supporting The Proposed Methodology

To provide evidence supporting the importance of a functional-view in the design of new learning algorithms, Chapters 4 and 5 describe two experimental case studies. The studies answer additional questions posed in the introductory chapter (Section 1.6): How effective is a functional decomposition analysis to identify possible limitations in the design? Is the implementation of functionalities through a set of components always a feasible task?

Isolating and Analyzing Components

The first study (Chapter 4) shows the importance of decomposing an algorithm into its constituent components to identify inherent strategy limitations. The study analyzes the strategy behind top-down decision-tree inducers. An inherent limitation in this strategy is known as the fragmentation problem, where the continuous partitioning of the instance space progressively lessens the statistical support of every disjunctive term in the final hypothesis (Section 4.2). Whereas previous studies look into solutions to circumvent the fragmentation problem, this study identifies the two causes from which the problem originates: 1) the coverage of disjunctive terms (i.e., tree branches) is mutually exclusive, such that an example cannot be intersected by two disjunctive terms, and 2) each partition is too coarse; many steps are required to delimit a single-class region in the instance space. These two sources are responsible

for a progressive loss of statistical support at every disjunctive term, the effect being further aggravated as the amount of concept variation (i.e., the probability that two neighbor examples differ in class value) increases.

By performing a functional decomposition analysis over a standard rule based system (*C4.5*-rules), I separated out the contribution of two different components. The first component, named global data analysis, evaluates each rule (i.e., disjunctive term) of the final hypothesis by recurring to all training examples (as opposed to only subsets of examples in decision tree induction). The second component eliminates rules (of the final hypothesis) based on a minimum-description-length principle. The analysis shows the global-data-analysis component having the greatest contribution in solving the fragmentation problem. The minimum-description-length component shows a minor contribution to the overall performance of the rule-based system. The individual effect of each of these components to the rule-based system had not been investigated before; it is interesting to observe how the global-data-analysis component is mostly responsible to recover the loss of statistical support suffered by each disjunctive term due to the fragmentation problem.

How effective is then a functional decomposition analysis to identify possible limitations in the design? From this study we can conclude that a functional decomposition analysis clarifies the effect that each component exerts during learning. In this way, possible limitations inherent by the design become easy to identify.

Effectively Combining Components

While the first study shows the importance of decomposing an algorithm to evaluate the individual contribution of each component, the second study shows how components from different strategies can be combined into a single mechanism. Chapter 5 explains how a decision tree can be enhanced by creating new logical expressions at every tree node (specifically monomials —conjunction of feature values or their negations). This feature-construction component is unstable, in that variations of the training sample may produce different new expressions. I show how the instability effect can be corrected by incorporating multiple-classifier techniques (e.g., *bagging* and *boosting*) locally, at every node of a decision tree; every splitting function becomes in this way a linear combination of logical expressions. Experimental results show, on average, an advantage in predictive accuracy when these techniques are used locally, at each node of a decision tree, compared to a decision tree constructing

a single logical expression at every tree node.

Is the implementation of functionalities through a set of components always a feasible task? The answer depends on how many components are available, and on the possibility of extending the applicability of each component. In other words, we wish to extract the principles explaining why a component is successful on a certain application, and then to be able to apply those principles elsewhere, where evidence suggests of their potential value. In this study, for example, the stabilization of the feature construction component was attained by changing the application of multiple-classifier techniques from a global application over whole hypotheses, to a local application inside decision trees. I claim the implementation of a functionality can in most cases be a feasible task if we are able to 1) extract components from current techniques and 2) generalize those components in order to apply them in multiple ways.

Discussion

Research in machine learning has often been characterized by the analysis or application of standard learning models. Research has been devoted to perfect current strategies or to adapt such strategies to certain class of domains (Chapter 1). An additional avenue of research is to explore —at a more refined level than an entire algorithm itself— the individual contribution of each component (Chapter 4). If such a study is performed over different models, we would not only be able to identify limitations in current designs, but also to propose new strategies based on the combination of existing components (Chapter 5). It may be argued that domains where satisfactory performance has already been achieved could be excluded from this study, and that in these cases we could simply keep applying existing approaches to learning. But an important goal in machine learning is to expand our current learning mechanisms to cover an even larger set of structured domains (Chapter 2). This can be possible only if a better understanding of what constitutes an effective learning mechanism is accomplished.

10.2 Improving Performance Through Flexible and Adaptable Representations

The second part of this thesis deals with the design and implementation of a new learning algorithm: *HCL* (Hierarchical Concept Learner). This part has two main goals: to put into practice the basic steps for the development of new algorithms (Section 3.4), and to assess the performance of *HCL* in both artificial and real-world domains.

Chapter 6 defines the general strategy behind *HCL*. The main idea is to construct consecutive layers in a hierarchy of intermediate concepts (i.e., partial hypotheses); the bottom of the structure accommodates original features whereas the top of the structure contains the final hypothesis. The hierarchical structure is intended to capture the intermediate steps necessary to bridge the gap between the original features and the (possibly complex) target concept (Section 6.2). *HCL* achieves two functionalities: 1) flexibility in the representation, by increasing the complexity of the hypothesis comprised at each hierarchical layer, and 2) adaptability in the search for different representations, to know when to stop adding more layers in top of the hierarchical structure. Chapter 6 also specifies the class of domains for which *HCL* is designed (Section 6.1).

