TIME SERIES LEARNING WITH PROBABILISTIC NETWORK COMPOSITES

BY

WILLIAM HENRY HSU

B.S., The Johns Hopkins University, 1993
M.S.E., The Johns Hopkins University, 1993

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

TIME SERIES LEARNING WITH PROBABILISTIC NETWORK COMPOSITES

William Henry Hsu, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1998
Sylvian R. Ray, Advisor

The purpose of this research is to extend the theory of uncertain reasoning over time through integrated, multi-strategy learning. Its focus is on **decomposable**, concept learning problems for classification of spatiotemporal sequences. Systematic methods of task decomposition using attribute-driven methods, especially attribute **partitioning**, are investigated. This leads to a novel and important type of unsupervised learning in which the feature construction (or extraction) step is modified to account for multiple sources of data and to systematically search for embedded temporal patterns. This modified technique is combined with traditional cluster definition methods to provide an effective mechanism for decomposition of time series learning problems. The decomposition process interacts with model selection from a collection of probabilistic models such as temporal artificial neural networks and temporal Bayesian networks. Models are chosen using a new quantitative (metric-based) approach that estimates expected performance of a learning architecture, algorithm, and mixture model on a newly defined subproblem. By mapping subproblems to customized configurations of probabilistic networks for time series learning, a hierarchical, supervised learning system with enhanced generalization quality can be automatically built. The system can improve data fusion capability (overall localization accuracy and precision), classification accuracy, and network complexity on a variety of decomposable time series learning problems. Experimental evaluation indicates potential advances in large-scale, applied time series analysis (especially prediction and monitoring of complex processes). The research reported in this dissertation contributes to the theoretical understanding of so-called **wrapper** systems for high-level parameter adjustment in inductive learning.

History is Philosophy teaching by examples.

Thucydides (c. 460-c. 400 B.C.), Athenian historian.

Quoted by Dionysius of Halicarnassus in: *Ars Rhetorica*, Chapter 11, Section 2.

Acknowledgements

First and foremost, my utmost gratitude goes to my advisor, Sylvian R. Ray. Professor Ray is one of those rare leaders who has shepherded not one but several research groups of the highest caliber during his years in academia. To associate with him has truly been an honor and a privilege. He is a true scholar, a generous and conscientious mentor, and a gentleman in every sense of the word. Where our research interests differ, he has always lent an ear and an open mind, and where they are similar, he has given tirelessly of his time, formidable experience, and piercing insight. To emulate him is my lifelong aspiration.

I thank the members of my committee: David E. Goldberg, Mehdi T. Harandi, and David C. Wilkins. Thanks to Professor Goldberg for an introduction to genetic algorithms and global optimization, but also for teaching by example how to be a better engineer. The educational clarity and the irrepressible drive for which he is known, and a few questions he asked at important junctures, have been a great help to me. Professor Harandi introduced me to a number of useful concepts in knowledge-based programming and software engineering, but equally important, gave me an education in responsible research. He sets a high standard in research and encourages others to follow, and I am grateful for the chance to participate in many interesting and substantial discussions with him and his group. I also appreciate his good advice on some important efforts, including, but not limited to, my dissertation. Finally, thanks to Professor Wilkins for supporting me as a research assistant and for the opportunity to work in his Knowledge-Based Systems Laboratory throughout most of my Ph.D. studies; it is a unique and diverse group to which I am glad to have contributed. Interacting with the many KBS members has been an interesting experience, and has led to a number of productive collaborations in knowledge-based systems, machine learning, and applied research.

Thanks to my professors, classmates, and friends from my undergraduate school, the Johns Hopkins University, for inspiring my appetite for research. I am especially grateful for the tutelage of Amy Zwarico and Simon Kasif, who gave me an early introduction to software engineering and intelligent systems.

Special thanks to Jesse Reichler and Chris Seguin, members of Professor Ray's research group, who have patiently listened to my research ideas and presentations (and rehearsals) on numerous occasions. Equally important, they taught me about areas that they knew better, and

held lively and rewarding discussions with me and with others during our years at UIUC. I look forward to working and associating with both of them for many years to come.

My thanks to the following researchers at UIUC for valuable discussions about my thesis research and for their candid and helpful feedback: Brendan Frey, Thomas Huang, Larry Rendell, Dan Roth, and Benjamin Wah. Thanks also to the following researchers at other universities and companies, who have given me feedback and advice during my years as a graduate student: Robert Hecht-Nielsen, Rob Holte, Kai-Fu Lee, and Mehran Sahami. Thanks especially to Dr. Hecht-Nielsen, who gave me advice on selecting a thesis topic at the 1996 World Congress on Neural Networks. Also, thanks to Mehran Sahami for introducing me to related work on model selection.

My appreciation and gratitude to experts in the areas of applied climatology, agricultural engineering, agricultural economics, crop sciences, and computational methods for precision agriculture who consulted me about experimental data. These include: Don Bullock, Mike Clark, Tom Frank, Steven Hollinger, Don Holt, Doug Johnston, Ken Kunkel, John Reid, Bob Scott, and Don Wilhite.

Thanks also to the students, research staff, and alumni of the UIUC Department of Computer Science and the Beckman Institute, especially Eugene Grois, Ole Jakob Mengshoel, and Ricardo Vilalta. I'd like to give special acknowledgement to the undergraduates at the KBS Lab who worked on research projects with me during my time there, especially: Nathan Gettings Yu Pan, and Victoria Lease. Nathan, Yu Pan, and Tori assisted with some experiments and implementations relevant to my dissertation, and were also an important wellspring of culture for me during the more grueling months of my research. Additionally, my thesis could not have been completed without the friendly, helpful, courteous, and professional administrative staff at the Beckman Institute, Aviation Research Lab, and Department of Computer Science. None of us can do it without you!

Last but certainly not least, I thank my parents for all their encouragement and love.

I have thanked a great many people, yet I know I have left some out. Mei-Yuh Hwang quoted a proverb in the acknowledgements section of her thesis that advises: when one is blessed with too many people to thank, one should thank God. I do.

1. Introduction

The purpose of this research is to improve existing methods for inductive concept learning from time series. A time series is, colloquially, any data set whose points are indexed by time and organized in nondecreasing order.[1] Time series learning refers to a variety of learning problems, including prediction of the next point in a sequence and *concept learning* where each data vector, or point, is an exemplar and the task is to classify the next ("test") exemplar given previous exemplars as training data. In traditional concept learning formulations, the order of presentation of exemplars is relevant only to the learning algorithm (if at all), not to the classifier (rule or other decision structure) that is produced. In time series classification (concept learning), however, it is generally relevant to both. Thus, the definition of concept learning is extended to time series by taking into account all previously observed data. Furthermore, class membership (i.e., the learning target) may be binary, general discrete-valued (or nominal), or continuous-valued. This dissertation therefore focuses on *discrete classification* over discrete time series.

This chapter describes the *wrapper* approach to inductive learning and how it has previously been used to enhance the performance (classification accuracy) of supervised learning systems. In this dissertation, I show how wrappers for *attribute subset selection* can also be incorporated into unsupervised learning – specifically, constructive induction – for redefinition of learning problems. This approach is also referred to as *change of representation* and *optimization of inductive bias*. I adapt the constructive induction framework to decomposition of learning tasks by substituting attribute *partitioning* for attribute subset selection. This leads to definition of multiple subproblems instead of a single reformulated problem. This affords the opportunity to apply multi-strategy learning; for time series, the choice of learning technique is based on the type of temporal, stochastic patterns embedded in the data. I develop a metric-based technique for identifying the closest type of pattern from among known, characteristic types. This allows each subproblem to be mapped to the most appropriate model (i.e., learning architecture), and also allows a (hierarchical) mixture model and training algorithm to be automatically selected for the entire decomposed problem. The benefit to supervised learning is reduced variance through multiple models (which I will refer to as *composites*) and reduced model complexity through problem decomposition and change of representation.

[1] More rigorously, we may require that the time index be nonnegative and that certain conventions be consistent for a training set and its continuation. Typical choices, regarding the representation of time series specifically, include discrete versus continuous time, synchronous versus asynchronous data vectors and variables (within each data vector), etc. [BJR94, Ch96].

1.1 Spatiotemporal Sequence Learning with Probabilistic Networks

A *spatiotemporal* sequence is a data set whose points are ordered by location and time. Spatiotemporal sequences arise in analytical applications such as time series prediction and monitoring [GW94, Ne96], sensor integration [SM93, Se98], and multimodal human-computer intelligent interaction. Learning to classify time series is an important capability of intelligent systems for such applications. Many problems and types of knowledge in intelligent reasoning with time series, such as diagnostic monitoring, prediction (or forecasting), and control automation can be represented as classification.

This section presents existing methods for concept learning from time series. These include local optimization methods such as delta rule learning (or *backpropagation* of error) [MR86, Ha94] and expectation-maximization (EM) [DLR77], as well as global optimization methods such as Markov chain Monte Carlo estimation [Ne96]. I begin by outlining the general framework of time series learning using probabilistic networks. I then discuss how certain time series learning problems can be processed using attribute-driven methods to obtain more tractable subproblems, to boost classification accuracy, and to facilitate multi-strategy supervised learing. This leads to a system design that integrates unsupervised learning and model selection to map each subproblem to the most appropriate configuration of probabilistic network. In designing a systematic decomposition and metric-based model selection system, I address a number of shortcomings of existing time series learning methods with respect to *heterogeneous* time series. In Section 1.1.4, I give a precise definition of heterogeneous time series and give examples of real-world analytical problems where they arise. Finally, I discuss the role of hierarchical *mixture models* in integrated, multi-strategy learning systems, especially their benefits for time series learning using multiple probabilistic networks.

1.1.1 Statistical and Bayesian Approaches to Time Series Learning

Time series occur in many varieties. Some are periodic; some contain values that are linear combinations of preceding ones; some observe a finite limit on the duration of values (i.e., the number of consecutive data points with the same value for a particular variable); and some observe attenuated growth and decay of values. These *embedded pattern types* describe the way that values of a time series evolve as a function of time, and are sometimes referred to as *memory forms* [Mo94]. A memory form can be characterized in terms of a hypothetical *process* [TK84] that generates patterns within the observed data (hence the term *embedded*). A memory form can be represented using various models. Examples include: generalized linear models in the case of

periodicity [MN83]; moving average models in the case of linear combinations [Mo94, MMR97], finite state models and grammars in the case of finite-duration patterns [Le89, Ra90], and exponential trace models in the case of attenuated growth and decay [Mo94, RK96, MMR97].

All of the above memory forms can exhibit noise, or uncertainty. The noisy pattern generator can be characterized as a stochastic process. In certain cases, the probability distributions that describe this process have specific structure. This allows information abut the stochastic component of a time series to be encoded as model parameters. Examples include graphical state transition models with distribution over transitions (a probabilistic type of *Moore model* or *Mealy model*, also known as *Reber grammars* [RK96]), or similar state models with distributions over transitions and outputs (also known as *hidden Markov models or HMMs* [Ra90]).

This dissertation focuses on graphical models of probability, specifically, *probabilistic networks*, or *connectionist networks*, as the models (hypothesis languages) used in inductive concept learning. These include simple recurrent networks [El90, Ha94, Mo94, Ha95, PL98], HMMs [Ra90, Le89], and temporal Bayesian networks [Pe88, He96]. Network architectures are further discussed in Section 1.2, Chapter 2, and Appendix B.1. The structure of a stochastic process can be learned using local and global optimization methods that fit the model parameters. For example, gradient learning can be applied to fit generalized linear models and multilayer perceptrons (also called feedforward artificial neural networks) [MR86], as well as other probabilistic networks, such as Bayesian networks and HMMs [BM94, RN95]. *Expectation-Maximization* (*EM*) [DLR77, BM94] is another local optimization algorithm that can be used to estimate parameters in graphical models; it has the added capability of being able to estimate missing data. Finally, Bayesian methods for global optimization include the *Markov chain Monte Carlo* family [Ne93, Gi96], which performs integration by random sampling from the conditional distribution of models given observed data [KGV83, AHS85, Ne92]. Appendix B.2 gives in-depth details of the time series learning algorithms applied in this dissertation.

1.1.2 Hierarchical Decomposition of Learning Problems

A key research issue addressed in this dissertation is *change of representation* for time series learning. Even more than in general inductive learning, change of representation is ubiquitous in analysis of spatiotemporal sequences. It occurs due to signal processing, multimodal integration of sensors and data sources, differences in temporal and spatial scales, geographic projections and subdivision, and operations for dealing with missing data over space and time (interpolation, downsampling, and Bayesian estimation). I investigate a particularly important form of change

of representation for real-world time series: *partitioning* of input. In Chapter 2, I will describe *attribute-driven* methods (subset selection and partitioning) for problem reformulation, and explain how these methods correspond to the *feature construction and extraction* phase of constructive induction [Ma89, RS90, Gu91, Do96]. Partitioning the input of a time series learning problem into subsets of attributes is the first step of a problem decomposition process that enables numerous opportunities for improved supervised learning. The benefits are discussed throughout Chapters 2, 3, and 4 and empirically demonstrated in Chapter 5. In brief, decomposing a learning problem by attribute partitioning results in the formation of a hierarchy of problem definitions that facilitates model selection and data fusion.

1.1.3 Constructive Induction and Model Selection: State of the Field

The decomposition process interacts with model selection from a collection of probabilistic models such as temporal artificial neural networks and temporal Bayesian networks. Traditionally, constructive induction has been directed toward such concerns as *hypothesis preference* [Mi83, Ma89, RS90, Do96, Io96, Pe97, Vi98], i.e., the formulation of new descriptors for concept classes that permit more tractable and accurate supervised learning. New descriptors are formed based upon the initial problem specification (the *ground attributes*, or *instance space* [RS90, Mi97]), the empirical characteristics of the training data, and prior knowledge about the test data (the *desired inference space* [DB98]). Similarly, decomposition of learning problems has dealt with focusing different induction algorithms (or components of a *mixture model* [RCK89, JJB91, JJNH91, JJ93, JJ94]) on different parts of the hypothesis space, to more easily describe the concept classes. The difference between most constructive induction and decomposition algorithms is that the former produces a single reformulated learning problem, while the latter produces several. In Chapter 2, I show how attribute partitioning meets objectives of both constructive induction and problem decomposition.

Constructive induction can be divided into two phases: reformulation of input and internal representations (*feature construction* [Do96] and *feature extraction* [KJ97]) and reformulation of the hypothesis language, or target concept (*cluster definition* [Do96]). Feature construction and extraction apply operators to synthesize new (*compound*) attributes from the original (*ground*) attributes in the input specification. By contrast, the method of *attribute subset selection* [Ki92, Ca93, Ko94, Ko95, KJ97] identifies those inputs upon which to focus an induction algorithm's attention. It does not, however, inherently perform any synthesis of new hypothesis descriptors. Subset selection is tied to the problem of *automatic relevance determination* (*ARD*), which

estimates the capability of an attribute to distinguish the output class in the context of other attributes [He91, Ne96]. In Chapter 2, I explain how attribute subset selection and partitioning can augment, or be substituted, for feature construction in a constructive induction system. The function of this modified system depends on whether subset selection or partitioning is used; in this dissertation, I focus on partitioning, whose purpose is to produce multiple subproblem definitions. An evaluation function is required to ensure that these definitions constitute a good *decomposition* of a time series learning problem.

One of the main novel contributions of this dissertation is an elucidation of the relationship among constructive induction (by attribute partitioning), mixture models, and *model selection*. Model selection is the problem of identifying a hypothesis language that is appropriate to the characteristics of a training data set [GBD92, Hj94, Sc97]. Chapter 3 focuses on how model selection can be improved, given a good decomposition of a task. Each model in my learning system is associated with a characteristic pattern (memory form) and identifiable types of prior ad conditional probability distributions. This association allows the most appropriate learning architecture, mixture model, and training algorithm to be applied for each subset of training data generated by constructive induction. The type of model selection I apply is *coarse-grained* (situated at the level of learning architecture – i.e., the *type of probabilistic network* to use) and *quantitative* (metric-based – i.e., based upon a measure of expected performance). Equally important, it is customized for multi-strategy learning where every choice of "strategy" is a probabilistic network for time series learning. This common trait simplifies the model selection framework and makes the system more uniform, but does not restrict its applicability in practice.

1.1.4 Heterogeneous Time Series, Decomposable Problems, and Data Fusion

By mapping subproblems to customized configurations of probabilistic networks for time series learning, a hierarchical, supervised learning system with enhanced generalization quality can be automatically built. This dissertation addresses data fusion [SM93] using different types of hierarchical mixture models. Data fusion is of particular importance to learning from *heterogeneous* time series, which I define here, by way of an analogy.

A *heterogeneous file* is any file containing multiple types of data [HZ95]. In operating systems applications (data compression, information retrieval, Internet communications), this is well defined: audio, text, graphics, video (or, more specifically, formats thereof) are file types. A *heterogeneous data set* is any data set containing multiple types of data. Because "types of data"

is a largely unrestricted description, this definition is much more nebulous than that for files – that is, until the learning environment (sources of data, preprocessing element, knowledge base, and performance element) is defined [CF82, Mi97]. Section 1.5 and Chapter 2 present this definition.

A *heterogeneous time series* is a data set containing multiple types of temporal data. There are several ways to decompose temporal data: by the location of the source (spatiotemporal maps); by granularity (i.e., frequency) of the sample; or by prior information about the source (e.g., an organizational specification for multiple sensors). This dissertation considers each of these, but focuses on the third aspect of decomposition. The goal of decomposition is to find a partitioning of the training data that results in the highest prediction accuracy on test data. To begin formalizing this notion, I begin by defining *decomposability*, in terms of its criteria as addressed in this research:

Definition. Given an attribute-based mechanism for partitioning of time series data sets, an assortment of learning models, a quantitative model selection mechanism, and a data fusion mechanism, a particular time series learning problem is *decomposable* if it admits separation into subproblems of lower complexity based on these mechanisms.

The attribute-based mechanism for partitioning is the topic of Chapter 2. The assortment of learning models (which comprises the learning architecture and the learning method) and the model selection mechanism are both formalized through the definition and explanation of *composite learning* in Chapter 3. The data fusion part of this definition is formalized in Chapter 4. Finally, analysis of overall network complexity is presented in Chapter 5.

1.1.5 System Overview

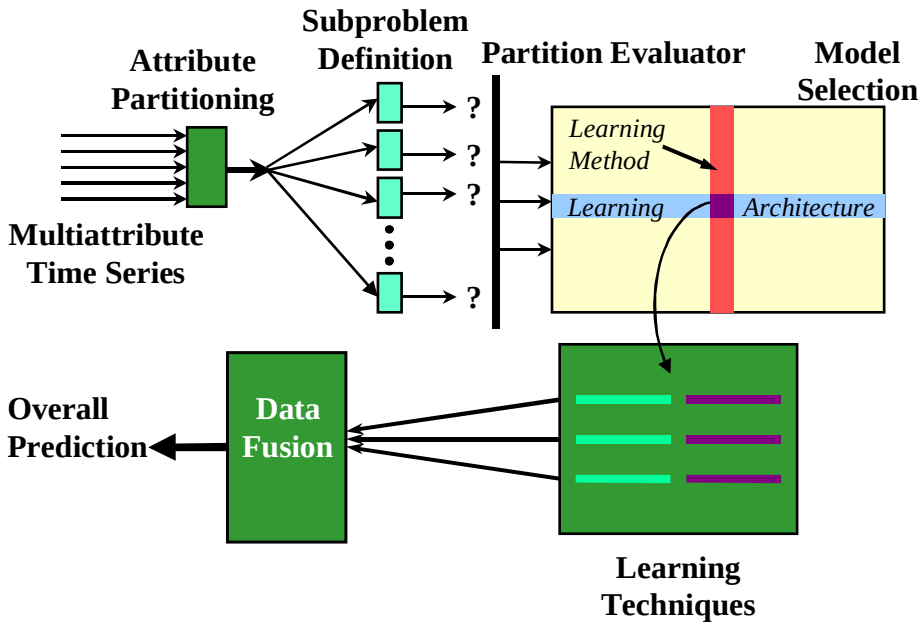

Figure 1. Overview of the integrated, multi-strategy learning system for time series.

Figure 1 depicts a learning system for decomposable, multi-attribute time series. The central elements of this system are:

1. A systematic mechanism for generating and **evaluating** candidate **subproblem definitions** in terms of **attribute partitioning**.[2]
2. A metric-based **model selection** component that maps subproblem definitions to learning techniques.
3. A **data fusion** mechanism for integration of multiple models.

Chapter 3 presents **Select-Net**, a high-level algorithm for building a complete **learning method** specification (*composite*) and training subnetworks as part of a system for multi-strategy learning. This system incorporates attribute partitioning into constructive induction to obtain multiple problem definitions (decomposition of learning tasks); brings together constructive induction and mixture modeling to achieve systematic definition of learning techniques; and integrates both with metric-based model selection to *search for efficient hypothesis preferences*.

[2] As I explain in Chapter 2, this may be a naïve (exhaustive) enumeration mechanism, but is more realistically implemented as a *state space search*.

1.2 Problem Redefinition for Concept Learning from Time Series

This section briefly surveys existing methods for problem reformulation, their shortcomings and assumptions, and potential application to time series learning.

1.2.1 Constructive Induction: Adaptation of Attribute-Based Methods

In probabilistic network learning, constructive induction methods tend to focus on literal *cluster definition* [Do96] rather than a systematized program of feature construction or extraction[3] and cluster definition. Cluster definition techniques are numerous, and include self-organizing maps and competitive clustering (aka vector quantization) [Ha94, Wa85]. The approach I report in Chapter 2 follows a regime of unsupervised inductive learning that is conventional in the practice of symbolic machine learning [Mi83], but has been adapted here for *seminumerical* learning (sometimes referred to as *subsymbolic*). The attribute-driven methods that I incorporate into an unsupervised learning framework perform what Michalski categorizes as both *constructive* and *selective* induction [Mi83].

1.2.2 Change of Representation in Time Series

Many previous theoretical studies have ascertained a need for change of representation in inductive learning [Be90, RS90, RR93, Io96, Mi97]. Systematic search for a beneficial change of representation amounts to a search for inductive bias [Mi80, Be90]. Recent work on constructive induction includes knowledge-guided methods [Do96], relational projections [Pe97], decomposable models [Vi98], explicit search for change of representation to boost supervised learning performance [Io96], and other algorithms for systematic optimization of hypothesis representation [Ha89, WM94, Mi97]. A common theme of this work, and of the expanding body of research on attribute subset selection [Ki92, Ca93, Ko94, Ko95], is that the hypothesis language in a supervised learning problem may be cast as a group of *tunable parameters*. This is the design philosophy behind attribute-based problem decomposition, described in Chapter 2.

1.2.3 Control of Inductive Bias and Relevance Determination

Subset selection is tied to the problem of *automatic relevance determination* (*ARD*), a process that, informally, is designed to assign the proper weight to attributes based upon their importance.

This is measured as the discriminatory capability of an attribute, given other attributes that may be included. Formal Bayesian and syntactic characterizations of relevance can be found in the work of Heckerman [He91], Neal [Ne96], and Kohavi and John [KJ97]. The significance of attribute partitioning to ARD is that partitioning extends the notion that relevance is a joint property of a group of attributes. It applies criteria similar to those used to "shrink" a set of attributes down to a minimal set of relevant ones. These criteria treat each separate subproblem as a candidate subset of attributes, but account for the imminent use of this subset for a newly defined target concept (found through cluster definition) and within a larger context (the mixture model for the entire attribute partition).

1.3 Model Selection for Concept Learning from Time Series

This section presents a synopsis of the model selection concepts that are introduced or investigated in this dissertation, and gives a map to the sections where they are explained and evaluated.

1.3.1 Model Selection in Probabilistic Networks

A central innovation of this dissertation is the development of a system for specifying the learning technique to use for each component of a time series learning problem. While the general methodology of model selection is not new [St77], nor is its use in technique selection for inductive learning [Sc97, EVA98], its application to time series through the explicit characterization of memory forms is a novel contribution of this research. I will refer to the specification of learning techniques for each component of a partitioned concept learning problem as a *composite* (specifically, *probabilistic network composites* for the kind of specifications generated in this particular learning system). I will also refer to the process of training probabilistic networks for each subproblem and for a hierarchical mixture model – according to this specification – as *composite learning*.

Each model in my learning system is associated with a characteristic pattern (memory form) and identifiable type of probability distribution over the training data. The former is a high-level descriptor of the conditional distribution of model parameters *for a particular model configuration (the architecture, connectivity, and size)*, given the observed data. That is, certain entire families of temporal probabilistic networks are good or bad for a particular data set; the

[3] Because this dissertation deals with constructive induction based on *attribute partitioning*, it will not

degree of match between the memory form and this family can be estimated by a metric. This metric is a predictor of performance by members of this family, if one is chosen as the model type for a subset of the data. The latter describes the estimated conditional distribution of mixture model parameters, *for a particular type of mixture*, given the data, as well as estimated priors for a particular model configuration.

1.3.2 Metric-Based Methods

Model selection has been studied in the statistical inference literature for many years [St77, Hj94], but has been addressed systematically in machine learning only recently [GBD92, Hj94]. Even more recent is the advent of metric-based methods [Sc97] for model selection. The purpose of metric-based methods in this research is to counteract the instability of certain configurations of probabilistic networks that make it difficult to conclusively compare the performance of two candidate configurations. Although statistical evaluation and validation systems, such as *DELVE* [RNH+96], have been developed for just this purpose, tracking the performance of a learning system across different samples remains an elusive task [Gr92, Ko95, KSD96]. The problems faced by researchers trying to compare network performance are aggravated when the data comes from a time series and the networks being evaluated belong to a hierarchical mixture model. Even if it were feasible to track performance on continuations of the time series [GW94, Mo94], subject to the dynamics of the learning system [JJ94], it would introduce another level of complication to consider all the different combinations of learning architectures *within* the mixture model. Yet the comparative results on these combination are precisely what is needed to properly evaluate candidate partitions and architectures given an already-selected mixture model and training algorithm. This is the motivation for using metrics for estimating expected performance of a learning technique, instead of the more orthodox method of gathering *descriptive statistics* on network performance using every combination. This design philosophy is further explained in Chapter 3.

1.3.3 Multiple Model Selection: A New Information-Theoretic Approach

Having postulated a rationale for metric-based model selection over multiple subproblems, it remains to formulate a hypothetical criterion for expected performance. In fact, this is one of the important design issues for the research reported in this dissertation. Chapter 3 describes the

---

make distinctions between feature construction and extraction. The interested reader is referred to [Ki86].

organization of my *database of learning techniques* and the metrics for selecting particular *learning architectures* (network types) and *learning methods* (training algorithms and mixture models) from it. The principle that motivated the design of metrics for selecting network types is that *learning performance for a temporal probabilistic network is correlated with the degree to which its corresponding memory form occurs in the data*.

The memory forms that I study in this dissertation include the *autoregressive integrated moving average* (*ARIMA*) family [BJR94, Hj94, Mo94, Ch96, MMR97, PL98], one that includes the *autoregressive moving average* (*ARMA*), *autoregressive* (*AR*), and *moving average* (*MA*) memory forms. These memory forms and their temporal artificial neural network (ANN) realizations [El90, DP92, Mo94, MMR97, PL98] are documented in Chapter 3, where I present a new approach to quantitative model selection that is based upon information theory [CT91]. In short, the metrics are designed to measure the decrease in uncertainty regarding predictions on test cases, or continuations of the time series, after the data set has been transformed according to a particular time series model. This transformation makes available all of the historical information that can be represented by the memory type of the candidate model, and the change in uncertainty is simply measured by the mutual information (i.e., the decrease in entropy due to conditioning on historical values). A similar approach was used to develop metrics for selecting a training algorithm and mixture model for a chosen partition of some time series data set, as documented in Chapter 3.

1.4 Multi-strategy Models

The overall design of the learning system is organized around a process of task decomposition and recombination. Its desired outcomes are: an improvement in classification accuracy through the use of multiple, customized models, *and* reduced complexity (both computational, in terms of convergence time, and structural, in terms of network complexity). This section addresses the definition and utilization of "good" subdivisions of a learning problem and the recognition of "bad" ones.

1.4.1 Applications of Multi-strategy Learning in Probabilistic Networks

One criterion for the merit of a candidate partition is the *quality of subproblems* it produces. Because my system is designed for multi-strategy learning [Mi93] using different types of temporal probabilistic networks [HGL+98, HR98b], a logical definition of quality is the expected

performance of *any* applicable network. In terms of model selection, I am interested in the expected performance of the network adjudged *best* for a particular learning subproblem definition. When metrics are properly calibrated and normalized, this allows the same evaluation function used in model selection to drive the search for an effective partition. This novel approach towards characterization of learning techniques in a multi-strategy system provides a tighter coupling of unsupervised learning and model selection. The focus of multi-strategy learning in this dissertation is to assemble a database of learning techniques. These should ideally be: *flexible* enough to express many of the memory forms that may occur in time series data; sufficiently rigorous (and *homogeneous*) for a coherent choice can be made between competing techniques; and possesses sufficiently few trainable parameters to make learning tractable.

1.4.2 Hybrid, Mixture, and Ensemble Models

Decomposable models are known by various terms in the machine learning community, including *hybrid* [WCB86, DF88, TSN90], *mixture* [RCK89, JJ94], and *ensemble* [Jo97a] models. "Hybrid" is usually a colloquial synonym for multi-strategy, but *mixture models* and *ensemble learning* have more formal definitions. Ensemble learning is defined as a parameter estimation problem that can be factored into subgroups of parameters, it is a staple of the literature on variational methods [Jo97a]. Mixture models are the type of integrative learning models that are investigated in depth in this dissertation. Chapter 4 is devoted to the discussion of how to adapt hierarchical mixtures to composite learning.

1.4.3 Data Fusion in Multi-strategy Models

Data fusion is one liability of having multiple sensors, subordinate models, or other sources of data in an intelligent system. In this research, data fusion arises naturally as a requirement due to problem decomposition. From the outset, one objective of problem decomposition has been to find a partitioning of the time series into homogeneous subsets. For a multiattribute time series, a *homogeneous* subset is a subset of attributes that, taken together, express one temporal pattern. A common example is a heterogeneous time series that comprises attributes that describe one temporal pattern (for instance, a moving average) and others that describe an additive noise model (e.g., Gaussian noise). Many approaches to time series analysis simply make the assumption that these homogeneous components exist and attempt to extract them [CT91, Ch96]. The goal of attribute partitioning is to find such partitions, on the principle that "piecewise"

homogeneous time series are easier to learn when each "piece" is mapped to the most appropriate model. The problem of fusing (or recombining) these partial models is a primary motivation for my study of data fusion. A collateral goal of attribute partitioning is to keep the overhead cost of data fusion (i.e., recombining partial models) low. The experiments reported in this dissertation demonstrate cases where partitions are indeed easier to learn and recombine.

Thus, the desired definition of *heterogeneous* is containing more than one data type, but *data type* is restricted in this research to mean "temporal pattern to be recognized" (comprising the memory form and other probabilistic characteristics that are enumerated and documented in Chapter 3 and Appendix C). Thus the definition of heterogeneity abstracts over issues of data *source*, *preprocessing* (normalization and organization), *scale* (temporal and spatial granularity), and *application* (inferential tasks). The desired definition of "decomposable" restricts heterogeneity to a particular *decomposition mechanism* (i.e., for representation and construction of subtasks, through grouping of input attributes and formation of intermediate concepts), assortment of *available models*, and *model selection mechanism*. These are qualities of the learning *system*, not the data set.

This research focuses on decomposable learning problems defined over heterogeneous time series. It is nevertheless important to be aware of time series that are heterogeneous, but not decomposable by the available tools. Such problems *should* properly be broken down into more self-consistent components for the sake of tractability and clarity; but due to lack of available models, incompleteness of the decomposition mechanism, or inaccuracy in the model selection mechanism, cannot be broken down *by the particular learning system*. Such problems are salient because the topic of this dissertation is not limited to the specific time series models and mixtures presented here. Specifically, I attempt to address the scalability of the system and its capability to support additional or alternative learning architectures. This requires consideration of the conditions under which a heterogeneous time series can be decomposed (i.e., what qualities the learning system must be endowed with, for the *learning problem* to be decomposable).

In time series analysis, the problem of combining multiple models is often driven by the *sources of data* that are being modeled. The purpose of hierarchical organization in the learning system documented here is to allow identification, from training data, of the best probabilistic match between patterns detected in the data and a prototype of some known stochastic process. This is the purpose of metric-based model selection, which – at the level of granularity applied –

is usually guided by prior knowledge of the generating processes (cf. [BD87, BJR94, Ch96, Do96]). Chapter 2 describes a knowledge-free approach for cases where such information is not available, yet the learning problem is still decomposable.

1.5 Temporal Probabilistic Networks

This section concludes the overview of the system for integrated, multi-strategy learning that is presented in this dissertation, with a survey of probabilistic network types used and compared.

1.5.1 Artificial Neural Networks

As Section 1.3 and Chapter 3 explain, the *ARIMA* family of processes is of particular interest to many current systems for time series learning. I study three variants of *ARIMA*-type models that are represented as temporal artificial neural networks: simple recurrent networks (*AR*) [Jo87, El90, PL98], time-delay neural networks or TDNNs (*MA*) [LWH87], and Gamma networks (*ARMA*) [DP92]. Algorithms for training these networks include delta rule learning (backpropagation of error) and temporal variants [RM86, WZ89]; *Expectation-Maximization* (*EM*) [DLR77, BM94], and *Markov chain Monte Carlo* (*MCMC*) methods [KGV83, Ne93, Ne96]. Finally, Chapter 4 documents how generalized linear models may be adapted to multilayer perceptrons in ANN-based hierarchical mixture models designed to boost learning performance.

1.5.2 Bayesian Networks and Other Graphical Decision Models

Bayesian networks are directed graph models of probability that can be adapted to time series learning [Ra90, HB95]. The types of Bayesian networks and probabilistic state transition models studied in this dissertation are temporal Naïve Bayesian networks (built using Naïve Bayes [KLD96]) and hidden Markov models [Ra90], built using parameter estimation algorithms – namely, EM [DLR77, BM94] and *maximum likelihood estimation* (MLE) by delta rule [BM94, Ha94]. Section 3.3 and Appendices B.1 and C.1 document these networks and the metrics used to select them.

1.5.3 Temporal Probabilistic Networks: Learning and Pattern Representation

Finally, the issue remains of how temporal patterns are represented in probabilistic networks. This is also the basis of metric design for model selection, at least for learning architectures. This question is answered by using the mathematical characterization of memory forms (called *kernel*

functions in temporal ANN learning) in the definition of metrics. Sections 3.3 and 3.4 and Appendices B.1 and C.1 discuss this characterization. The representation of temporal patterns is also empirically important to mixture models, metric normalization and system evaluation. This is addressed in Chapters 5 and 6.