Chapter 7 details the implementation of *HCL*. Here, each component is isolated and analyzed in detail; the goal is to generate a clear view of the algorithm's mechanism (Section 7.1 and Section 7.2). In this chapter, the analysis of each component is analytical, rather than empirical, which helps to explain the rationale behind *HCL*'s implementation. Section 7.3 shows how all components are integrated into a specific sequence aimed at achieving the functionalities mentioned above.

Discussion

Chapter 7 answers a question posed in the introductory chapter (Section 1.6): What lessons can be drawn from the design and implementation of the algorithm?

- Two steps in the methodology of Section 3.4 are relatively simple to accomplish: define the class of domains of study $\mathcal{S}$, and arrange the implemented components into a specific sequence. The difficulty attached to the first step lies in the search for metrics that can identify the degree to which a given property (e.g., feature interaction) is present in $\mathcal{S}$. This was not an obstacle

in *HCL*: the algorithm is designed to deal with various degrees of feature interaction, roughly estimated through variation ∇ (Section 2.5). When ∇ is not known, (e.g., real-world domains) the number of hierarchical levels in *HCL* estimates the complexity of the domain under study (Section 9.2). The second step —arranging components into a specific sequence— is straightforward after a strategy and functionalities have been defined.

- Searching for the right components given the functionalities is a more complex task. Here we must know what components are available, and what their effects are during learning. A functional decomposition analysis can help elucidate the contribution of each component. Additionally, we need to define local metrics that can assess the performance of components, aside from the criteria employed to evaluate the entire mechanism.

- The most difficult step is to create a new strategy, together with a specification of a set of goal functionalities. In *HCL*, a hierarchical structure seemed the most natural choice in searching for different degrees of representational complexity, while at the same time capturing intermediate concepts. Nevertheless, different problems may arise with apparently no clear strategy to follow. Further investigation is necessary to suggest more specific guidelines to produce new strategies.

Experiments Testing *HCL*

A Functional Decomposition Analysis

This section responds to the question (Section 1.6): How well are the functionalities in the design achieved by the proposed implementation? In Chapter 8, *HCL* is decomposed into different parts, each part assessed individually (experiments use artificial domains). Regarding flexibility in the design (Section 8.2), *HCL* is effective in building more hierarchical levels when concept variation is high and when the size of the training set is small (the number of levels goes up to approximately 3 to 5 levels). Regarding adaptability in the design (Section 8.3), *HCL* tends to overestimate the true accuracy of the hypothesis obtained at the final hierarchical level. Under high variation (i.e., the probability that two neighbor examples differ in class value) the overestimation can go up to 40% points accuracy. In addition, *HCL* is compared to a different version of the algorithm where only one hierarchical level is constructed

(Table 8.1). *HCL* standard performs significantly better when the domain is difficult (i.e., the domain has high variation). The difference varies between 4% and 16% points in accuracy.

In the search for logical expressions, the feature construction component (Section 8.4) proved effective in eliminating large portions of the space of all possible expressions (on average, for every constructed feature approximately another feature is eliminated). A pruning component in *HCL* improves conciseness, but with certain loss in predictive accuracy (difference is of approximately 1% in average).

A Comparison with Standard Learning Models

This section responds to the question (Section 1.6): What is the performance of the algorithm when compared to standard models? In Chapter 9, *HCL* shows a general advantage in predictive accuracy over several standard learning models when using artificial domains (domains exhibiting characteristics for which the algorithm is designed). The advantage observed by *HCL* in these domains (up to between 30% and 40% point accuracy when compared to *C4.5*-trees and k nearest-neighbor) is more evident when variation is high (Section 9.1). CPU time in *HCL* is extremely costly: the time spent for a single run can be orders of magnitude above *C4.5*-trees.

In real world domains (Section 9.2), results vary according to the class of domains under study. *HCL*'s performance is similar to standard models when the domain is simple (few intermediate terms characterize the target concept). Domains exhibiting sources of difficulty other than high feature interaction affect *HCL*'s performance; in these domains predictive accuracy is around average. Domains characterized by high feature interaction match the design of *HCL*. Here, *HCL* outperforms other models significantly (difference goes up to 5% points when compared to *C4.5*-trees), e.g., chess-end game, promoters, ionosphere. In these domains, the flexible and adaptable mechanism of *HCL* is able to explore enough levels of representational complexity to bridge the gap between the primitive features and the target concept.

Results on both artificial and real-world domains can be unified by looking into those situations where *HCL* improves over standard models. On artifical domains, *HCL* has an advantage when variation is high ($\nabla > 20\%$). Concept variation (∇) is a rough estimation of the amount of feature interaction. High interaction among features means each feature alone conveys little information about the structure of the target concept. The result is an irregular distribution of examples throughout

Table 10.1: Averaged results on predictive accuracy for both artifical domains ($\nabla > 20\%$) and real-world domains.

Concept	*C4.5*trees-		*C4.5*rules	*DALI*	*HCL*
	unpruned	pruned			
AVRGReal – world	85.44	85.24	89.37	89.40	90.60*
AVRGartificial	71.11	72.06	77.70	86.09	86.93*

the instance space, for which most learning algorithms are unprepared (because of the similarity-based assumption; Section 2.5). On real-world domains, *HCL* denotes good performance when no sources of difficulty are present except for high feature interaction. These unfamiliar structured domains are characterized by a low-level example representation, such that many intermediate concepts must be discovered before a good concept approximation is reached.