2. Attribute-Driven Problem Decomposition for Composite Learning

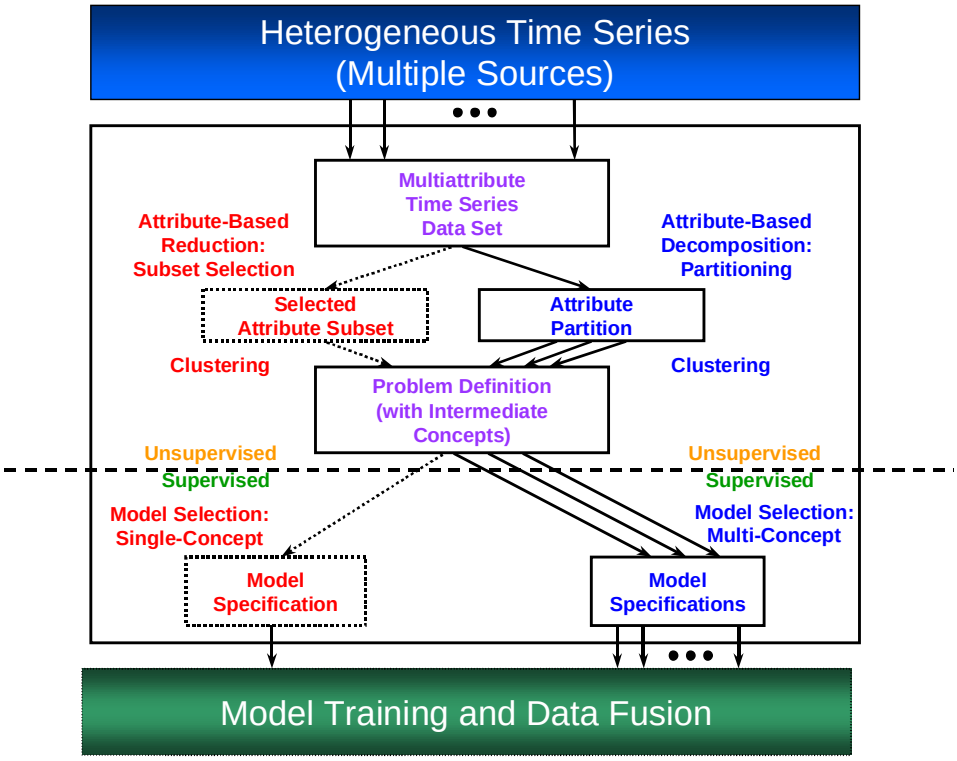

Figure 2. Systems for Attribute-Driven Unsupervised Learning and Model Selection

Many techniques have been studied for decomposing learning tasks, to obtain more tractable subproblems and to apply multiple models for reduced variance. This chapter examines *attribute-based* approaches for problem reformulation, which start with restriction of the set of input attributes on which the supervised learning algorithms will focus. First, I present a new approach to problem decomposition that is based on finding a good *partitioning* of input attributes. Kohavi's research on attribute subset selection, though directed toward a different goal for problem reformulation, is highly relevant; I explain the differences between these approaches and how subset selection may be adapted to task decomposition. Second, I compare top-down, bottom-up, and hybrid approaches for attribute partitioning, and consider the role of partitioning in feature extraction from heterogeneous time series. Third, I discuss how grouping of input attributes leads naturally to the problem of forming *intermediate concepts* in problem decomposition, and how this defines different subproblems for which appropriate models must be selected. Fourth, I survey the relationship between the unsupervised learning methods of this chapter (attribute-driven decomposition and conceptual clustering) and the model selection and supervised learning methods of the next. Fifth, I consider the role of attribute-driven problem decomposition in an integrated learning system with model selection and data fusion.

2.1 Overview of Attribute-Driven Decomposition

Figure 2 depicts two alternative systems for attribute-driven reformulation of learning tasks [Be90, Ki92, Do96]. The left-hand side, shown with dotted lines, is based on the traditional method of attribute *subset selection* [Ki92, KR92, Ko95, KJ97]. The right-hand side, shown with solid lines, is based on attribute *partitioning*, which I have adapted in this dissertation to decomposition of time series learning tasks. Given a specification for reformulated (reduced or partitioned) input, new intermediate concepts can be formed by unsupervised learning (e.g., conceptual clustering); the newly defined problem or problems can then be mapped to one or more appropriate hypothesis languages (model specifications). The new models are selected for a reduced problem or for multiple subproblems obtained by partitioning of attributes; in the latter case, a data fusion step occurs after individual training of each model.

2.1.1 Subset Selection and Partitioning

Attribute subset selection is the task of focusing a learning algorithm's attention on some subset of the given input attributes, while ignoring the rest [KR92, KJ97]. Its purpose is to discard those attributes that are irrelevant to the learning target, which is the desired concept class in the case of supervised concept learning. I adapt subset selection to the systematic decomposition of learning problems over heterogeneous time series. Instead of focusing a single algorithm on a single subset, the set of all input attributes is partitioned, and a specialized algorithm is focused on *each* subset. This research uses subset partitioning to *decompose* a learning task into parts that are individually useful, rather than to *reduce* attributes to a single useful group.

Kohavi's work on attribute subset selection is highly relevant to this approach [KJ97]. The important difference is that subset selection is designed for a single-model learning system; it considers relevance with respect to this model and tests attributes based upon a *global* criterion: the overall target and all other candidate attributes. Partitioning, by contrast, is designed for multiple-model learning. Relevance is a property of a subset and an intermediate target, and candidate attributes are tested based upon this *local* criterion.

Each alternative methodology has its pros and cons, and the difference in their respective purposes makes them largely *incomparable*. Partitioning methods are intuitively more suitable for decomposable learning problems, and we can devise a simple experiment to demonstrate this. Suppose a learning problem ***P***, defined over a heterogeneous time series, can be decomposed into

two subtasks, $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$, and a model fusion task, $\boldsymbol{P_F}$, and we are able to train models $\boldsymbol{M_1}$, $\boldsymbol{M_2}$ and $\boldsymbol{M_F}$ to some desired level of prediction accuracy. Let $\boldsymbol{S}$ be the subset of original attributes of $\boldsymbol{P}$ that are selected by a subset selection algorithm. Consider the space of models based on $\boldsymbol{S}$ that belong to a given set of available model types with trainable parameters and hyperparameters, and whose network complexity and convergence time do not exceed the totals for $\boldsymbol{M_1}$, $\boldsymbol{M_2}$ and $\boldsymbol{M_F}$. (I formalize the notion of "available model" by defining a *composite* in Chapter 3.) Suppose further that, *with high probability*, a non-modular model does not belong to this space; that is, suppose that it is improbable that a non-modular model from our "toolbox" can do the job using $\boldsymbol{S}$, as efficiently as the modular model. *If* subset selection is used only to choose $\boldsymbol{S}$ for a single non-modular model (as it often is), then we can conclude that it is less suitable than partitioning for problem $\boldsymbol{P}$. In Chapter 5, I give concrete examples of real and synthetic data sets where this scenario holds, including cases where $\boldsymbol{S}$ is the entire set of input attributes (i.e., none are irrelevant), yet there exists a useful partitioning.

Note, however, that $\boldsymbol{S}$ can still be used in a modular learning model (and can even be repartitioned first). Thus, knowing that the problem is decomposable does not conclude anything about the aptness of subset selection in general. It is still a potentially useful (and sometimes indispensable) preprocessing step for partitioning, especially considering that under the literal definition, partitioning *never* discards attributes.

2.1.2 Intermediate Concepts and Attribute-Driven Decomposition

In both attribute subset selection and partitioning, attributes are grouped into subsets that are relevant to a particular task: the overall learning task or a subtask. Each subtask for a partitioned attribute set has its own inputs (the attribute subset) and its own *intermediate concept*. This intermediate concept can be discovered using unsupervised learning algorithms, such as *k-means clustering*. Other methods, such as competitive clustering or vector quantization (using radial basis functions [Lo95, Ha94, Ha95], neural trees [LFL93], and similar models [DH73, RH98]), principal components analysis [Wa85, Ha94, Ha95], Karhunen-Loève transforms [Wa85, Ha95], or factor analysis [Wa85], can also be used.

Attribute partitioning is used to control the formation of intermediate concepts in this system. Given a restricted view of a learning problem through a subset of its inputs, the identifiable target concepts may be different from the overall one. In concept learning, for example, there are typically fewer resolvable classes. A natural way to deal with this simplification of the learning

problem is to decrease the number of target classes for the learning subproblem. Specifically, taking the original concept classes as a baseline and grouping them into equivalence classes results in a simplification of the problem. Let us refer to the learning subtasks obtained in this fashion as a *factorization* [HR98a] of the overall problem (so named because they exploit *factorial* structure in the original classification learning problem, and because submodel complexity is a polynomial factor of the overall model complexity). Attribute subset selection yields a single, reformulated learning problem (whose intermediate concept is neither necessarily different from the original concept, nor intended to differ). By contrast, attribute partitioning yields multiple learning *subproblems* (whose intermediate concepts may or may not differ, but are simpler by design when they do differ).

The goal of this approach is to find a natural and principled way to specify *how* intermediate concepts should be simpler than the overall concept. In Chapter 3, I present two mixture models, the ***H****ierarchical* ***M****ixture of* ***E****xperts* (HME) of Jordan *et al* [JJB91, JJNH91, JJ94], and the ***S****pecialist-****M****oderator* (SM) network of Ray and Hsu [RH98, HR98a]. I then explain why this design choice is a critically important consideration in how a hierarchical learning model is built, and how it affects the performance of multi-strategy approaches to learning from heterogeneous time series. In Chapter 4, I discuss how HME and SM networks perform data fusion and how this process is affected by attribute partitioning. Finally, in Chapters 5 and 6, I closely examine the effects that attribute partitioning has on learning performance, including its indirect effects through intermediate concept formation.

2.1.3 Role of Attribute Partitioning in Model Selection

Model selection, the process of choosing a hypothesis class that has the appropriate complexity for the given training data [GBD92, Sc97], is a consequent of attribute-driven problem decomposition. It is also one of the original directives for performing decomposition (i.e., to apply the appropriate learning algorithm to each homogeneous subtask). Attribute partitioning is a determinant of subtasks, because it specifies new (restricted) views of the input and new target outputs for each model. Thus, it also determines, indirectly, what models are called for.

There is a two-way interaction between the partitioning and model selection systems. Feedback from model selection is used in partition evaluation; hence, the system is a *wrapper*, defined by Kohavi [Ko95, KJ97] as an integrated system for *parameter adjustment* in supervised

inductive learning that uses feedback from the induction algorithm. This feedback can be defined in terms of a generic evaluation function over hypotheses generated by the induction algorithm. Kohavi considers parameter tuning over a number of learning architectures, especially decision trees, where attribute subsets, splitting criterion, termination condition are examples of parameters [Ko95]. The primary parameter in this wrapper system is attribute partitioning; a second, a high-level model descriptor (the architecture and learning method). The feedback mechanism is similar to that applied by Kohavi [Ko95], with the additional property that multiple model types are under consideration (each generating its own hypotheses). Furthermore, *predictive* rather than *descriptive* statistics are used to estimate expected model performance: that is, rather than measuring the actual prediction accuracy for every combination of models, I have developed evaluation functions for the individual model types and for the overall mixture. Chapter 3 further explains this design.

Model selection is in turn controlled by the attribute partitioning mechanism. This control mechanism is simply the problem definition produced by unsupervised learning algorithms. It is directly useful as an input for performance estimation, which in turn is used to evaluate attribute partitions (cf. [Ko95, KJ97]). This static evaluation measure can be applied to simply accept or reject single partitions. A more sophisticated usage that I discuss in Chapter 3 is to apply the evaluation measure as an inductive bias in a state space search algorithm. This search considers entire families of attribute partitions simultaneously [Ko95, KJ97], a form of inductive bias (cf. [Mi80, Mi82]).

2.2 Decomposition of Learning Tasks

Having presented the basic justification and design rationale for attribute partitioning, I now examine in some greater depth the way in which it can be used to decompose learning tasks defined on *heterogeneous* data sets, especially time series. I first consider the relation between attribute partitioning and subset selection, focusing on the common assumptions and limitations of both methods. I then consider alternative attribute-driven methods for decomposition of supervised inductive learning tasks, such as constructive induction. The purpose of this discussion is not only to provide further justification for the partitioning approach, but also to define its scope within the province of *change-of-representation* systems [Be90, Do96, Io96]. Finally, I assess the pertinence of attribute partitioning to heterogeneous time series, documenting it with a simple theoretical example that will be further realized in Chapter 5.

2.2.1 Decomposition by Attribute Partitioning versus Subset Selection

Practical machine learning algorithms, such as decision surface inducers [Qu85, BFOS84] and instance-based algorithms [AKA91], degrade in prediction accuracy when many input attributes are irrelevant to the desired output [KJ97]. Some algorithms such as Naïve-Bayes and multilayer perceptrons (simple feedforward ANNs) are less sensitive to irrelevant attributes, so that their prediction accuracy degrades more slowly in proportion to irrelevant attributes [DH73, BM94]. This tolerance, however, comes with a tradeoff: Naive-Bayes and feedforward ANNs with gradient learning tend to be *more* sensitive to the introduction of relevant but *correlated* attributes [JKP94, KJ97].

The problem of *attribute subset selection* is that of finding a subset of the original input attributes (features) of a data set, such that an induction algorithm, applied to only the part of the data set with these attributes, generates a classifier with the highest possible accuracy [KJ97]. Note that attribute subset selection chooses a set of attributes from existing ones in the concept language, and does not synthesize new ones; there is no feature extraction or construction cf. [Ki86, RS90, Gu91, RR93, Do96].

The problem of attribute *partitioning* is that of finding a set of nonoverlapping subsets of the original input attributes whose union is the original set. Note that this original set may contain irrelevant attributes; thus, it may be beneficial to apply subset selection as a *preprocessing* step. As for subset selection, the objective of partitioning is to generate a classifier with higher training accuracy; but the purpose of the two approaches differs in a key aspect of model *organization*. Partitioning assumes that multiple models, possibly of different types, will be available for supervised learning. It therefore has the subsidiary goals of finding an *efficient* decomposition, with components that can be *mapped to appropriate models* relatively easily. Efficiency means lower model complexity required to meet a criterion for prediction accuracy; this overall complexity can often be reduced through task decomposition. Conversely, the achievable prediction accuracy may be higher given modular and non-modular models of comparable complexity. [HR98a] documents an attribute-driven algorithm for constructing mixture models in time series learning, the latter case is demonstrated. Efficiency typically entails decomposing the problem into well-balanced components (distributing the learning load evenly). Mapping subproblems to the appropriate models has significant consequences for learning performance (both prediction accuracy and convergence rate), as I discuss in Chapter 3. An inductive bias can

be imposed on partition search (cf. [Be90, Mi80]) in order to take the available models (learning architectures and methods) into account.

Both subset selection and partitioning produce no complex or compound attributes; in Michalski's terminology [Mi83], both can be said to perform pure *selective induction*, taken by themselves. The intermediate concept formation step that follows, however, has elements of *constructive induction* [Mi83]. **Subset selection and partitioning address collateral but different problems**. In this dissertation, partitioning is specifically applied to *definition of new subproblems* in time series learning. References to *attribute-driven reformulation* are intended to include subset selection and partitioning, while *attribute-driven decomposition* refers to partitioning and other methods that *divide* the attributes rather than choose among them.

As Kohavi and John note [KJ97], subset selection is a practical rather than a theoretical concern. This is true for attribute partitioning as well. While the optimal Bayesian classifier for a data set need never be restricted to a subset of attributes, two practical considerations remain. First, the true target distribution is not known in advance [Ne96]; second, it is intractable to fit or even to approximate [KJ97]. Modeling this unknown target distribution is an aspect of the classic bias-variance tradeoff [GBD92], which pits model generality (bias reduction, or "coding more parameters") against model accuracy (variance reduction, or "refining parameters"). Intractability of finding an optimal model, or hypothesis, is a pervasive phenomenon in current inductive learning architectures such as Bayesian networks [Co90], ANNs [BR92], and decision surface inducers [HR76]. An important ramification of these two practical considerations is that an "optimal" attribute subset or partitioning should be defined with respect to the *whole learning technique*, in terms of its change of representation, inductive bias, and hypothesis language. This includes both the learning algorithm (as Kohavi and John specify [KJ97]) and the hypothesis language, or *learning architecture* (i.e., the model parameters and hyperparameters [Ne96]).

2.2.1.1 State Space Formulation

Figure 3 contains example state space diagrams for attribute subset selection (subset inclusion) and partitioning. Each state space is a partially ordered set (poset) with an ordering relation $\preceq$ that is transitive and asymmetric. The ordering relation corresponds to operators that navigate the search space (i.e., move up or down in the poset, between connected vertices).

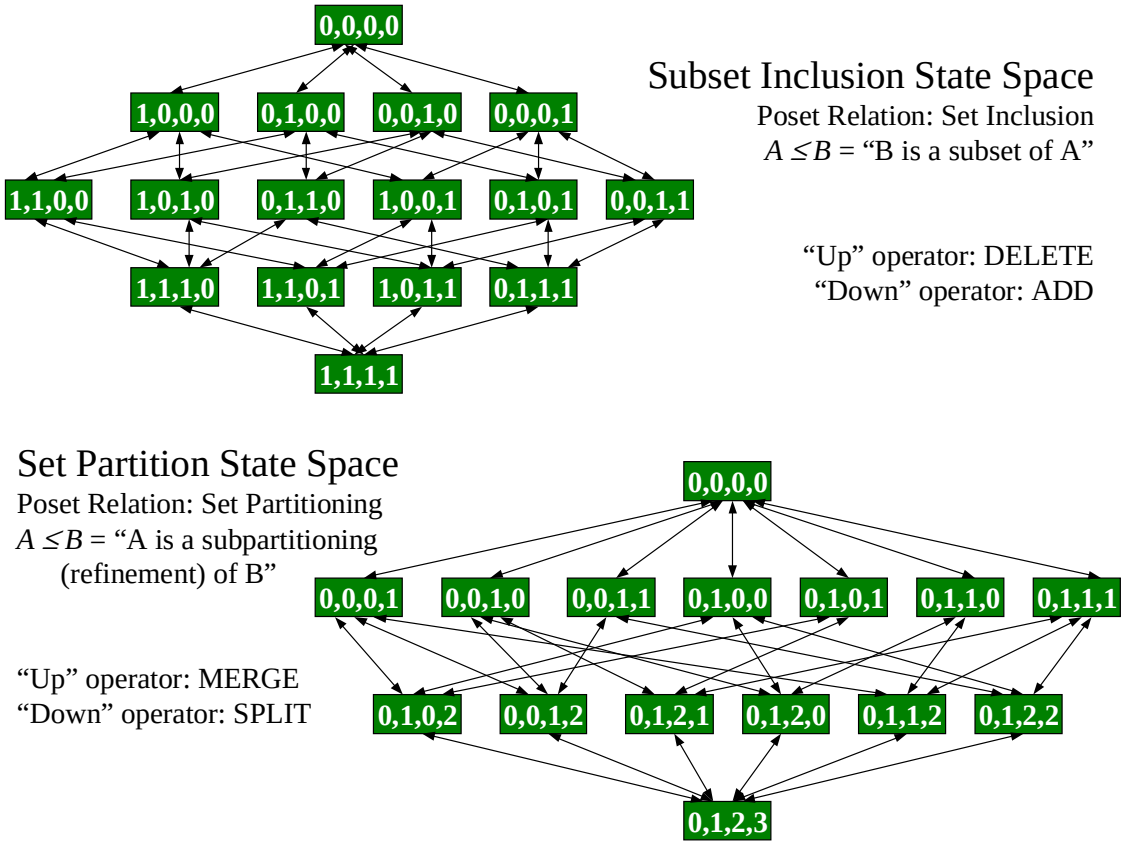

Figure 3. The state space diagrams for subset selection and partitioning

The ordering relation for subset selection is set inclusion, the converse of set containment; it is usually denoted $\supseteq$. The set is ordered in this fashion to conform to the "top down" convention for state space search (i.e., the vertex 0,0,0,0, denoting the empty set, is the root). Note that this usage of "top down" refers to the *search*, not the process of constructing an attribute set, which is instead best described as "bottom up" (because we start with 0 attributes and add more). For n attributes, there are n bits in each state of the subset inclusion state space, each an indicator of whether an attribute is present (1) or absent (0) [KJ97]. The relation $\supseteq$ corresponds to operators that **add** or **delete** single attributes to or from a candidate subset; these are analogous to stepwise linear regression operators (forward selection and backward elimination) in statistics [Ri88, KJ97]. The size of the state space for n attributes is $O(2^n)$, so it is impractical to search the space exhaustively, even for moderate values of n.

The ordering relation for subset partitioning is set partitioning; for example, A = {{1}{2}{3, 4}} is a subpartitioning of B = {{1, 2}{3, 4}}, so we can write $A \leq B$. The partitions are coded according to membership labels: for example, 0,1,1,2 denotes the partition {{1}{2, 3}{4}}.

Thus, partition *A* in the above example would be coded 0,1,2,2; partition *B*, 0,0,1,1. The root denotes {{1, 2, 3, 4}} (i.e., search begins with a default state that denotes one monolithic class, corresponding to a *non-modular* model) and the bottom element denotes {{1}{2}{3}{4}} (corresponding to a *completely decomposable* model). For *n* attributes, there are *n* labels in each state. The relation $\preceq$ corresponds to operators that **split** a single subset or **merge** a pair of subsets of a candidate partition. The size of the state space for *n* attributes is B_n, the *n*th Bell number, defined as follows[4]:

$$B_n = \sum_{k=0}^{n} S(n,k)$$

$$S(n,k) = \begin{cases} 0 & \text{if } n < k \text{ or } k = 0, n \neq 0 \\ 1 & \text{if } n = k \\ S(n-1,k-1) + kS(n-1,k) & \text{otherwise} \end{cases}$$

Thus, it is impractical to search the space exhaustively, even for moderate values of *n*. The function B_n is $\omega(2^n)$ and $o(n!)$, i.e., its asymptotic growth is strictly *faster* than that of 2^n and strictly *slower* than that of $n!$. It thus results in a highly intractable evaluation problem if all partitions are considered. For practical illustration, I implemented a simple dynamic programming algorithm to generate partitions according to the recurrence above [CLR90]. Experiments using all $B_9 = 21147$ and $B_{10} = 115975$ partitions of data sets with 9 and 10 attributes respectively (6 of which belonged to a 1-of-C coding of a single measurement) were performed on a 300-MHz Intel Pentium II workstation running Windows NT 4.0. The data sets contained 367 discrete exemplars each. Computing the mutual information between each subset of each partition and the overall (5-valued, 1-of-C-coded) target concept took approximately 5 minutes of wall clock time for the 9-attribute version and approximately 2.5 hours for the 10-attribute version. Without customized paging, memory consumption for the 10-attribute experiment approached the amount of primary memory for the workstation (256 megabytes). Thus, even an 11-attribute partition would be prohibitive to optimize by brute force (i.e., without search). Chapter 5 and Appendix A document this performance issue.

2.2.1.2 Partition Search

Because naïve enumeration of attribute partitions by is highly intractable, a logical next step is to optimize them by state space search. This entails an evaluation function over states in the

[4] *S* is a recurrence known as the Stirling Triangle of the Second Kind. It counts the number of partitions of an *n*-set into *k* classes [Bo90].

partition state space, depicted in Figure 3. *Informed* (heuristic-based) search algorithms that apply to this problem formulation are: hill-climbing (also called greedy search or gradient ascent), best-first, beam search, and A* [BF81, Wi93, RN95]. The generic template for these informed search algorithms is as follows:

Search algorithm template

---

1. Put the initial state (root vertex) on the OPEN list;
   CLOSED list ← ∅, BEST ← initial state.
2. **while** the BEST vertex has changed within the last *n* iterations **do**
3. Let $v = \arg\max_{w \in \text{CANDIDATES}} f(w)$
   (get the state from CANDIDATES with maximal f(*w*)).
4. Remove *v* from OPEN, add *v* to CLOSED.
5. If $f(v) - \varepsilon > f(BEST)$, then BEST ← *v*.
6. Expand *v* by applying all *downward* operators to *v*, yielding a list of *v*'s children.
7. For each child not in the CLOSED or OPEN list, evaluate and add to the OPEN list.
8. Update CANDIDATES.
9. Return BEST.

The specific algorithms are differentiated by the definitions of f (step 3) and of CANDIDATES (step 7):

Algorithm	f(w)	CANDIDATES
Hill climbing	h(w)	the list of current children of *v*
Best-first	h(w)	the entire OPEN list
Beam search	h(w)	first *k* elements of the OPEN list sorted in decreasing order of *f*
*A**	h(w) + g(w)	the entire OPEN list

where h(w) is the heuristic evaluation function (the higher, the better) and g(w) is the cumulative root-to-vertex fitness (the actual total).

Chapter 5 documents the results for partition search using each of these algorithms.

2.2.2 Selective Versus Constructive Induction for Problem Decomposition

This section presents a comparative survey of methods for decomposing a data set by reformulating attributes. So far, this chapter has focused on selective induction methods rather than constructive induction (the *synthesis* of new attributes). I now present a brief justification for this choice.

Constructive induction, the formation of complex or compound attributes, is a method for transforming low-level attributes into useful ones [Mi83, RS90, RR93, Do96]. It divides inductive learning into two phases: *attribute synthesis* (also known as *feature construction*) and *problem redefinition* (also known as *cluster description*) [Do96]. Attribute synthesis produces a transformed input by composing attributes using operators (e.g., arithmetic composition). This constrains the choice of suitable hypothesis languages for describing the target concept, which in turn defines a new learning problem. Much of computational learning theory is devoted to the question of how to choose one of these hypothesis languages [Ha89, KV91]. The division of induction into attribute synthesis and problem redefinition is analogous to Figure 2. First, the input specification is transformed – in this case, by applying operators to form new attributes rather than selecting or partitioning them. The redefinition of the learning problem is accomplished by forming new concepts, and selecting the appropriate hypothesis language or languages. These steps are organized into a single abstract phase in the traditional constructive induction framework [Mi83, Do96]. I pay explicit attention to the boundary between intermediate concept formation and model selection, because this distinction is important to decomposition of time series learning problems, and to the modular and hierarchical mixture approaches I apply in this dissertation.

The objective of attribute synthesis in the constructive induction framework is to combat the phenomena of *dispersion* and *blurring* [RS90, RR93, Do96]. Dispersion is a property of concepts, wherein exemplars belonging to each class are scattered throughout instance space, *or the projection thereof under a particular attribute subset* [RS90]. Blurring is another property of concepts, wherein attributes that do not appear individually useful turn out to be jointly useful [RR93]. Another way to say this is that *relevance is a joint property of attributes, not an independent one* [He91, Ko95, KJ97]. Blurring is a converse property of decomposability by attributes (wherein attributes that are jointly useful can be separated without loss of coherence); it is thus a primarily *inter*-subset property from the point of view of problem decomposition. Dispersion is a symptom of attribute subset "insufficiency", meaning that more knowledge,

orderings, or additional attributes are required for coherence; it is thus primarily an *intra*-subset property. The reason for concentrating on attribute partitioning instead of attribute synthesis in this dissertation, therefore, is that partitioning is more *directly conscious* of the issues of problem decomposition, redefinition, model selection, and data fusion (the right hand side of Figure 2) toward which my approach aims.

This concludes my brief justification of partitioning methods in constructive induction. Some experimental comparisons are documented in Chapter 5.

2.2.3 Role of Attribute Extraction in Time Series Learning

This research focuses on multiattribute time series, especially when they are decomposable by attribute partitioning. Thus, the data sets that I consider are restricted to multiattribute time series, where each attribute represents a single channel of information through time (also known as *multichannel* time series). Attribute partitioning as applied to multichannel time series achieves problem redefinition by grouping attributes together, to produce subsets that we can think of as "large attributes". This is especially apropos for multichannel time series because large attributes (the inputs to learning subproblems) occur naturally based upon the multiple sources of data, such as sensors. In Chapters 4 and 5, I document several real-world and synthetic time series that admit this type of decomposition.

The next section discusses how attribute partitioning may be used to drive problem redefinition, the second phase of constructive induction.

2.3 Formation of Intermediate Concepts

2.3.1 Role of Attribute Grouping in Intermediate Concept Formation

Attribute partitioning produces a reformulation of the input to supervised concept learning. We can think of this reformulation as problem *decomposition* from the point of view of multiattribute learning and problem *specialization* from the point of view of multi-strategy learning. The result, however, is the same: each subproblem has its input restricted to a subset of the original attributes. This restriction is the driving force behind intermediate concept formation, part of the second phase of constructive induction. Intermediate concept formation is also known as *cluster description* [Do96] when the learning paradigm is single-concept constructive induction. This dissertation deals with the formation of intermediate concepts in support of a

hierarchical, multi-strategy time series learning system. The principle is similar, however, as the same unsupervised learning methods may be used to re-target the desired outputs whether there is one set of inputs or several. Thus, attribute partitioning prepares the input; intermediate concept formation, the output; and the result is a set of redefined learning subproblems for which model selection and training are made easier.

2.3.2 Related Research on Intermediate Concept Formation

The same techniques used to form new concepts from unlabeled data ("deciding what to learn") can be brought to bear in attribute-driven problem decomposition, namely: conceptual clustering and vector quantization, self-organizing systems, and other concept discovery algorithms. Conceptual clustering methods are those that group exemplars into conceptually simple classes, based upon attribute-centered criteria such as syntactic constraints, prior relational knowledge, and prior taxonomic knowledge [SM86, Mi93]. Other clustering techniques include competitive clustering or vector quantization methods. A well-known example is *k-means* clustering, which finds prototypes (cluster centers) through an iterative refinement algorithm [DH73, Ha94, Ha95]. Competitive clustering using radial-basis functions [Ha94, Ha95], neural trees [LFL93], and other distance-based models has been studied in the artificial neural networks and information processing literature [RN95]. Vector quantization, the problem of finding an efficient intermediate representation (also known as a *codebook*) for learning (or *model estimation*) problems in signal processing, has also been heavily researched and is the source of a number of algorithms that apply to concept formation [DLR77, Le89]. Self-organization is a process of unsupervised learning whereby significant patterns or features in the input data are discovered using an adaptive model [RS88, Ko90, Ha95]. Architectures such as self-organizing feature maps, which relate the topology of input data to its probability distribution, have been discovered by Kohonen [Ko90] and investigated by Ritter and Schulten [RS88]. Finally, domain-specific and architecture-specific algorithms for concept discovery from data, such as hidden-variable induction algorithms for Bayesian networks [Pe88, LWYB90, CH92], also have the capability to produce intermediate concepts.

2.3.3 Problem Definition for Learning Subtasks

The formation of intermediate concepts (learning targets, or desired output attributes) completes a process of subproblem definition as depicted in Figure 2. This is, however, only the beginning of the overall process of problem decomposition, which comprises problem redefinition, model selection, and model reintegration (data fusion). What is accomplished by

partitioning attributes and defining a new target concept for each subset is the specification of a *self-contained* learning subproblem for which a *specialized model* can be selected. In multiattribute learning, "self-contained" means that the attributes are sufficient for a well-defined intermediate target (i.e., exhibit low *dispersion* [RR93] with respect to that target) and distribute the learning task evenly (i.e., take advantage of decomposability, resulting in controlled *blurring* [RS90] *across attributes*). Chapters 3, 4, and 5 examine how the quality of this subproblem definition affects the subsequent model selection, training and integration phases.

2.4 Model Selection with Attribute Subsets and Partitions

Model selection is the process of finding a hypothesis class, or language, that has the appropriate complexity for the given training data [GBD92, Sc97]. This section previews the role of attribute subset selection and partitioning in model selection. It then shows how attribute partitioning enhances the more interesting aspect of model selection where multiple (not necessarily identical) models are called for.

2.4.1 Single versus Multiple Model Selection

The model selection problem can be described as optimization of model organization (e.g., determining the topology of a Bayesian network or artificial neural network, also known as *structuring* [Pe86, CH92]), hyperparameters [Ne96], or parameters [GBD92, Sc97]. In all of these cases, model selection can be directed towards a single problem definition or towards multiple subnetworks, groups of hyperparameters, or groups of parameters. Multiple model selection is more salient to problems decomposed using attribute partitioning, for the obvious reason that partitions have multiple subsets and thereby induce multiple subproblems. It is also of greater interest in the context of multi-strategy learning [HGL+98]. In this research, I am specifically interested in the case where multiple models may be applied, but the learning architectures and methods (mixture model and individual training algorithms) are not necessarily identical. This is the case where the *data set* is heterogeneous and the *learning problem* is decomposable.

2.4.2 Role of Problem Decomposition in Model Selection

Problem decomposition by attribute-based methods affects model selection in two ways: directly, via input reformulation, and indirectly, via problem redefinition and reformulation of the data. The direct relationship between partitioning and multiple model selection (and between

subset selection and single model selection) is predicated upon attribute-based model selection decisions. That is, whatever decisions can be made about the hypothesis language based purely upon syntactic specification of the input (including which attributes belong to a given subset) are directly influenced by the partitioning used. The indirect effects can also be based upon syntactic properties of the subproblems, but only if the reformulated output is taken into account. Note, however, that this output (the intermediate concept) may be found by *purely syntactic* clustering.

Typically, neither the problem redefinition nor the model selection process is based only upon syntactic properties; the statistical content of the data thus plays an important role. This dissertation, being primarily concerned with multiattribute time series data and the decomposition thereof, thus focuses on the indirect case. For example, attributes can be grouped together such that each *projection* of the data (i.e., the "columns" corresponding to each subset) [Pe97] can be learned using a particular temporal model such as an *autoregressive* or *moving average* model [Mo94]. In this case, the prediction that a projection "can be learned" using a given model (hypothesis language) is in the purview of model selection, and certainly requires information about the reformulated data (namely, how tractable the new subproblem is given a candidate model). This is the subject of Chapter 3.

2.4.3 Metrics and Attribute Evaluation

In this dissertation, I develop a quantitative method for *coarse-grained* model selection. By coarse-grained, I mean determination of very high-level hyperparameters such as the network architecture, mixture model, and training algorithm. These can be adjusted automatically, but are traditionally considered beyond the "core" learning system. I argue in Chapter 3 that for decomposable learning problems, indiscriminate use of "nonparametric" models such as feedforward and recurrent artificial neural networks is too unmanageable. That is, leaving the tasks of problem decomposition, model adaptation (i.e., changing model parameters and hyperparameters to attain the appropriate internal representation for hypotheses), and model integration (making coherent sense out of an hybrid [Mi93], mixture [JJ94, Bi95], ensemble [Jo97a], or *composite* model) is too much to expect! This is especially true in applications of time series learning (where my performance element is classification for monitoring and prediction), because decision surfaces are more sensitive to error when the target concept is a future event of importance. The alternative I propose and investigate is to assume that there is a "right tool for each job" in a decomposable learning problem, where "each job" is found by

attribute partitioning and the "right tool" is identified by coarse-grained, quantitative (or *metric-based*) model selection.

The interaction between this model selection system and the partitioning algorithm is that *evaluation metrics for models provide feedback for partitions*. Figure 1, in Chapter 1, illustrates this design. Partitioning and model selection operate concurrently, with the partitioning algorithm producing *candidate partitions* (either by naïve enumeration or by search). Model selection evaluates learning architectures with respect to *each subset* of a partition (recall that an objective of attribute-driven decomposition is to map subproblems to different learning architectures). It also evaluates learning methods (mixture models and training algorithms) *as a total function of the partition*. This second component of the evaluation metric can be fed back to the partition search algorithm as a heuristic evaluation function. The partitioning algorithm, in turn, uses this feedback to produce better candidate partitions.