Artificial domains exhibiting high variation and unfamiliar structured real-world domains favor the design of *HCL* because both groups require the discovery of interactions among features. This is necessary to bridge the gap between the primitive initial representation and the (possibly) high-level representation characterizing the target concept. To support this claim Table 10.1 shows the average performance of *HCL* over artificial domains where $\nabla > 20\%$ (first row), and the average performance of *HCL* on real-world domains where feature interaction is assumed the only source of difficulty present (second row). Both results denote a significant advantage of *HCL* over a standard decision tree and rule-based system. The difference is more evident on artificial domains where conditions are controlled (interaction raises to levels that are uncommon among real-world domains). *HCL* outperforms (on average) *C4.5*trees by approximately 14% points and *C4.5*rules by almost 10%. On real-world domains (same as Table 9.2), the difference goes up to 5% when compared to *C4.5*trees. Thus, *HCL*'s design is effective in learning structured concepts characterized by high feature interaction.

Limitations and Disadvantages

The experimental analysis of Chapters 8 and 9 identifies several deficiencies in the current implementation of *HCL*:

- Like any other algorithm drawing statistical inferences from data, *HCL* is sensitive to the size of the training sample. Specifically, when the training sample is small, the 10-fold cross-validation component overestimates the true accuracy almost always significantly; the effect is evident in small training samples

(around 100 examples) where the component becomes biased.

- The feature construction component is affected by noise in feature and class values (the same effect can be observed in almost any feature-construction algorithm). Under those circumstances, the newly constructed expressions tend to do poorly in discriminating examples having different class values. The effect gets progressively worse as new expressions build over previous expressions already corrupted by the noise signal.

- When the set of features characterizing the examples is incomplete, in the sense that no information exists to discover all possible terms of the target concept, *HCL*'s performance degrades. In these cases the algorithm is expected to continue building new expressions, even though high-level terms may lead to data overfitting.

- CPU time is extremely costly. The time spent for a single run in the experimental section can be orders of magnitude above *C4.5*-trees (in the worst case).

- The feature construction component explores the space of logical expressions by adding one literal at a time to previous expressions. This process may never reveal complex feature interactions, because each feature alone is unable to discriminate examples of different class values. A solution could be to assess the value of subsets of feature values (i.e., the value of subspaces over the instance space) to capture those interactions.

The next section (Section 10.3) explains future directions driving *HCL* into more promising design paths.

10.3 Future Work

In this section I try to respond to the question (Section 1.6): Is the current implementation amenable to extensions through refinements? Is the basic strategy easy to reconfigure? The design of *HCL* can be modified in two different ways: by altering/replacing the different components, and by reorganizing components into a different sequence (i.e., into a new strategy). I analyze two possible courses of action based on these alternatives.

Learning Perceptrons

One way to augment flexibility in the representation of *HCL* is to replace the component that brings a logical expression from each replicate of the instance space (Search-Expression, Figure 7.4), to a component that instead brings a perceptron. Each partial term in *HCL* would now be represented as a linear combination of primitive features and previous expressions. The final hypothesis would be a linear combination of perceptrons. The new mechanism could be seen as a neural network dynamically deciding on the network topology. The idea is to recruit neurons at a single layer until a hypothesis to the target concept is found [1].

But, as explained before (Section 3.1), refining a learning model does nothing to modify the underlying strategy. Augmenting the quality of the partitions exerted at each node of a decision tree does not eliminate inherent limitations in the basic strategy (e.g., does not eliminate the fragmentation problem, Section 4.2). Thus, different from perceptron trees, the modified *HCL* builds over a strategy where limitations are known, and where we have already identified the class of domains over which the algorithm is expected to outperform standard models (Section 9.2). For example, while adding perceptrons in *HCL* could come with significant gains in accuracy, some disadvantages are gained: interpretability in the original *HCL* would be lost, as each partial concept is now represented in terms of weighted nodes. A thorough empirical evaluation can provide enough information to decide how much gain is obtained with the new modification.

Improving CPU Time

HCL could be enhanced by reducing CPU time. Such functionality could make *HCL*'s execution time competitive: the current —sequential— implementation results too slow for any practical applications. A component affecting *HCL*'s run time is the stratified 10 fold cross-validation. Section 7.2 describes how at each hierarchical level, this component estimates the accuracy of a hypothesis that builds over the set of original features and all previously constructed terms. If the estimation improves over that obtained at the level below, the mechanism continues building new hierarchical layers. Notice that the execution time in *HCL* could be reduced by computing the

[1] The idea above is similar to that of perceptron trees (Utgoff, 1988), or, in general, to the idea of using non-axis-parallel partitions as splitting functions in decision trees (Brodley & Utgoff, 1995).

cross-validation component in parallel. In theory, execution time could be reduced by a factor of k, where k is the number of cross-validation folds (ignoring overhead time associated to the parallelization).

Learning Transformations

Two kinds of difficulty in a domain are 1) high feature interaction, and 2) the presence of patterns obtained from the composition of separate regions over the instance space (Section 2.4). In this section I elaborate on the second source of difficulty, and suggest on new directions in which *HCL* could expand the space of patterns being explored.