2.5 Application to Composite Learning

2.5.1 Attribute-Driven Methods for Composite Learning

The *raison d'être* of model selection in concept learning is to produce a hypothesis language specification for supervised learning. When this choice has been committed to the acceptable partition or partitions, training can proceed independently with the specified model for each partition. It is the object being specified that I refer to as a *composite*: the complete problem definition (an attribute partition, mixture model, learning algorithm, and selected architectures for each subset – a temporal, probabilistic subnetwork in each case). Thus, the entire learning system revolves around attribute partitioning, intermediate concept formation, and multiple model selection, with each phase driving the subsequent ones and feedback from model selection to partitioning. A final consideration, which I summarize in Section 2.5.3 and address in depth in Chapter 4, is how multiple models are recombined to improve prediction accuracy.

2.5.2 Integration of Attribute-Driven Decomposition with Learning Components

To fully understand the interaction between attribute-driven decomposition and multi-strategy learning with model selection, it is useful to briefly survey some existing research. The relevant systems are integrative learning systems with attribute evaluation. These include the *wrapper* and *filter* approaches, described by Kohavi [Ko95]. The *attribute filter* approach, documented by

Kohavi and John [KJ97] and previously investigated by Almuallim and Dietterich [AD91], Kira and Rendell [KR92], Cardie [Ca93], and Kononenko [Ko94], is a simple methodology for attribute subset selection that evaluates attributes outside the context of any induction algorithm. The effective assumption is that relevance is an entity independent of the hypothesis language – a dubious assumption in most cases and highly susceptible to properties such as greediness in decision surface inducers [KJ97]. While technically, knowledge of the hypothesis language can be captured by the selection function, this is highly impractical as Kohavi and John document in their survey of filtering techniques [KJ97]. John, Kohavi, and Pfleger have identified a number of weaknesses in attribute filtering, and derived several pathological datasets to demonstrate these weaknesses [JKP94]. I show in Chapters 5 and 6 that such pathologically bad cases constitute a combinatorially significant proportion of datasets. This is demonstrable whether the data are generated exhaustively, deterministically according to constraints, or stochastically according to sampling constraints.

The *wrapper* approach to attribute subset selection, pioneered by Kohavi [Ko95], is an alternative to attribute filtering that takes the induction algorithm into account. I adapt the generalized wrapper developed by Kohavi and John [KJ97] to attribute partitioning in support of integrated model selection. This approach is depicted in Figures 22 and 23, in Chapter 6.

The advantages of using wrappers for attribute partitioning, in addition to granting a facility for taking a particular induction algorithm into account, are as follows:

1. When multiple models (hypothesis languages) are available, the *components* of a partition (its subclasses) should be treated as separate parts of a decomposed learning task. Each part has its own learning target and requires a hypothesis language well suited to expressing it. The task of attribute-driven constructive induction, as described in Sections 2.2-2.3, is to find this target; the task of model selection, described in Chapter 3, is to find an efficient corresponding language. In this case, the hypothesis language is determined by two criteria: the time series representation (*learning architecture*) for a particular subset and the *learning method* for an entire partition.
2. A partition should be evaluated on the basis of *coherence* (i.e., each subclass must be *cohesive* and have attributes relevant to the local or specialized learning target) and *efficiency* (it should not involve too much computational effort to combine models for each subclass).

A typical method for implementing inductive bias in this integrated system is heuristic search over the state space of partitions [BF81, RN95]. This technique generalizes over a number of algorithms that incorporate quantitative feedback. It is similar to the approaches investigated by Kohavi and John [KJ97], but the difference is that the evaluation function is computed over partitions, not subsets.

2.5.3 Data Fusion and Attribute Partitioning

To complete the process of multi-strategy learning depicted on the right-hand side of Figure 2, a system must be able to reintegrate the trained components. I accomplish this by organizing a hierarchical mixture model based upon each accepted attribute partition. This mixture model contains *specialist* subnetworks that are trained using the subproblem definitions and *moderator* networks designed to integrate these subnetworks. Each specialist subnetwork takes as input the "columns" of the training data specified by an attribute subset, and is trained using the intermediate concept formed for those columns) and uses the model specification found by model selection. Each specialist network belongs to a given type of probabilistic network such a simple recurrent network or a time-delay neural network; this architecture is specified for each subproblem. Finally, the moderator networks are selected based on the same metric-based method (documented in Chapter 3). This process combines subnetworks in a bottom-up fashion until there is a single moderator network. The tree-structured overall network can be trained level-by-level (and supports *stacking* [Wo92] as a statistical validation method). The way that moderator targets are defined is a function of the mixture model. The two general categories of mixture model that I consider are *specialist-moderator (SM) networks* [RH98] and *hierarchical mixtures of experts (HME)* [JJ94]. The hallmark of SM networks is bottom-up refinement of intermediate concepts (and, conversely, top-down decomposition of learning tasks) [RH98, HR98a]. The hallmark of HME is iterative specialization of subnetworks to distribute the learning task evenly across moderators and among specialists. The choice of mixture models is discussed in Chapter 3. The data fusion properties are the subject of Chapter 4 and are addressed in greater depth there.

3. Model Selection and Composite Learning

	Specialist-Moderator Network			Hierarchical Mixture of Experts		
	Gradient	EM	Metropolis	Gradient	EM	Metropolis
Gamma Memory	✓	–	★	✓	★	★
Time Delay Neural Network (TDNN)	✓	–	★	✓	★	★
Simple Recurrent Network (SRN)	✓	–	★	✓	★	★
Hidden Markov Model (HMM)	✓	✓	–	✓	✓	–
Temporal Naïve Bayesian Network	★	★	–	★	★	–

Legend: ✓ = known combination; ★ = current research; – = beyond scope of current research

Table 1. Learning architectures (rows) versus learning methods (columns)

The ability to decompose a learning task into simpler subproblems prefigures a need to map these subproblems to the appropriate models. The general mapping problem, broadly termed *model selection*, can be addressed at very minute to very coarse levels. This chapter examines quantitative, metric-based approaches for model selection at a coarse level. First, I present and formalize a new approach to multi-strategy supervised learning that is enabled by attribute-driven problem decomposition. This approach is based upon a natural extension of the *problem definition and technique selection* process [EVA98].[5] Second, I present a rationale for using quantitative metrics to accumulate evidence in favor of particular models. This leads to the design presented here, a metric-based selection system for *time series learning architectures* and *general learning methods*. Third, I present the specific time series learning architectures that populate part of my collection of models, along with the metrics that correspond to each. Fourth, I present the training algorithms and specific mixture models that also populate this collection, along with the metrics that correspond to each. Fifth, I document a system I have developed for normalizing metrics and a method for calibrating the normalization function from training corpora.

3.1 Overview of Model Selection for Composite Learning

Table 1 depicts a database of learning techniques. Each row lists a temporal learning architecture (a type of artificial neural network or Bayesian network); each column, a specific

learning method (a type of mixture model and learning algorithm). This section presents a new metric-based algorithm for mapping each component of a decomposed time series learning problem to an entry in this database. This algorithm selects the learning technique *most strongly indicated by the characteristics of each component*. The objective of this approach is not only to map subproblems to *specialized* techniques for supervised learning, but also to map the combined learning problem to the most appropriate *mixture model* and supervised training algorithm. This process is enabled by the systematic decomposition of learning problems and the redefinition of subproblems. My attribute-driven method for problem decomposition, given in Chapter 2, comprises partitioning of input attributes and "cluster definition" (retargeting of intermediate outputs to newly discovered concepts). We begin the next phase with the resulting subproblems.

3.1.1 Hybrid Learning Algorithms and Model Selection

By applying attribute-driven methods to partition a time series learning task and formulate intermediate concepts (i.e., specialized targets) for each subtask, we have obtained a *redefinition of the overall supervised learning problem*. This redefinition is *modular*, in that training of the individual components can occur concurrently and locally (even independently, if the mixture model so specifies). Another benefit of this local computation is that it supports a hierarchy of multiple models. This dissertation considers two ways in which a hierarchy of models can capture different aspects of the learning task as defined by partitioning: through specialization of redundant models (top-down), and through refinement of coarse-grained specialists (bottom-up). Both methods are designed to reduce variance and to be based upon attribute partitioning. To properly account for the interaction between automatic methods for problem decomposition and automatic methods for model selection, a characterization of *model types* is needed. In order to partially automate the kind of high-level decisions that practitioners of multi-strategy learning make, this characterization must indicate the *level of match* between a subproblem and each specific type of learning model under consideration. This provides the capability to predict the expected performance, given the candidate subproblem and model.

3.1.1.1 Rationale for Coarse-Grained Model Selection

Model selection is the problem of choosing a hypothesis class that has the appropriate complexity for the given training data [St77, Hj94, Sc97]. Quantitative, or *metric-based*, methods for model selection have previously been used to learn using highly flexible models with many

[5] I will henceforth use the term *model selection* to refer to both traditional model selection and the metric-based methods for technique selection as presented here.

degrees of freedom [Sc97], but with no particular assumptions on the structure of decision surfaces (e.g., that they are linear or quadratic) [GBD92]. Learning without this characterization is known in the statistics literature as *model-free estimation* or *nonparametric statistical inference*. A premise of this dissertation is that, for learning from heterogeneous time series, indiscriminate use of such models is too unmanageable. This is especially true in diagnostic monitoring applications such as crisis monitoring, because decision surfaces are more sensitive to error when the target concept is a catastrophic event [HGL+98].

The purpose of using model selection in *decomposable* learning problems is to *fit* a suitable hypothesis language (model) to each subproblem. A subproblem is defined in terms of a subset of the input and an intermediate concept, formed by unsupervised learning from that subset. Selecting a model entails three tasks. The first is *finding partitions* that are consistent enough to admit at most one "suitable" model per subset. The second is *building a collection of models* that is flexible enough so that some partition can have at least one model matched to each of its subsets. The third is to *derive a principled quantitative system for model evaluation* so that exactly one model can be correctly chosen per subset of the acceptable partition or partitions. These tasks indicate that a model selection system *at the level of subproblem definition* is desirable, because this corresponds to the granularity of problem decomposition, the design choices for the collection of models, and the evaluation function. This is a more comprehensive optimization problem than traditional model selection typically adopts [GBD92, Hj94], but it is also approached from a less precise perspective; hence the term *coarse-grained*.

3.1.1.2 Model Selection versus Model Adaptation

For heterogeneous time series learning problems, indiscriminate use of nonparametric models such as feedforward and recurrent artificial neural networks is often too unmanageable. As [Ne96] points out, the models that are referred to as *nonparametric* in ANN research actually do have well-defined parameters (trainable weights and biases) and hyperparameters (distribution parameters for priors). A major difficulty and drawback of using ANNs in time series learning is the lack of semantic clarity that results from having so many degrees of freedom. Not only is the optimization problem proportionately more difficult, but it is often nontrivial (or entirely infeasible) to map "internal" parameters to concrete uncertain variables from the problem [Pe95]. A theoretical result that is often abused in this context is that a neural network with sufficient degrees of freedom can express any hypothesis [RM86]. This does *not*, however, mean that a single, maximally flexible model should always be applied instead of multiple specialized ones.

The syndrome that I refer to as "indiscriminate use" is the typically mistaken assumption that, even for decomposable learning problems, it is an effective use of computational power to apply the single model. In effect, that single model is being required to achieve automatic problem decomposition, relevance determination, localized model adaptation, and data fusion. The alternative suggested by the "no-free-lunch" principle is to make these processes *explicit*, and attempt to provide some unifying control over them through a high-level algorithm.

The remainder of this section describes a novel type of coarse-grained, metric-based model selection that selects from a known, fixed "repertoire" or "toolbox" of learning techniques. This is implemented as a "lookup table" of architectures (rows) and learning methods (columns). Each architecture and learning method has a characteristic that is positively (and uniquely, or almost uniquely) correlated with its expected performance on a time series data set. For example, naïve Bayes is most useful for temporal classification when there are many discriminatory observations (or symptoms) all related to the hypothetical causes (or syndromes) that are being considered [KSD96, He91]. The *absolute* strength of this characteristic is measured by an indicator metric. To determine its *relative* strength, or *dominance*, this measure must be normalized and compared against those for other characteristics. For example, the indicator metric for temporal naïve Bayes is simply a score measuring the degree to which observed attributes are relevant to discrimination of every pair of hypotheses. The highest-valued metric thus identifies the dominant characteristic of a subset of the data. This assumes that the subset is sufficiently homogeneous for a single characteristic to dominate *and to be recognized*.

The metric-based approach literally emphasizes *selection* of models, whereas most existing approaches are more parameter-intensive, and might better be described as model *adaptation*. This is an important distinction when attempting to learn from heterogeneous data. Model adaptation then tends to suffer acutely from the complexity costs of having many degrees of freedom, while problem decomposition with coarse-grained model selection can relieve some of this overhead.

3.1.2 Composites: A Formal Model

This section defines *composites*, which are attribute-based subproblem definitions, together with the learning architecture and method for which this alternative representation shows the strongest evidence.

Definition. A *composite* is a set of tuples $\mathbf{L} = ((A_1, B_1, \theta_1, \gamma_1, S_1), \ldots, (A_k, B_k, \theta_k, \gamma_k, S_k))$, where A_i and B_i are sets of input and output attributes, θ_i and γ_i are *names* of network parameters and hyperparameters cf. [Ne96] (i.e., the learning architecture), and S_i is the *name* of a learning method (a training algorithm and a mixture model specification).

A composite is depicted in Figure 1 of Chapter 1, in the box labeled "learning techniques".

Intuitively, a composite describes all of the model descriptors that can be chosen by the overall learning system. This includes the trainable weights and biases; the specification for network topology (e.g., number, size, and connectivity of hidden layers in temporal ANNs); the initial conditions for learning (prior distributions of parameter values); and most important for time series learning, the process model. The process model describes the type of temporal pattern that is anticipated and the stochastic process assumed to have generated it. In terms of network architecture, it specifies the *memory type* (the mathematical description of the pattern as a finite function of time) [Mo94, MMR97].

A composite also specifies the network types for moderator networks (also known as *gating* [JJ94], *fusion* [RH98], or *combiner* [ZMW93] networks) in the mixture model. Because the problem is decomposed by attribute partitioning, a moderator network is always required whenever there is more than one subset of attributes. I discuss this aspect of composites in Section 3.4 and in Chapter 4. Finally, a composite specifies the training algorithm to be used for an entire partition (i.e., each subproblem, as defined for each subset). Both the mixture model and the training algorithm are selected based upon quantitative analysis of the entire partition, as I explain in Section 3.4.

Property. In a learning system where task decomposition is driven by attribute partitioning, the set union of A_i is the original set of attributes A (by definition of a *partition*) and each set of output attributes B_i is an intermediate concept corresponding to A_i.

The reason why attribute subsets are included in a composite is that they specify the way that a problem is partitioned with sufficient information to build the subnetworks for each subproblem (i.e., to extract the input and produce the target outputs for every subnetwork). Thus, a composite contains every specification needed to generate a hierarchical model (specialists and moderators)

given the training data. Composites are generated using the algorithm given in the following section.

3.1.3 Synthesis of Composites

A general algorithm for composite time series learning follows.

Given:

1. A (multiattribute) time series data set
   $D = ((\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}))$ with input attributes
   $\boldsymbol{A} = (a_1, \ldots, a_I)$ such that $\boldsymbol{x}^{(i)} = (x_1^{(i)}, \ldots, x_I^{(i)})$ and output attributes $\boldsymbol{B} = (b_1, \ldots, b_O)$ such that $\boldsymbol{y}^{(i)} = (y_1^{(i)}, \ldots, y_O^{(i)})$
2. A constructive induction function F (as described in Chapter 2) such that $F(A, B, D) = \{(A', B')\}$, where A' is an attribute partition and B' is a group of intermediate concepts for each attribute subsets, found by problem redefinition (cluster description) using A'.

Algorithm **Select-Net** *(D, **A**, **B**, F)*

repeat

Generate a candidate representation $(A', B') \in F(A, B, D)$.

for each learning architecture $\boldsymbol{\tau}^a$

for each subset A_i' of A'

Compute *architectural* metrics $\boldsymbol{x}_{i\boldsymbol{\tau}}{}^{a} = m_{\boldsymbol{\tau}}^{a}(A_i', B_i')$ that evaluate $\boldsymbol{\tau}^a$ with respect to (A_i', B_i').

for each learning architecture $\boldsymbol{\tau}^d$

Compute *distributional* metrics $\boldsymbol{x}_{\boldsymbol{\tau}}{}^{d} = m_{\boldsymbol{\tau}}^{d}(A', B')$ that evaluate $\boldsymbol{\tau}^d$ with respect to (A', B').

Normalize the metrics $\boldsymbol{x}_{\boldsymbol{\tau}}$ using a precalibrated function $\boldsymbol{G}_{\boldsymbol{\tau}}$ – see Equation 1.

Select the most strongly prescribed architecture (θ, γ) and learning method S for (A', B'), i.e., the table entry (row and column) with the highest metrics.

if the fitness (strength of prescription) of the selected model meets a predetermined threshold

then accept the proposed representation and learning technique $(A', B', \theta, \gamma, S)$

until the set of plausible representations is exhausted

Compile and train a *composite,* **L**, from the selected complex attributes and techniques.

Compose the classifiers learned by each component of **L** using data fusion.

$$t_\tau : \text{shape parameter}$$

$$\lambda_\tau : \text{scale parameter}$$

$$G_\tau(x_\tau) = \int_0^{x_\tau} f_\tau(x)\,\mathrm{d}\,x$$

$$f_\tau(x) = \frac{\lambda_\tau\, \mathrm{e}^{-\lambda_\tau x} (\lambda_\tau x)^{t_\tau - 1}}{\Gamma(t_\tau)}$$

$$\Gamma(t_\tau) = \int_0^{\infty} \mathrm{e}^{-y}\, y^{t_\tau - 1}\,\mathrm{d}\,y$$

Equation 1. Normalization formulas for metrics x_τ (τ = metric type)

The normalization formulas for metrics simply describe how to fit a multivariate gamma distribution f_τ, based on a *corpus of homogeneous data sets* (cf. [HZ95]). Each data set is a "training point" for the metric normalization function, G_τ (i.e., the shape and scale parameters of f_τ).

The above algorithm describes how to build a composite as a *specification* for supervised, multi-strategy learning on a decomposed time series, and how use it to carry out this learning. The remainder of this chapter describes the mechanisms for populating the database of learning techniques using all of the model components that are described in a composite, and how to calibrate the metrics for selecting these components.

3.2 Quantitative Theory of Metric-Based Composite Learning

This section presents the metric-based component of a new multi-strategy model selection system. Using approximate predictors of performance for each learning technique, I develop the algorithm outlined in Section 3.1.3 for selecting the learning technique most strongly *indicated by the data set characteristics*.

3.2.1 Metric-Based Model Selection

The premise that nonparametric models such as feedforward and recurrent artificial neural networks should not be applied without explicit organization is borne out by numerous studies in the literature [GBD92]. The negative consequences of such ad-hoc usage, as reflected in network complexity, convergence speed, and prediction quality, is especially evident in time series

learning [GW94, Ne96]. Prediction quality typically has a nonlinear utility in time series prediction, with the shape of this function depending upon the inferential application [He91, BDKL92, RN95, DL95]. In Chapters 4 and 5, I describe some synthetic and real-world problems in time series learning that illustrate the overhead costs of single-strategy, adaptive models versus multi-strategy model selection.

Quantitative methods for model selection allow the performance of inductive learning techniques to guide the choice of a particular technique from among many different configurations. The metrics used in *estimating* performance might be direct measurements or indirect predictors. Direct measurements include *descriptive statistics*, such as the mean and variance for prediction accuracy and the confidence intervals for each mean (using a particular technique). Greiner, for example, developed an estimation procedure for performance of an induction algorithm that is used for tuning of various learning parameters [Gr92]. The performance of two candidate induction algorithms is compared by holding a "race" between them: specifically, prediction accuracy is computed until the confidence intervals for the mean (at some specified confidence level) no longer overlap. Kohavi incorporated this procedure into attribute subset selection [Ko95]. The *DELVE* system, developed at the University of Toronto, takes a more sophisticated approach toward tracking and analyzing the long-term (and case-specific) competitive behavior of learning algorithms, also using descriptive statistics [RNH+96].

In some learning applications, time and computational resource limitations make it less feasible to collect descriptive statistics than to compute metrics that *estimate expected performance*. In coarse-grained model selection, this method of *indirect prediction* is efficient when a learning technique is one of several combinations (as is the case for the database shown in Table 1). The savings in computational work are combinatorially magnified when there are subproblems for which a model must be selected. The interaction among these subproblems at the level of the mixture model (i.e., at all *moderator network* levels, as defined in Section 3.4 and Chapter 4) means that every combination of learning technique configurations among subproblems must be tested. As Table 1 shows, there are up to 23 implemented configurations that may be tested for a *single* subproblem. Even with equality constraints on the mixture model and algorithm (the selected column), the number of combinations is an exponential function of the number of subsets, with a relatively large base. Appendix A describes this growth in more detail. There may also be insufficient computational resources to obtain *conclusive* descriptive statistics. In this case the predictive method may or may not help, depending on how

representative the training corpus is for metric normalization (see Section 3.5 and Chapter 5). That is, metrics for indirect prediction may outperform "try and see" methods if the volume of training data is relatively small compared to the size of the desired inference space (the volume of test data), *but* the metrics have been calibrated with many more representative test beds. Finally, feedback from the normalized metrics is useful as a heuristic evaluation function for attribute partition search, as documented in Section 2.4.3 and Chapter 5.

3.2.2 Model Selection for Heterogeneous Time Series

This section first surveys three types of *linear* processes [GW94, Mo94] for time series learning. Next, it presents three corresponding artificial neural network models that are specifically designed to represent each process type and the algorithms that are used to train these models. Then it surveys additional types of temporal patterns that can be efficiently expressed by temporal Bayesian network models and discusses how the same algorithms can sometimes be adapted to train them. Finally, it examines the methodology of existing mixture models and explains how the two used in my system were developed.

To model a time series as a stochastic process, one assumes that there is some mechanism that generates a random variable at each point in time. The random variables *X(t)* can be univariate or multivariate (corresponding to single and multiple attributes or *channels* of input per exemplar) and can take discrete or continuous values, and time can be either discrete or continuous. For clarity of exposition, my experiments focus on discrete classification problems with discrete time. The classification model is *generalized linear regression* [Ne96], also known as a *1-of-C coding* [Sa98] or *local coding* [KJ97].

Following the parameter estimation literature [DH73], time series learning can be defined as finding the parameters $\Theta = \{\theta_1, \ldots, \theta_n\}$ that describe the stochastic mechanism, typically by maximizing the likelihood that a set of realized or *observable* values, $\{x(t_1), x(t_2), \ldots, x(t_k)\}$, were actually generated by that mechanism. This corresponds to the backward, or maximization, step in the *expectation-maximization (EM)* algorithm [DH73]. Forecasting with time series is accomplished by calculating the conditional density $P(X(t) \mid \{\Theta, \{X(t-1), \ldots, X(t-m)\}\})$, when the stochastic mechanism and the parameters have been identified by the observable values *{x(t)}*. The order *m* of the stochastic mechanism can, in some cases, be infinite; in this case, one can only approximate the conditional density.

Despite recent developments with nonlinear models, some of the most common stochastic models used in time series learning are parametric linear models called *autoregressive (AR)*, *moving average (MA)*, and *autoregressive moving average (ARMA)* processes.

MA or moving average processes are the most straightforward to understand. First, let *{Z(t)}* be some fixed zero-mean, unit-variance "white noise" or "purely random" process (i.e., one for which $Cov[Z(t_i), Z(t_j)] = 1$ *iff* $t_i = t_j$, 0 otherwise). *X(t)* is an *MA(q)* process, or "moving average process of order *q*", if $X(t) = \sum_{\tau=0}^{q} \beta_\tau Z(t-\tau)$, where the β_τ are constants. It follows that $E[X(t)] = 0$ and $Var[X(t)] = \sum_{\tau=0}^{q} \beta_\tau$. Moving average processes are often used to describe stochastic mechanisms that have a finite, short-term, linear "memory" [Mo94, Ch96, MMR97, PL98]. The *input recurrent* network [RH98], a type of *exponential trace* memory [Mo94, MMR97], is an example of a model for *MA(1)*.

AR or autoregressive processes are processes in which the values at time *t* depend linearly on the values at previous times. With *{Z(t)}* as defined above, *X(t)* is an *AR(p)* process, or "autoregressive process of order *p*", if $\sum_{\upsilon=0}^{p} \alpha_\upsilon X(t-\upsilon) = Z(t)$, where the α_υ are constants. In this case, $E[X(t)] = 0$, but the calculation of $Var[X(t)]$ depends upon the relationship among the α_υ; in general, if $|\alpha_\upsilon| \geq 1$, then *X(t)* will quickly diverge. AR processes can be expressed by certain exponential trace memory forms (specifically, Jordan recurrent networks [Jo87, PL98]) or by *time-delay* or *tapped delay-line neural networks* (*TDNNs* [LWH90, MMR97] or *delay-space embedding* [Mo94]). They are equivalent to infinite-length *MA* processes [BD87, Ch96].

ARMA is a straightforward combination of *AR* and *MA* processes. With the above definitions, an *ARMA(p, q)* process is a stochastic process *X(t)* in which $\sum_{\upsilon=0}^{p} \alpha_\upsilon X(t-\upsilon) = \sum_{\tau=0}^{q} \beta_\tau Z(t-\tau)$, where the $\{\alpha_\upsilon, \beta_\tau\}$ are constants [Mo94, Ch96]. Because it can be shown that *AR* and *MA* are of equal expressive power, that is, because they can both represent the same linear stochastic processes (possibly with infinite *p* or *q*) [BJR94], ARMA model selection and parameter fitting should be done with specific criteria in mind. For example, it is typically appropriate to balance the roles of

the *AR(p)* and *MA(q)*, and to limit *p* and *q* to small constant values for tractability (empirically, 4 or 5) [BJR94, Ch96, PL98]. The *Gamma memory* [DP92, PL98] is an example of an *ARMA (p, q)* model.

In *heterogeneous* time series, the embedded temporal patterns belong to different categories of statistical models, such as *MA(1)* and *AR(1)*. Examples of such embedded processes are presented in the discussion of the experimental test beds in Chapter 5 and the appendices. As discussed in Section 2.2, a multiattribute time series learning problem can be decomposed into homogeneous subtasks by synthesis of attributes or by partitioning. Decomposition of time series by partitioning is applicable in multimodal sensor fusion (e.g., for medical, industrial, and military monitoring), where each group of input attributes represents the bands of information available to a sensor [SM93]. Analogously, in geospatial (map-referenced) data mining, attributes may be grouped on the basis of sensor or measurement sites (i.e., how the locations of observations are clustered). Complex attributes may be *synthesized* explicitly by constructive induction, as in causal discovery of latent (hidden) variables [He96]; or implicitly by preprocessing transforms [HR98a, RH98].

Artificial neural network architectures that correspond to ARMA, MA, and AR processes are called, respectively, *Gamma networks* (whose individual units are known as Gamma *memories*) [DP92, Mo94, MMR97, PL98], *simple recurrent networks* (*SRNs*) [Mo94, Ha95, MMR97, PL98], and *time-delay* or *tapped delay-line neural networks* (*TDNNs*) [LWH90, Ha94, Mo94, MMR97, PL98]. Note that SRNs may represent *either* nonlinear AR (Jordan-type [Jo87, PL98]) or nonlinear MA(1) (input-recurrent [Mo94, MMR97, PL98, RH98]) processes. Minsky and Papert [MP69] first discussed SRNs, but adaptations of *delta rule learning*[6] such as *backpropagation through time* (*BPTT*) were first developed by Rumelhart *et al* [RM86]. Specific architectural types were developed by Elman [El90], Jordan [Mo94], and Principé *et al* [PL98].

Other algorithms that are used to train temporal ANNs include the *Expectation-Maximization* (*EM*) algorithm [DLR77, BM94], a local optimization algorithm for probabilistic networks, and the *Metropolis* algorithm for simulated annealing [KGV83, Ne93]. Appendix B describes these algorithms, and their implementation in my system, in greater detail.

[6] A family of local, gradient-based optimization algorithms also referred to as *backpropagation* of error.

Both EM and gradient learning can be used to learn the conditional probabilities associated with parent sets in temporal Bayesian networks [Ne93] and with state transitions and output distributions in hidden Markov models [Le90, BM94]. This approximate process is known as *parameter estimation* in the statistical inference literature [GBD92, Ne96]. Appendix B also describes the adaptation of EM and gradient learning to parameter estimation in temporal Bayesian networks.

Our survey of the learning components with which to populate a database of concludes with a brief discussion of mixture models and hierarchical structure. Sections 3.4 and 3.5 and Chapter 4 provide more technical detail regarding the architecture of hierarchical mixtures (especially fusion networks) as used in this dissertation. Meanwhile, the following synopsis gives the design rationale for the organization of columns shown in Table 1. A mixture model is needed to reintegrate intermediate predictions from multiple subnetworks for each subproblem obtained by systematic decomposition. This research considers two basic designs for mixture models: bottom-up refinement to account for the differences in intermediate concepts achieved by input attribute partitioning. This corresponds to a single-pass construction whose purpose is to combine multiple "specialist" models (possibly of different types) with lower resolution capability into a more powerful model with reduced localization error. The mixture model used to implement this design is called a *specialist-moderator network* [HR98a, RH98], and it is fully documented in Chapter 4. The second category of mixture models used is the top-down "load-distributing" mixture that divides the learning problem by weighting subtrees of the hierarchy so as to force specialization of the individual subnetworks to different parts of the (possibly multimodal) overall target distribution. This corresponds to a multi-pass training procedure whose purpose is to find a "good split" of the mixture ("gating network") weights and an even distribution of the learning task among "expert" subnetworks. The mixture model used to implement this design is a variant of the *Hierarchical Mixture of Experts* (*HME*) of Jordan *et al* [JJB91, JJNH91, JJ94], which assumes identical inputs and intermediate target concepts. I relax this assumption for compatibility with the attribute partitioning and multi-strategy learning approach. Chapter 4 discusses the ramifications of this design.

3.2.3 Selecting From a Collection of Learning Components

The remainder of this section describes a novel type of metric-based model selection that selects from a known, fixed "repertoire" or "toolbox" of learning techniques. This is

implemented as a "lookup table" of *architectures* (rows) and *learning methods* (columns). In object-oriented design terms, the learning architecture corresponds to the data structures and instance variables of a class definition; the learning method (both the algorithms and mixing procedure), to the methods of this class. Each architecture and learning method has a characteristic that is positively (and uniquely, or almost uniquely) correlated with its expected performance on a time series data set. For example, naïve Bayes is most useful for temporal classification when there are many discriminatory observations (or *symptoms*) all related to the hypothetical causes (or *syndromes*) that are being considered [He91]. The strength of this characteristic is measured by an *architectural* or *distributional* metric. Each is normalized and compared against those for other (architectural or distributional) characteristics. For example, the architectural metric for temporal naïve Bayes is simply a score measuring the degree to which observed attributes are *relevant* to discrimination of every pair of hypotheses. The "winning" metrics thus identifies the dominant characteristics of a *subset* of the data (if this subset is sufficiently homogeneous to identify a single winner). These subsets are acquired by selecting *input attributes* (i.e., channels of time series data) from the original exemplar definition (cf. [KJ97]).

The metrics are called *prescriptive* because each one provides evidence in favor of an architecture or method. The design principle for prescriptive metrics is twofold. First, the goal is to derive a *normalized, quantitative* measure for each model *category* of the degree to which a training data set matches its *characteristics*. The characteristics of interest are the *memory form* [Mo94], which captures the short-term memory capabilities of a time series model. The measure should be quantitative and continuous in order to admit computation to the desired degree of precision and comparison with any other measure. Furthermore, it must be normalized in order for this comparison to be well defined. Because each metric prescribes a particular model type, there are one-to-one correspondences between architectural metrics and rows of Table 1 and between distributional metrics and columns of Table 1. Each metric should be high if and only if the memory form (for architectural metrics) or a similar unique and *learnable* characteristic (for distributional metrics) is present. According to these design criteria, model can be selected by simply accepting the model that is prescribed (or endorsed) by the highest-valued metric. This means that their *ranges* should be finite and identical.

The next section describes a database of available learning architectures and methods (mixture models and algorithms). Based on the formal characterization of these learning

techniques as time series models [GW94, Mo94, MMR97], indicator metrics can be developed for the *temporal structure* and *mixture distribution* of a *homogeneous* time series (i.e., one that *has* identifiable dominant characteristics). The highest-valued (normalized) *architectural* metric is used to select the learning architecture; the highest for *distribution* is used to select the learning method.

3.3 Learning Architectures for Time Series

For time series, we are interested in actually *identifying* a stochastic process from the training data (i.e., a process that generates the observations). The performance element, time series classification, will then apply a *model* of this process to a continuation of the input (i.e., "test" data) to generate predictions. The question I have addressed in this chapter is: "To what degree does the training data (or a restriction of that data to a subset of attributes) probabilistically match a prototype of some *known* stochastic process?" This is the purpose of metric-based model selection: to estimate the degree of match between a subset of the observed data and a known prototype. Prototypes, in this framework, are *memory forms* [Mo94], and manifest as embedded patterns *generated by the stochastic process* that the memory form describes. For example, an exponential trace memory form can express certain types of MA(1) processes. The kernel function for this process is given in Section 3.2.2. The more precisely a time series can be described in terms of exponential processes (wherein future values depend on exponential growth or decay of previous values), the more strongly it will match this memory form. The stronger this match, the better the expected performance of an MA(1) learning model, such as an input recurrent (*IR*) network. Therefore, a metric that measures this degree of match on an arbitrary time series is a useful predictor of IR network performance.

3.3.1 Architectural Components: Time Series Models

Learning Architecture	Architectural Metric
Simple recurrent network (SRN)	**Exponential trace (AR) score**
Time delay neural network (TDNN)	**Moving average (MA) score**
Gamma network	**Autoregressive moving average (ARMA) score**
Temporal naïve Bayesian network	**Relevance score**
Hidden Markov model (HMM)	**Test set perplexity**

Table 2. Learning architectures and their prescriptive metrics

Table 2 lists five learning architectures (the rows of a "lookup table") and the indicator metrics corresponding to their strengths. The principled rationale behind the design of these metrics is that each is based on an attribute chosen to *correlate positively* (and, to the extent feasible, *uniquely*) with the *characteristic memory form* of a time series. A *memory form* as defined by Mozer [Mo94] is the representation of some specific temporal pattern, such as a limited-depth buffer, exponential trace, gamma memory [PL98], or state transition model.

SRNs, TDNNs, and gamma networks are all temporal varieties of artificial neural networks (ANNs) [MMR97]. A *temporal naïve Bayesian network* is a restricted type of Bayesian network called a *global knowledge map* (as defined by Heckerman [He91]), which has two stipulations. The first is that some random variables may be temporal (e.g., they may denote the durations or rates of change of original variables). The second is that the topological structure of the Bayesian network is learned by naïve Bayes. A hidden Markov model (HMM) is a stochastic state transition diagram whose transitions are also annotated with probability distributions (over output symbols) [Le89].

3.3.2 Applicable Methods

The methods that can be used with each learning architecture are indicated in Table 1 by the symbols ✓ (denoting an existing implementation) and ★ (denoting a new implementation developed for this dissertation). Rows are integrated with columns to form a complete description of a learning technique (the part of a composite other than the problem definition). This is implemented by building a probabilistic network (temporal ANN or temporal Bayesian network) with the topology specified by the selected row, training it together with the other subnetworks, and incorporating it into the overall mixture model. Appendix B gives technical details of this implementation. Some training issues and the ramifications for the mixture model are discussed in Section 3.4.