Many functions can be expressed as the composition of patterns that repeat periodically over time (or other independent variables). Current classification methods view learning as the search for a consistent decision criterion that can distinguish examples belonging to different classes. A hypothesis is basically a function over the original features that splits the instance space into different regions. This view, however, limits the class of patterns that can potentially be discovered. In particular, it limits the search for patterns to a first level of abstraction (Rendell, 1985). Consider that clusters of examples representing single class regions could also interact among other clusters in different ways, in which case no pattern can capture such interactions. The idea can be extended farther out by simply considering that interactions may also exist among clusters of clusters of examples.

HCL could be expanded to learn patterns among different regions of examples in the following way. Each layer of abstraction in the hierarchical structure represents a linear combination of partial concepts. Before the next layer of abstraction is constructed, the algorithm could be armed with transformation operators that can explain the way different partial concepts interact. By a transformation operator I mean a function over partial concepts that can suggest on possible abstract relations (other than logical combinations) such as symmetry or periodicity. Knowing what transformation operators are applicable at a certain layer can rely on knowledge about the domain under study. This strategy could reveal interesting patterns in those real-world domains for which abstract descriptors are expected to be relevant (e.g., weather prediction, protein-folding, stock-market prediction, etc.).

10.4 Summary of Contributions

To conclude, Tables 10.2 and 10.3 give a summary of the contributions contained in this thesis (some of them were mentioned before in Section 1.5). Table 10.2 concentrates on the development of inductive learning algorithms (part I of this thesis); Table 10.3 concentrates on the design and development of *HCL* (part II of this thesis). On each table, the first row mentions what the contribution is, and where it is located. The second column briefly details on each contribution. It is my hope that these contributions will prove helpful in the continuing effort to increase our knowledge about the automation of learning.

Table 10.2: Summary of contributions regarding the development of new inductive learning algorithms (part I).

Form of the Contribution	Benefits of the Contribution
A functional view in the design of new learning algorithms Chapter 1, Section 1.3	The development of new algorithms can be guided by explicit attention to specific functionalities.
A characterization of all structured domains Chapter 2, Section 2.3	Formally characterizes all structured domains according to their inherent complexity. The characterization enables us to identify unfamiliar structured domains where most current learning techniques experience difficulties.
Basic steps for the development of new algorithms Chapter 2, Section 2.3	Outlines basic steps for the definition of the mechanism and strategy of a new learning algorithm. The design follows predefined goals, and considers characteristics of the class of domains under study.
Evidence supporting the importance of adopting a functional view during the design process Chapter 4, Section 4.3 Chapter 5, Section 5.3	Case 1: A functional decomposition analysis over a rule-based algorithm (*C4.5*rules) identifies two components. One component (global-data-analysis) effectively attacks the fragmentation problem. The second component (minimal-description-length principle) shows a minor contribution to the system (contrary to common belief). Case 2: The study explains how techniques to combine classifier techniques (*bagging* and *boosting*) can be applied at each node of a decision tree (with accuracy gain). The study supports the claim that components from different strategies can be combined into a single mechanism.

Table 10.3: Summary of contributions regarding the design and development of *HCL* (part II).

Form of the Contribution	**Benefits of the Contribution**
Design and Implementation of the new *HCL* algorithm. *HCL* adds a flexible concept-language representation and adaptability in learning Chapters 6 and 7	The algorithm allows building a hierarchy of partial subconcepts. The algorithm introduces two main functionalities in the design: 1) flexibility in the concept-language representation builds new logical expressions from primitive features and previous expressions to progressively increment the complexity of the representation, and 2) adaptability in learning evaluates the degree of complexity at each new level of representation. This evaluation dynamically finds the hypothesis having the right degree of representational complexity.
Common problems in machine learning attacked by *HCL* Section 7.4	The *HCL* algorithm addresses common concerns in machine learning: The bias-variance problem. Low variance is expected on every hypothesis as a result of using multiple-classifier techniques. Bias is reduced by increasing the number of possible hypothesis at each new hierarchical level. Data overfitting is attacked by estimating the off-training accuracy of the most current hypothesis using stratified 10-fold cross validation.

Appendix A

Definitions for Artificial Concepts

Let: $X = (x_1, x_2, \ldots, x_n)$,
$\text{address}(x_1, x_2, \ldots, x_m) = 2^0 x_1 + 2^1 x_2 + \ldots + 2^{m-1} x_m$

Definitions:

DNF9a : $x_2x_3 + \bar{x_2}x_3x_7 + x_2\bar{x_3}x_8 + x_2\bar{x_7}\bar{x_8} + \bar{x_3}x_7x_8$
DNF9b : $x_1x_2x_3 + \bar{x_1}\bar{x_2} + x_7x_8x_9 + \bar{x_7}\bar{x_9}$
DNF12a : $x_1x_2x_9x_{12} + x_1x_2\bar{x_9}x_{11} + \bar{x_1}x_2x_5x_9 + \bar{x_1}\bar{x_2}x_8\bar{x_9}$
DNF12b : $x_1x_2\bar{x_7}\bar{x_8} + x_1x_2x_{11}x_{12} + x_1\bar{x_2}\bar{x_{11}}x_{12} +$
$\bar{x_1}x_2x_{11}\bar{x_{12}} + \bar{x_1}\bar{x_2}\bar{x_{11}}\bar{x_{12}} + x_7x_8x_{11}x_{12}$
CNF9a : $(x_4 + x_6)(\bar{x_4} + \bar{x_5} + x_7)$
CNF9b : $(x_1 + \bar{x_2} + x_3)(\bar{x_1} + x_2 + x_5)(x_1 + x_2 + \bar{x_3})$
CNF12a : $(x_8 + x_9)(x_7 + x_6 + x_5)(\bar{x_8} + \bar{x_6} + x_2)$
CNF12b : $(x_1 + x_2 + x_6)(\bar{x_1} + \bar{x_2} + x_7)(x_9 + x_{10} + x_{12})$
$(\bar{x_6} + \bar{x_7} + x_8)(x_9 + \bar{x_{10}} + \bar{x_{12}})(\bar{x_8} + \bar{x_9} + \bar{x_7})$
MAJ9a : $\{X \mid (\sum_{i=1}^{9} x_i) \geq 3\}$
MAJ9b : $\{X \mid (\sum_{i=1}^{9} x_i) \geq 6\}$
MAJ12a : $\{X \mid (\sum_{i=1}^{12} x_i) \geq 4\}$
MAJ12b : $\{X \mid (\sum_{i=1}^{12} x_i) \geq 8\}$
PAR9a : $\{X \mid ((\sum_{i=1}^{3} x_i) \bmod 2) > 0\}$
PAR9b : $\{X \mid ((\sum_{i=1}^{6} x_i) \bmod 2) > 0\}$
PAR12a : $\{X \mid ((\sum_{i=1}^{4} x_i) \bmod 2) > 0\}$
PAR12b : $\{X \mid ((\sum_{i=1}^{8} x_i) \bmod 2) > 0\}$
MUX9 : $\{X \mid x_{(\text{address}(x_1,x_2)+3)} = 1\}$
MUX12 : $\{X \mid x_{(\text{address}(x_1,x_2,x_3)+4)} = 1\}$

Bibliography

Aha, D. M., Kibler, D., & Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, *6*(1), 37–66.

Anderson, J. R., & Matessa, M. (1992). Explorations of an incremental, bayesian algorithm for categorization. *Machine Learning*, *9*, 275–308.

Andrews, D., & Herzberg, A. (1985). *Data: A Collection of Problems from Many Fields for the student and Research Worker*. Springer-Verlag.

Baffes, P. (1992). Growing layers of perceptrons: introducing the extentron algorithm. In *International Joint Conference on Neural Networks*.

Bailey, T., & Elkan, C. (1993). Estimating the accuracy of learned concepts. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 895–900.

Bradshaw, G. (1993a). Why did the wright brothers get there first? (part 1). *Chemtech*, *23*(6), 8–13.

Bradshaw, G. (1993b). Why did the wright brothers get there first? (part 2). *Chemtech*, *23*(7), 16–22.

Brazdil, P., Gama, J., & Henery, R. (1994). Characterizing the applicability of classification algorithms using meta level learning. In *European Conference on Machine Learning ECML-94*, pp. 83–102.

Breiman, L. (1996a). Bagging predictors. *Machine Learning*, *24*, 123–140.

Breiman, L. (1996b). Stacked regressions. *Machine Learning*, *24*, 49–64.

Breiman, L. (1996c). Bias, variance, and arcing classifiers. Tech. rep. TR 460, University of California, Berkeley, CA. 94720.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth, Belmont, CA.

Breiman, L., & Spector, P. (1992). Submodel selection and evaluation in regression. the x-random case. *International Statistical Review*, *60*, 291–319.

Brodley, C. E., & Utgoff, P. E. (1995). Multivariate decision trees. *Machine Learning*, *19:1*, 45–78.

Buntine, W. (1991). Learning classification trees. In *Artificial Intelligence Frontiers in Statistics*, pp. 182–201. D.J. Hand, Chapman and Hall, London.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, *3*(4), 261–284.

Clarke, M. R. B. (1977). A Quantitative Study of King and Pawn Against King. In Clarke, M. R. B. (Ed.), *Advances in Computer Chess*, Vol. 1, pp. 108–118. Edinburgh University Press.

DeJong, G., & Mooney, R. (1986). Explanation-based learning: An alternate view. *Machine Learning*, *1*(2), 145–176.

Dietterich, T. G., Hild, H., & Bakiri, G. (1990). A comparative study of id3 and back-propagation for english text-to-speech mapping. In *Proceedings of the Seventh International Conference on Machine Learning*, pp. 24–31.

Efron, B. (1983). Estimating the error rate of a prediction rule: Improvement on cross-validation. *Journal of the American Statistical Association*, *78*, 316–331.

Efron, B., & Tibshirani, R. (1993). *An Introduction to the Boostrap*. Chapman & Hall.

Fahlman, S. E., & Lebiere, C. (1990). The cascade-correlation learning architecture. In Touretzky, D. S. (Ed.), *Advances in Neural Information Processing Systems*. Morgan Kaufmann Publishers, Inc. (also techreport CMU-CS-90-100, Carnegie Mellon University).

Fayyad, U. (1994). Branching on attribute values in decision tree generation. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 154–169.

Fisher, D., & McKusick, K. (1989). An empirical comparison of id3 and back-propagation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 788–793.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, *2*(2), 139–172.