3.3.3 Metrics for Selecting Architectures

The prototype architectural metrics for temporal ANNs are average autocorrelation values for the preprocessed data. Memory forms for temporal ANNs can be characterized using a formal mathematical definition called the kernel function. Convolution of a time series with this kernel function produces a transformed representation under its memory form [Mo94, MMR97]. The design principle behind the architectural metrics for temporal ANNs is that a memory form is

strongly indicated if the transformed time series has significantly lower uncertainty (conditional entropy) than the original series.

For example, to compute the degree of match with an *MA(1)* process, convolution of an exponential decay window (an MA(1) kernel function) is first applied [MMR97]. The decrease in entropy obtained by conditioning on this window of width *p* is then compared against that for other memory forms. This estimates the predictive power of the model if chosen as the learning architecture. The convolutional formalism and metrics for MA, AR, and ARMA processes are given in Appendix C.

The score for temporal naïve Bayesian network is the average number of variables relevant to each pair of diagnosable causes (i.e., hypotheses) [He91]. This score is computed by constructing a Bayesian network by naïve Bayes [Pe88] and then averaging a relevance measure (cf. [KJ97]) on the conditional distribution of symptoms (input attributes) versus syndromes (hypotheses). This relevance measure may be as simple as an average of the number of relevant attributes. Kohavi and John [KJ97] survey relevance measures from the literature and compares their merits for attribute subset selection. Heckerman [He91] also defines a relevance measure for Bayesian network structuring that may be useful as a prescriptive metric for temporal Bayesian network architectures.

Finally, the indicator metric for HMMs is the empirical perplexity (arithmetic mean of the branch factor) for a constructed HMM [Le89].

3.4 Learning Methods

Learning Method	Distributional Metric
HME, gradient	**Modular cross entropy**
HME, EM	**Modular cross entropy + missing data noise**
HME, MCMC	**Modular cross entropy + sample complexity**
Specialist-moderator, gradient	**Factorization size**
Specialist-moderator, EM	**Factorization size + missing data noise**
Specialist-moderator, MCMC	**Factorization size + sample complexity**

Table 3. Learning methods and their prescriptive metrics

Table 3 lists six learning methods that correspond to the columns of the database of learning techniques depicted in Table 1. These learning methods are organized under the main heading of "mixture model used" and the subheading of "training algorithm". Section 3.4.1 presents the two available mixture models, justifies their use in constructing the database, and explains how they govern both the learning architectures and the application of training algorithms. Section 3.4.2 relates the training algorithms to specific learning architectures (described in Section 3.3) and describes how this combination defines the overall learning technique. Finally, Section 3.4.3 documents how the distributional metrics are derived and how they are used to select the learning methods.

3.4.1 Mixture Models and Algorithmic Components

This section documents the mixture models that are used to organize specialized time series models (subnetworks) into a hierarchy, and their relation to training algorithms for each subnetwork.

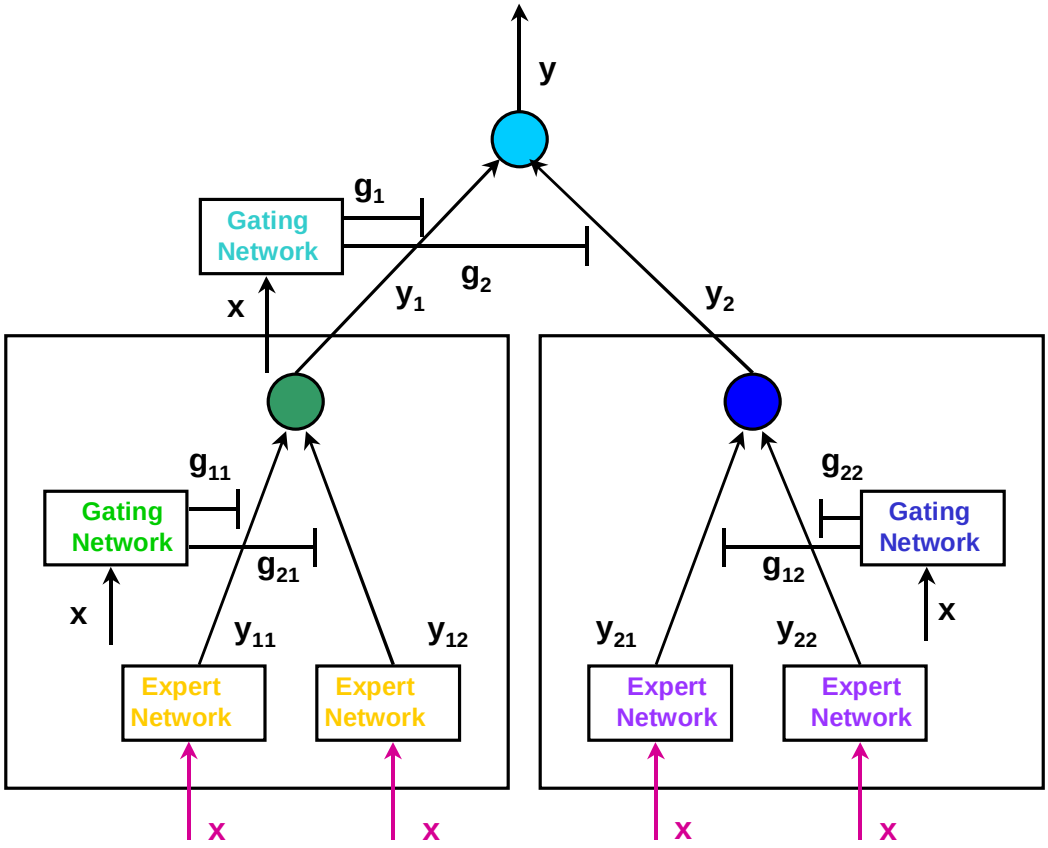

Figure 4. A Hierarchical Mixture of Experts (HME) network

A *hierarchical mixture of experts* (HME), shown in Figure 3, is a mixture model composed of generalized linear elements (as used in feedforward ANNs) [JJHN91, JJ94]. It can be trained by gradient learning, expectation-maximization [JJ94], or Markov chain Monte Carlo (MCMC) methods (i.e., random sampling as in the Metropolis algorithm for simulated annealing) [MMR97].

A *specialist-moderator network* is a new, hierarchical mixture model that can combine predictions from different learning architectures and whose components have different input and output attributes [HR98a, RH98]. Specialist-moderator networks are discussed briefly in this section and in greater detail in Chapter 4.

Figure 4 depicts a *specialist-moderator (SM)* network, which combines classifiers in a bottom-up fashion. Its primary novel contribution is an ability to learn using a hierarchy of inductive generalizers (components) while utilizing *differences among input and output attributes* in each component. These differences allow the network to form *intermediate targets* based on the learning targets of its components, yielding greater resolution capability and higher classification accuracy than a comparable non-modular network. In time series learning, this typically means reduced localization error, such as in multimodal sensor integration [HR98a, RH98]. Each component (box) in Figure 1 denotes a self-contained statistical learning model such as a multilayer perceptron, decision tree, or Bayesian network. I choose to experiment with artificial neural networks (ANNs) because the target application is time series classification, and ANNs readily admit extension to time series [El90, PL98]. The terms *specialist* or *moderator* may denote arbitrary learning models in the overall network (a tree of components), but are assumed to be ANNs here.

An SM network is constructed from a specification of input and output attributes for each of several modules (the leaves of the network). Training data and test input will be presented to these "specialists" according to this specification. The construction algorithm simply generates new input-output specifications for *moderator* networks. The target output classes of each parent are the Cartesian product (denoted $\times$) of its children's, and the children's outputs *and* the concatenation of their inputs (denoted $\mathbf{o}$) are given as input to the parent.

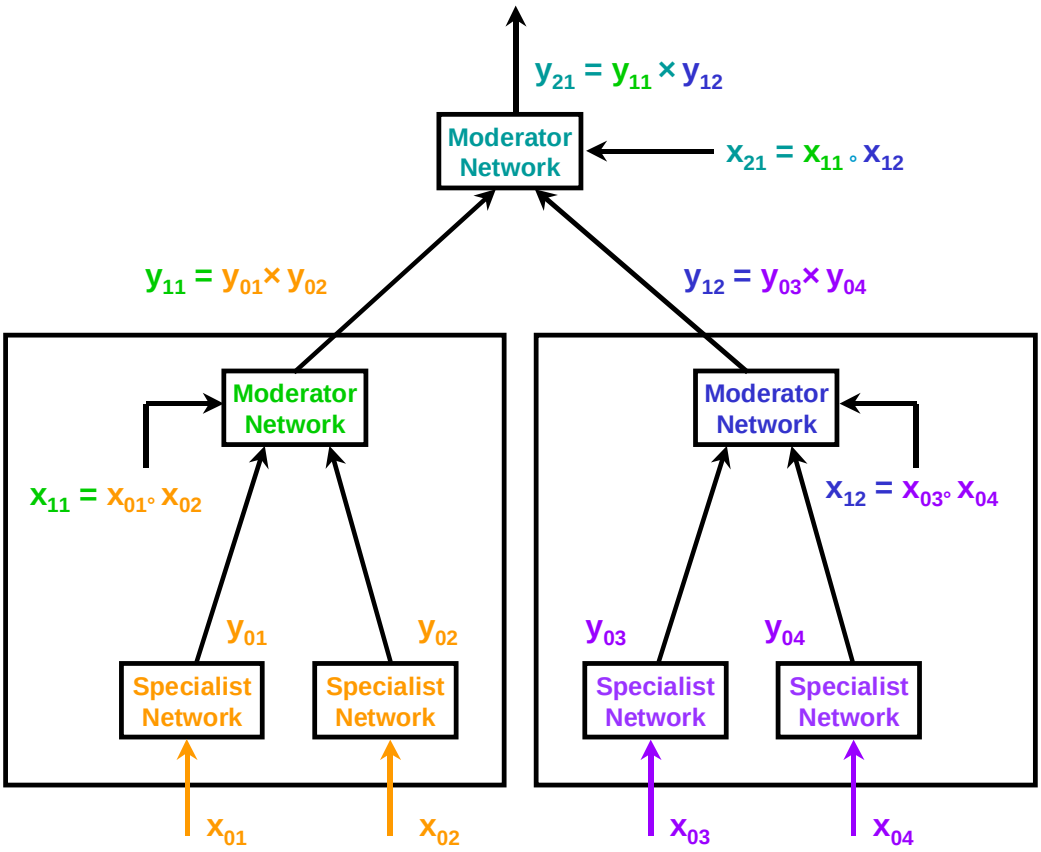

Figure 5. A Specialist-Moderator network

One significant benefit of this abstraction approach is that it exploits factorial structure (i.e., the ability of high-level or abstract learning targets to be factored) in decomposable learning tasks. This results in a reduction in network complexity compared to non-modular or non-hierarchical methods, *whenever this structure can be identified* (using prior knowledge, or more interestingly, through clustering or vector quantization methods). In addition, the bottom-up construction supports natural grouping of input attributes based on *modalities* of perception (e.g., the data *channels* or observable attributes available to each "specialist" via a particular sensor). In Chapter 5, I demonstrate that the test error achieved by a specialist-moderator network on decomposable time series learning problems is lower than that for non-modular feedforward or temporal ANN (given limits on complexity and training time).

3.4.2 Combining Architectures with Methods

The learning combination specified by combining a particular learning *architecture* and training *algorithm* determines a learning specification for a single subproblem. When taken in

the context of a particular *mixture model*, this combination corresponds to a single entry (row and column) in Table 1. Each entry, plus the definition *(A_i, B_i)* for the subproblem, is therefore equivalent to one tuple $(A_1, B_1, \theta_1, \gamma_1, S_1)$ in a composite $\mathbf{L} = ((A_1, B_1, \theta_1, \gamma_1, S_1), \ldots, (A_k, B_k, \theta_k, \gamma_k, S_k))$. All tuples together constitute **L**, which is used as a training specification for the entire partition.

Training of each subnetwork occurs concurrently. It is *independent* in the case of SM networks (that is, there is no communication of data between any specialist subnetworks) and *interleaved* in the case of HME networks (that is, information is transmitted through the gating network mechanism because each level is successively updated on every top-down pass). Thus, SM networks are fully data parallel, while HME networks require synchronization.

3.4.3 Metrics for Selecting Methods

This section outlines the metrics for selecting learning methods, which are further documented in Appendix C. It is important to note that a distributional metric is a *function of an entire partition* (as applied to a training data set), while one architectural metric is calculated *for every subset of that partition*. When compiling a composite, the same distributional metric is used with every row of the lookup table (i.e., the corresponding choice of column is identical).

The *distributional* metrics for HME networks are based on modular mutual information. Mutual information is defined as the Kullback-Leibler distance between joint and product distributions for two random variables, or, in this case, groups of them [CT91]. The conditional mutual information measure to be *maximized* is that between each subset of attributes and the desired output, given a fixed sum of the mutual information measures conditioned on all preceding subsets in some arbitrary ordering. The conditional mutual information measure to be *minimized* is the cumulative or "overlap" region [Jo97b]. This minimizes the amount of uncertainty (i.e., "work") to be performed by the gating component and tends to evenly distribute this work among all branches of the tree-structured mixture model (cf. [JJ94]). This metric is derived and fully documented in Appendix C.

The metrics for specialist-moderator networks are proportional to factorization size (the number of distinguishable equivalence classes of the overall mixture divided by the product of its components'). This metric is derived and fully documented in Appendix C.

To select a learning algorithm, gradient learning is defined as a baseline, and a term is added for the gain from estimation of missing data (by EM) [JJ94] or global optimization (by MCMC) [Ne96], adjusted for the conditional sample complexity.

3.5 Theory and Practice of Composite Learning

This section concludes the presentation of the composite learning algorithm **Select-Net** and the metric-based model selection phase. First, I list the main desiderata for composites and the metrics used to select their learning model portions. These are related to hypotheses that may be evaluated for the metrics, and for the normalization process that allows them to be compared. Second, I outline a technique for calibrating metrics based on representative data sets (corpora). Third, I list the uses of the normalized metric values in my system.

3.5.1 Properties of Composite Learning

The desired properties for all architectural and distributional metrics are as follows:

1. Each can be normalized and compared on an objective scale to any other architectural or distributional metric for an arbitrary learning problem defined on an attribute subset or partition. This suggests that the metric be quantitative and continuous-valued.

2. Each is *positively* correlated with the expected performance of the corresponding learning architecture or method. This hypothesis can be, and is, tested empirically, with the findings reported in Chapter 5 as part of the general results on composite learning.

3. Each is *uniquely* correlated (or more strongly correlated than any other metric, with a high degree of statistical confidence) with this expected performance. Again, this hypothesis can be, and is, tested as part of the evaluation experiments for composite learning.

3.5.2 Calibration of Metrics From Corpora

Calibration of model selection metrics from corpora is a well-tested method in empirical methods for speech recognition [Le89] and natural language processing. The formula for normalizing metrics in this system, given in Equation 1 of Section 3.1.3, is an application of this

method to learning from heterogeneous time series. Previously, I have successfully used a very similar approach to calibrate metrics for technique selection in heterogeneous file compression. Details are documented briefly in Appendix D, but the interested reader is referred to [HZ95] for the full explanation. The corpora used to calibrate my normalization function comprise both real-world and synthetic data. The normalization parameters that are being estimated, or learned, by this higher-order training process are the *shape* and *scale* parameters, t_τ and λ_τ, for each multivariate Gamma distribution (with 5 variables for learning architectures and 6 for learning methods).

3.5.3 Normalization and Application of Metrics

Once the metrics are properly normalized, the selection mechanism for learning architectures and methods is straightforward: it suffices to choose the row or column corresponding to the maximal normalized metric value (ties are very rare when the metrics in question do not have the maximum value on the normalized scale). The metrics, however, are also applied as feedback for partition search (and can be used in other wrapper-driven parameter tuning systems, cf. [Ko95, KJ97]).

4. Hierarchical Mixtures and Supervised Inductive Learning

Decomposition of supervised learning tasks in this dissertation entails three stages: subproblem definition, model selection for subproblems, and reintegration of trained models. This chapter examines the third and final stage, reintegration, by means of *hierarchical mixture models*. First, I present the problem of *data fusion* in composite learning, and a generic, hierarchical approach using probabilistic networks. This generic design originated concurrently with that of the learning systems for the first two stages. Second, I survey the *hierarchical mixture of experts* (*HME*), a multi-pass architecture for integration of submodels. HME supports a type of self-organization over submodels that are assumed to be identical in the original formulation. I adapt HME to multi-strategy learning from time series. Third, I survey the *specialist-moderator* (*SM*), a single-pass architecture for integration of non-identical models. The SM network was specifically designed for data fusion in decomposition of learning tasks. Fourth, I present the high-level algorithms for constructing and training hierarchical mixture models of both types using composites (specifications of subnetwork types and training algorithms). The constructions and training procedures raise some analytical issues and performance issues, which I address here. Fifth, I investigate some important properties of hierarchical mixture models that are useful in evaluating their performance empirically.