Freund, Y., & Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and am application to boosting. In *Computational Learning Theory. Second European Conference, EUROCOLT'95*, pp. 23–37. Springer-Verlag.

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 148–156. San Francisco: Morgan Kaufmann.

Fu, L.-M., & Buchanan, B. G. (1988). Learning intermediate concepts in constructing a hierarchical knowledge base. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 659–666.

Gama, J., & Brazdil, P. (1995). Characterization of classification algorithms. In *7th Portuguese Conference on Artifical Intelligence, EPIA*, pp. 189–200 Funchal, Madeira Island, Portugal.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, *4*, 1–58.

Hadzikadic, M., & Y., Y. D. Y. (1989). Concept formation by incremental conceptual clustering. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 831–836.

Hirose, Y., Yamashita, K., & Hijiya, S. (1991). Back-propagation algorithm which varies the number of hidden units. *Neural Networks*, *4*, 61–66.

Ioerger, T. R., Rendell, L. A., & Subramaniam, S. (1995). Searching for representations to improve protein sequence fold-class prediction. *Machine Learning*, *21* (1–2), 151–175.

Kohavi, R. (1995). A study of cross validation and boostrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1137–1143. Morgan Kaufmann.

Kohavi, R. (1996). Bias plus variance decomposition for zero-one loss functions. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. ?–? Morgan Kaufmann.

Kohavi, R. (1994). Bottom-up induction of oblivious read-once decision graphs. In *Proceedings of the European Conference on Learning*, pp. 154–169.

Kwok, S. W., & Carter, C. (1990). Multiple decision trees. In *Uncertainty in Artificial Intelligence*, Vol. 4, pp. 327–335. Elsevier Science Publishers.

Langley, P. (1996). *Elements of Machine Learning*. Morgan Kaufmann Publishers, Inc., San Francisco, CA.

Langley, P., & Simon, H. A. (1995). Applications of machine learning and rule induction. *Communications of the ACM*, *38*, 54–64.

Lapedes, S. A., Steeg, E. W., & Farber, R. (1995). Use of adaptive networks to define highly predictable protein secondary-structure classes. *Machine Learning*, *21* (1–2), 103–124.

Li, M., & Vitányi, P. M. B. (1992). Philosophical issues in kolmogorov complexity. In in Computer Science, L. N. (Ed.), *International Colloquium on Automata, Languages, and Programming*, Vol. 623. Springer Verlag, Berlin.

Li, M., & Vitányi, P. M. B. (1993). *An Introduction to Kolmogorov Complexity and its Applications*. Springer Verlag, New York.

Matheus, C. J. (1989). *Feature Construction: An Analytical Framework and an Application to Decision Trees*. Ph.D. thesis, University of Illinois at Urbana-Champaign.

Matheus, C. J. (1990). The need for constructive induction. In *Proceedings of the Seventh International Conference on Machine Learning*, pp. 173–177.

Michalski, R. S. (1983). A theory and methodology of inductive learning. In *Machine Learning: An Artificial Intelligence Approach*, chap. 1, pp. 83–134. Tioga Publishing Co., Palo Alto, CA.

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

Michalski, R. S., & Stepp, R. E. (1983). Learning from observation: Conceptual clustering. In *Machine Learning: An Artificial Intelligence Approach*, chap. 11, pp. 331–363. Tioga Publishing Co., Palo Alto, CA.

Michie, D. (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood.

Mitchell, T. (1980). The need for biases in learning generalizations. Tech. rep. CBM-TR-117, Computer Science Department, Rutgers University, New Brunswick, NJ 08903.

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

Murphy, O. J., & Pazzani, M. (1991). Id2-of-3: Constructive induction of m-of-n concepts for discriminators in decision trees. In *Proceedings of the Eighth International Workshop on Machine Learning*, pp. 183–187. San Francisco: Morgan Kaufmann.

Oliveira, A. L. (1995). Inferring reduced ordered decision graphs of minimum description length. In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 421–429. Morgan Kaufmann Publishers, Inc.

Pagallo, G., & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, *5*, 71–99.

Pérez, E., & Rendell, L. A. (1996). Learning despite concept variation by finding structure in attribute-based data. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 391–399.

Pérez, E., Vilalta, R., & Rendell, L. A. (1996). On the importance of change of representation in induction. In *What is Inductive Learning?: A Worshop of the Canadian Conference on Artificial Intelligence*.

Pérez, E. (1997). *Learning Despite Complex Attribute Interaction: An Approach Based on Relational Operators*. Ph.D. thesis, University of Illinois at Urbana-Champaign.

Qian, N., & Sejnowski, T. (1988). Predicting the secondary structure of globular proteins using neural network models. *Journal of Molecual Biology*, *202*, 865–884.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In Michalski, R., Carbonell, J., & Mitchell, T. (Eds.), *Machine Learning: An Artificial Intelligence Approach*, chap. 15, pp. 463–482. Tioga Publishing Co., Palo Alto, CA.

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

Quinlan, J. R. (1987). Generating production rules from decision trees. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pp. 304–307.

Quinlan, J. R. (1993). Comparing connectionist and symbolic learning methods. In Hanson, S., Drastal, G., & Rivest, R. (Eds.), *Computational Learning Theory and Natural Learning Systems: Constraints and Prospects*, Vol. 1, pp. ?–? MIT Press, Cambridge, MA.

Quinlan, J. R. (1994). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc., Palo Alto, CA.

Quinlan, R. (1995). Oversearching and layered search in empirical learning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1019–1024. Morgan Kaufmann.

Quinlan, R. (1996). Bagging, boosting, and c4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 725–730. MIT Press.

Ragavan, H., & Rendell, L. A. (1991). Relieving limitations of empirical algorithms. In *Proceedings of the Change of Representation Workshop at the Twelfth International Joint Conference on Artificial Intelligence*.

Ragavan, H., & Rendell, L. A. (1993). Lookahead feature construction for learning hard concepts. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 252–259. San Francisco: Morgan Kaufmann.

Rao, R. B., Gordon, D., & Spears, W. (1995). For every generalization action, is there really an equal and opposite reaction? analysis of the conservation law for generalization performance. In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 471–479.

Rendell, L. (1986). A general framework for induction and a study of selective induction. *Machine Learning*, *1*(2), 177–226.

Rendell, L. (1988). Learning hard concepts. In *Proceedings of the Third European Working Session on Learning*, pp. 177–200.

Rendell, L., & Ragavan, H. (1993). Improving the design of induction methods by analyzing algorithm functionality and data-based concept complexity. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 952–958.

Rendell, L. A. (1985). Substantial constructive induction using layered information compression: Tractable feature formation in search. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 650–658.

Rendell, L. A., & Cho, H. (1990). Empirical learning as a function of concept character. *Machine Learning*, *5*(3), 267–298.

Rendell, L. A., & Seshu, R. (1990). Learning hard concepts through constructive induction: Framework and rationale. *Computational Intelligence*, *6*, 247–270.

Rivest, L. (1987). Learning decision lists. *Machine Learning*, *2*(3), 229–246.

Romaniuk, S. G. (1996). Learning to learn with evolutionary growth perceptrons. In Pal, S. K., & Wang, P. P. (Eds.), *Genetic Algorithms for Pattern Recognition*, chap. 9, pp. 179–211. CRC Press, Boca Ratón, Florida.

Romaniuk, S. (1994). Learning to learn: Automatic adaptation of learning bias. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 871–876. MIT Press.

Rost, B., & Sanders, C. (1993). Prediction of protein secondary structure at better than 70% accuracy. *Journal of Molecual Biology*, *232*, 584–599.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Vol. 1, pp. 318–362. MIT Press.

Rymon, R. (1993). An SE-tree based characterization of the induction problem. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 268–275. San Francisco: Morgan Kaufmann.

Schaffer, C. (1993). Selecting a classification method by cross-validation. *Machine Learning*, *13*, 135–143.

Schaffer, C. (1994). A conservation law for generalization performance. In *Proceedings of the Eleventh International Conference on Machine Learning*, pp. 259–265. San Francisco: Morgan Kaufmann.

Schuurmans, D., Ungar, L., & Foster, D. (1997). Characterizing the generalization performance of model selection strategies. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 340–348.

Shao, J. (1993). Linear model selection by cross-validation. *Journal of the American Statistical Association*, *88*, 486–494.

Shavlik, J. W., Hunter, L., & Searls, D. (1995). Introduction to the special issue on applications in molecular biology. *Machine Learning*, *21*(1–2), 5–9.

Thornton, C. (1996). Parity: The problem that won't go away. In *Proceedings of the 11th Biennial Conference of the Canadian Society for Computational Studies of Intelligence, Toronto Ontario*, pp. 362–374.

Towell, G., & Shavlik, J. (1991). Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules. *Advances in Neural Information Processing Systems*, *4*, 977–984.

Towell, G., & Shavlik, J. (1993). Extracting refined rules from knowledge-based neural networks. *Machine Learning*, *13*, 71–101.

Utgoff, P. E. (1986). Shift of bias for inductive concept learning. In Michalski, R., Carbonell, J., & Mitchell, T. (Eds.), *Machine Learning: An Artificial Intelligence Approach, Vol. II*, chap. 5, pp. 107–148. Morgan Kaufmann Publishers, Inc., Los Altos, CA.

Utgoff, P. E. (1988). Perceptron trees: A case study in hybrid concept representations. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 601–606.

Vilalta, R., Blix, G., & Rendell, L. A. (1997). Global data analysis and the fragmentation problem in decision tree induction. In *9th European Conference on Machine Learning*, pp. 312–326. Lecture Notes in Artificial Intelligence, Vol. XXX, Springer-Verlag, Heidelberg.

Vilalta, R., & Rendell, L. A. (1997). Integrating feature construction with multiple classifiers in decision tree induction. In *14th International Conference on Machine Learning*, pp. 394–402. Morgan Kaufman.

Vilalta, R. (1994). A general schema for lookahead feature construction in inductive learning. Master's thesis, University of Illinois at Urbana-Champaign.

Vitányi, P., & Li, M. (1996). Ideal MDL and its relation to Bayesianism. In *ISIS: Information, Statistics Induction in Science*, pp. 282–291. World Scientifoc, Singapore.

Vitányi, P., & Li, M. (1997). On prediction by data compression. In *9th European Conference on Machine Learning*. Lecture Notes in Artificial Intelligence, Vol. XXX, Springer-Verlag, Heidelberg.

Watanabe, S. (1969). *Knowing and Guessing*. John Wiley & Sons, New York.

Watanabe, S. (1985). *Pattern Recognition: Human and Mechanical*. John Wiley & Sons, New York.

Webb, G. I. (1995). Opus: An efficient admissible algorithm for unordered search. *Journal of Artificial Intelligence Research*, *3*, 431–435.

Weiss, S. M., & Kapouleas, I. (1989). An empirical comparison of pattern recognition of neural nets and machine learning classification methods. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 781–787.

Weiss, S. M., & Kulikowski, C. A. (1990). *Computer Systems That Learn*. Morgan Kaufmann Publishers, Inc., San Mateo, CA.

Weiss, S., & Indurkhya, N. (1993). Optimized rule induction. *IEEE Expert*, *8*(6), 61–69.

Weiss, S., & Indurkhya, N. (1998). *Predictive Data Mining*. Morgan Kaufmann Publishers, Inc., San Francisco, CA.

Wnek, J., & Michalski, R. S. (1994). Comparing symbolic and subsymbolic learning. In *Machine Learning: An Artificial Intelligence Approach, Vol. IV*, chap. 19, pp. 489–519. Morgan Kaufmann Publishers, Inc., San Francisco, CA.

Wogulis, J., & Langley, P. (1989). Improving efficiency by learning intermediate concepts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 657–662.

Wolberg, W. H., & Mangasarian, O. L. (1990). Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the Naional Academy of Sciences*, *87*, 9193–9196.

Wolpert, D. (1992). Stacked generalization. *Neural Networks*, *5*, 241–259.

Wolpert, D. (1996). The lack of a priori distinctions between learning algorithms and the existence of a priori distinctions between learning algorithms. *Neural Computation*, *8*, 1341–142.

Zhang, P. (1992). On the distributional properties of model selection criteria. *Journal of the American Statistical Association*, *87*, 732–737.

Curriculum Vitae

Education

- PhD. in Computer Science.
  University of Illinois at Urbana-Champaign.
  Doctoral Committee: Larry Rendell, Gerald DeJong, Caroline Hayes, and Gary Bradshaw.
  GPA 4.0/4.0.

- M.S. in Computer Science.
  University of Illinois at Urbana-Champaign 1995.
  GPA 4.0/4.0.

- B.S. in Computer Science.
  Instituto Tecnológico y de Estudios Superiores de Monterrey, México, 1989.
  GPA 99.8/100.0.
  All-departments best student award.

Research Interest

Machine Learning, Pattern Recognition, Neural Nets, Data Mining, Artificial Intelligence.

Publications

- Vilalta R., Blix G., and Rendell L. (1997) "Global Data Analysis and the Fragmentation Problem in Decision Tree Induction". In the 9th European Conference on Machine Learning, Lecture Notes in Artificial Intelligence, Vol. XXX, Springer-Verlag, Heidelberg Springer-Verlag.

- Vilalta R. and Rendell L. (1997) "Integrating Feature Construction with Multiple Classifiers in Decision Tree Induction". In the 14th International Conference on Machine Learning.

- Perez E., Vilalta R., and Rendell L. "On the Importance of Change of Representation in Induction" (Invited talk). Presented at the Workshop of Inductive Learning for the 1996 Canadian Conference on Artificial Intelligence.
- Vilalta R. (1995), "A General Schema for Lookahead Feature Construction in Inductive Learning". University of Illinois at Urbana-Champaign. Master's Thesis.
- Vilalta R., Blix G., and Rendell L. (1995) "The Value of Lookahead Feature Construction in Decision Tree Induction". Beckman Institute, University of Illinois at Urbana-Champaign, Technical Report UIUC-BI-AI-95-01.

Affiliations & Reviewer

- Member of the American Association for Artificial Intelligence (1995-97).
- Reviewer for the National Conference on Artificial Intelligence (1995-96).
- Reviewer for the International Conference on Machine Learning (1996).

Programming Skills

- Languages: C, C++, Java, Lisp, Scheme, Prolog, Pascal, Fortran, several assemblers.
- Operating Systems, Window Managers: UNIX, DOS, OS2, X-Windows.

Professional Experience

- IBM, System's Engineer, 1990-1991, México.
  - Development and maintenance of computer systems for handling large databases in financial operations. The system was made operational on IBM mainframes, and used by major banks and financial institutions.
  - Installation and maintenance of "ImagePlus": an image processing system for the digital manipulation of electronic documents.

  - Building of a rule-based expert system to assist in decision-making for financial institutions.

- Research Assistantship at the Dept. of Agriculture, University of Illinois at Urbana-Champaign, 1996-1997. Development of a software tool for the application of machine leaning techniques in agricultural databases.

- Development of a system for handling student records at the Instituto Tecnológico y de Estudios Superiores de Monterrey, México, 1989. The project involved communication between several computers to access on-line information regarding student information.

Honors and Awards

- Distinction as the student graduating with best GPA among students from all departments, Instituto Tecnológico y de Estudios Superiores de Monterrey, México, December, 1989; a bronze plate was placed on the library's award wall.

- Honored as best student during the IBM training program in computer systems, México, 1990.

- Awarded a 5-year Fulbright scholarship, together with a grant from the government of México, to pursue graduate studies in the United States, 1991.