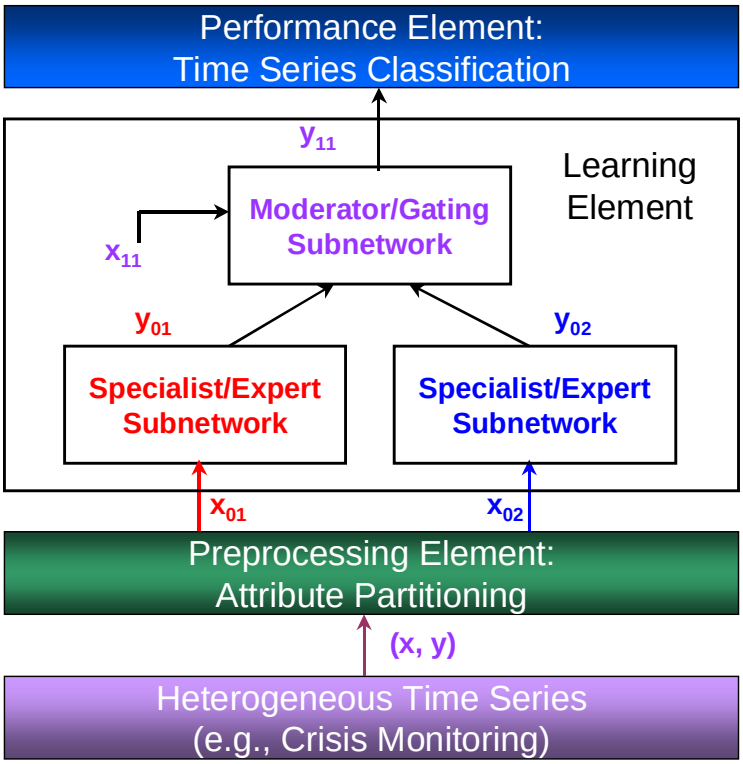

Figure 6. Role of hierarchical mixtures in decomposition of time series learning

4.1 Data Fusion and Probabilistic Network Composites

Figure 6 depicts a learning system for decomposable time series. The central element of this system is a hierarchical *mixture model* – a general architecture for combining predictions from submodels. In this dissertation, the submodels are temporal probabilistic networks such as recurrent ANNs. Attribute partitioning, described in Chapter 2, produces the subdivided inputs, x_{0n}, to these *specialist*, or *expert*, subnetworks (henceforth called *specialists* or *experts*). Unsupervised learning methods such as self-organizing feature maps (SOFMs) [Ko90] and competitive clustering [Ha94] are applied to form intermediate targets y_{0n}, also as described in Chapter 2. The subnetwork types, the algorithms used to train them, and the overall organization of the mixture model (including the types of *moderator*, or *gating*, subnetworks used), are selected from a database of components. This database and the metric-based model selection process are documented in Chapter 3. The overall concept (y_{11}) is the learning target for the top-level moderator subnetwork (henceforth called a *moderator*) in this hierarchy. Definition of inputs and outputs for specialists and moderators is the topic of this chapter.

This section presents hierarchical mixture models for reintegrating all the trained components of a composite time series model. Two kinds of mixtures are applied in this research: *specialist-moderator* (*SM*) networks and *hierarchical mixtures of experts* (*HME*). I begin by outlining the general framework of hierarchical mixture models and how *SM networks* and *HME* relate to problem decomposition and multi-strategy learning. I then explain how sensor and data fusion applications make this relationship especially important to time series learning. Finally, I discuss the role of hierarchical mixtures in my overall system, especially their benefits towards attribute-driven problem decomposition and metric-based model selection for composite learning.

4.1.1 Application of Hierarchical Mixture Models to Data Fusion

In time series analysis, the problem of combining multiple models is often driven by the *sources of data* that are being modeled. In formal terms, a source of data is any stochastic process that contributes to the observed data. For time series, we are interested in actually *identifying*, from training data, the best probabilistic match to a prototype of some known stochastic process. This is the purpose of metric-based model selection, where each "known process" has its own architectural metric and prescribed learning architecture. Traditionally, domain knowledge about the sources of data is used in their decomposition [HR98a, HGL+98]. This is discussed in Section 1.1; examples of heterogeneous time series with multiple data sources include multimodal sensor integration (sensor fusion) and multimodal HCI. Chapter 2

describes a knowledge-free approach that can be applied when such information is not available, but the learning problem is decomposable.

A *mixture model* is one that combines the outputs of a finite set of subordinate models by weighted averaging [Ha94]. The weights are referred to as *mixing proportions* [Ha94], *mixing coefficients*, *gating coefficients* [JJ94], or simply "weights". Traditionally, a mixture model is formally defined as a probability density function (pdf), *f*, that is the sum of weighted contributions from subordinate models:

$$f(\mathbf{y};\boldsymbol{\theta},\boldsymbol{\pi}) = \sum_{n=1}^{N} \pi_n f_n(\mathbf{y};\boldsymbol{\theta})$$

$$\text{where } \sum_{n=1}^{N} \pi_n = 1 \quad \text{and} \quad \pi_n \geq 0 \quad \text{for all } n$$

f_n are the individual pdfs for mixture components, drawn from populations S_n, $1 \leq n \leq N$, and f is a pdf over samples $\mathbf{y}$ drawn uniformly from S. That is, f_n denotes the likelihood that S_i contributes $\mathbf{y}$ to the mixture S. π_i denotes the *normalized weight* for this likelihood [Ha94]. The parameters $\boldsymbol{\theta}$ include all unknowns in the model upon which the distributions f_n are to be conditioned (i.e., the internal parameters of the subordinate models). As I explain below, this generalizes over all of the mutable parameters in the learning architecture, such as network weights and biases. The hyperparameters $\boldsymbol{\pi}$ are simply the mixing coefficients. The mixture modeling problem is to fit $\boldsymbol{\pi}$, given training data $(\mathbf{y_1}, \ldots, \mathbf{y_n}, \mathbf{y})$. An alternative definition [JJ94] that is more familiar to the nomenclature of connectionist (probabilistic network) learning is to estimate the distribution of $\mathbf{y}$ as a weighted sum of predictions $\mathbf{y}_n$ (the outputs of expert submodels, henceforth referred to as "experts"):

$$\mathbf{y} = \sum_{n=1}^{N} \pi_n \mathbf{y}_n$$

$$\text{where } \sum_{n=1}^{N} \pi_n = 1 \quad \text{and} \quad \pi_n \geq 0 \quad \text{for all } n$$

π_i still denotes the *normalized weight* for a likelihood function over samples from a population, but we have now specified that the *estimator* for the likelihood function is the output of an expert. As Jordan and Jacobs [JJ94] and Haykin [Ha94] note, experts may be arbitrary learning components. For example, Haykin specifically considers experts that are *rule generators* or arbitrary probabilistic network regression models, with real-valued, discrete, or 1-of-C

("locally") coded targets [KJ97, Sa98]. In this dissertation, I have considered only discrete (including binary) and 1-of-C-coded classification.

Finally, an even more flexible formulation of mixture models is as *a hierarchical mixture network*, whose vertices all represent subnetworks. The leaves are experts or specialist networks; the internal vertices, gating or moderator subnetworks. The target distribution $f(\mathbf{y})$ is thus described as a parameter estimation problem, where the submodel parameters θ belong to probabilistic networks such as feedforward or recurrent ANNs. For feedforward ANNs (also called *multilayer perceptrons*), the mixture model description is:

$$\mathbf{y} = f\left(\mathbf{y}^n\right), \quad 1 \le n \le N$$

where

$$f_k\left(\mathbf{y}^n\right) = b_k + \sigma_k\left(\sum_{j=1}^{H} v_{jk}\, h_j\left(\mathbf{y}^n\right)\right) \quad 1 \le k \le O$$

$$h_j\left(\mathbf{y}^n\right) = a_j + \sigma_j\left(\sum_{i=1}^{I} u_{ij}\, y_i^n\right) \quad 1 \le j \le H$$

$$\mathbf{y}^n = f^n\left(\mathbf{x}^n\right)$$

$$f_k^n\left(\mathbf{x}^n\right) = b_k^n + \sigma_k^n\left(\sum_{j=1}^{H^n} v_{jk}^n\, h_j^n\left(\mathbf{x}^n\right)\right) \quad 1 \le k \le O^n$$

$$h_j^n\left(\mathbf{x}^n\right) = a_j^n + \sigma_j^n\left(\sum_{i=1}^{I^n} u_{ij}^n\, x_i^n\right) \quad 1 \le j \le H^n$$

f_k and h_j denote outputs from the output and hidden layers, respectively, of a multilayer perceptron. It is the overall output f_k that we seek to estimate. σ denotes a transfer function (from the hidden to the output layer for σ_k; from the input to the hidden layer for σ_j). u and v denote ANN weights; a and b, ANN biases. The size of each network layer in units is denoted I, H, or O, for input, hidden, and output layers, respectively. Finally, the superscript n over any function, parameter, or size variable indicates that it belongs to expert or specialist n. In some ways, this characterization of a mixture model is more specific than the previous two (it restricts the learning architecture to a probabilistic network – a feedforward ANN in the above mathematical definition). It is, however, also more *general*, because it permits a general, possibly nonlinear, form for the fusion mechanism (rather than forcing f to be a linear combination of f_i values).

The problem of data fusion, in the context of composite learning for time series, can naturally be interpreted as one of mixture modeling. Each expert is a probabilistic network trained on some intermediate target concept, which was formed by attribute partitioning and problem redefinition (e.g., competitive clustering). This unsupervised learning phase is described in Chapter 2. The particular expert (architecture and training) used is determined using metric-based model selection on the resulting subproblem definition and a database of available components. The overall organization of the mixture model is also selected by metric-based analysis. The entire analytical step is described in Chapter 3. The selected experts are trained, resulting in classifiers that map subsets of the input to intermediate predictions. In time series learning, training data is typically a sequence of *historical observations* of the data, and the predictions are made on a *continuation* of this input [GW94]. Section 1.1 and Chapter 5 describe synthetic and real-world experiments on time series prediction using this paradigm. Both "laboratory" applications (where continuations are simulated or have been previously collected and buffered) and "field" applications (where the new input data is collected online, and learning occurs during this collection)[7] conform to the paradigm [GW94]. This dissertation considers both modes of time series analysis but focuses on learning in the offline mode and application of the performance element (classification for *diagnostic monitoring* and prediction) in the online mode.

4.1.2 Combining Classifiers for Decomposable Time Series

To understand how mixture models are used in inductive concept learning from time series, let us interpret $\mathbf{y}_i$ as a 1-of-C-coded output vector from each of n classifiers. These may be decision structures (trees, lists, regression splines, etc.), genetic classifiers, or – in the scope of this research – Bayesian and artificial neural networks. We wish to weight these classifiers to produce the target prediction $\mathbf{y}$, an overall classification for the observed inputs. In general concept learning by mixture models, all inputs $\mathbf{x}$ are presented to each subnetwork during training, and the trained network is treated as the classifier f_i. Based upon the attribute-driven problem decomposition method of Chapter 2, I emend this to *subsets* of $\mathbf{x}$. This is depicted as $\mathbf{x}_{0n}$ in Figure 6. The outputs of specialists are *predicted* values, $\hat{\mathbf{y}}_{0n}$ of $\mathbf{y}_{0n}$ (which denote the actual values, or desired output). These $\hat{\mathbf{y}}_{0n}$ values are passed on as input to moderators at the next level, as depicted in Figure 6.

[7] This is also known as *situated* [RN95], *lifelong* [Th96], or *in vivo* [WS97] learning.

The complete input for a moderator includes (at least) the inputs to all of its descendants and the predictions of its children. It is important to note that, although the specialists are trained concurrently, they are not necessarily trained *independently*; specifically, HME is a multipass algorithm that iteratively updates the weights of gating and expert subnetworks. The classifier produced by this interleaved training algorithm still operates in similar fashion (a single, bottom-up pass) in the performance element (also referred to as *recall mode* in the ANN literature) [PL98]. SM networks, by contrast, train subnetworks in a single bottom-up pass, so that all specialists or intermediate-level moderators are used to generate predictions $\hat{\mathbf{y}}_{ln}$ only once.

The mixture model thus refers to the function that maps mixture components to overall outputs. In connectionist (probabilistic network) learning, however, this term is informally extended to include the learning algorithm for weights. As the third definition shows, the subordinate models (including moderators) may be self-contained learning architectures.

Thus, when I refer to "combining models", three definitions (the first two informal; the third, formal) apply:

1. **Combining subnetworks**. The mixture model expresses weights on the outputs of each subnetwork *on each exemplar, after training.*
2. **Classifiers**. A classifier is the performance element for which a probabilistic network can acquire knowledge, in the form of decision structures, rules, etc. In the heuristic classification framework of this dissertation, a classifier is fully described by the values of network parameters after training. The mixture model expresses weights on the outputs of each classifier *on each exemplar*.
3. **Predictions**. The mixture model expresses weights on predictions of the model on each exemplar. The predictions are the elements of the subpopulations for which mixing coefficients are being estimated relative to an overall distribution.

The next two sections define *partitioning* and *aggregation* mixtures, two general types of mixture models that are exemplified by HME and SM networks.

4.2 Composite Learning with Hierarchical Mixtures of Experts (HME)

This section presents the HME architecture and discusses its adaptation to multi-strategy learning, as an *integrative* method. HME is one of two mixture models that may be selected in

my composite learning system. I review existing learning procedures for HME, consider how alternative optimization techniques (such as Markov chain Monte Carlo algorithms for Bayesian learning [Ne96, Jo97a]) may be applied, and discuss the incorporation of these methods into the "repertoire" of learning techniques presented in Chapter 3.

4.2.1 Adaptation of HME to Multi-strategy Learning

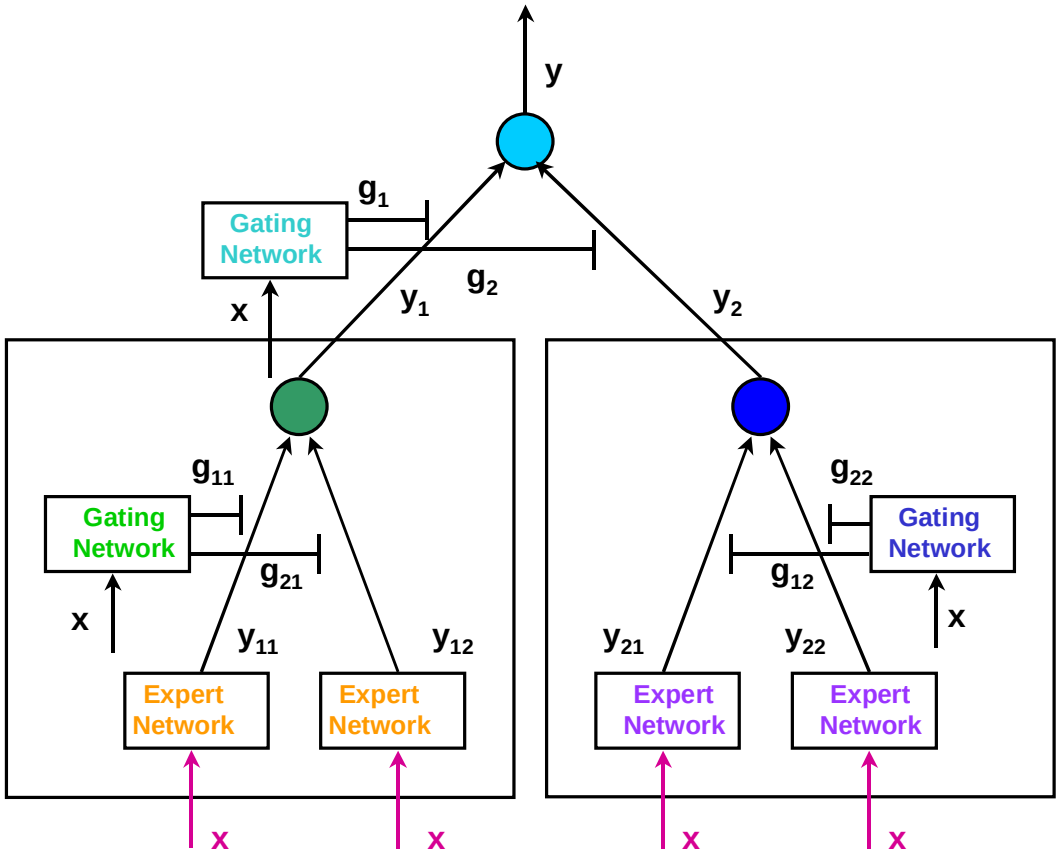

Figure 7. A Hierarchical Mixture of Experts (HME) network

Figure 7 shows an HME network of height 2, with branch factor 2 (i.e., 4 expert networks at the leaves). Note that the expert and gating networks all receive the same input x. Equally important, the target output values y_{lj}, for level l and (gating or expert) network j, are also identical.

Traditional HME uses a tree-structured network of *generalized linear models* (GLIMs), or fixed, continuous nonlinear functions with linear parameters [MN83]. The class of GLIMs includes single layer perceptrons with linear, sigmoidal, and piecewise linear transfer (activation) functions; these implement regression models, binary classification models, and hazard models

for survival analysis, respectively [JJ94, Ne96]. GLIMs of the type listed above are used at the leaves of the network as *expert* subnetworks in the mixture. The mixing is implemented by *gating* GLIMs that combine expert network outputs.

In place of GLIMs, I use general feedforward networks with nonlinear (sigmoidal or hyperbolic tangent) or piecewise linear input-to-hidden layer transfer functions and linear hidden-to-output layer transfer functions. The purpose of this modification is to permit an arbitrary fusion function to be learned. I will refer to this function, which is implemented by all of the interior (moderator) subnetworks as a whole, as a *mixture function.* As Kohavi *et al* [KSD96] point out, however, mixture functions that are *not* linear combinations of the input (i.e., those that do not have the same mixing coefficients for any input data) are semantically obscure. Furthermore, the real issue is not the ability to fit a mixture perfectly, because (just as in general concept learning) it is always possible to learn by rote if there are sufficient model resources. The true criterion is *generalization* quality. In general concept learning as well as mixture modeling, we can evaluate generalization by means of cross validation methods [Ri88, Wo92]. The upshot of these considerations is that some discretion is essential when we undertake to use a general mixture function instead of a linear gating or fusion model.

Finally, in order to adapt HME to *decomposition* of time series learning problems, it is necessary to commit a crucial change to the construction. Specifically, the inputs to experts (at the leaves of the tree-structured network) are *nonidentical*. Chapter 2 describes attribute partitioning algorithms that split the input data along "columns". Each expert receives the data corresponding to a single subset of input attributes (i.e., the columns or channels specified by that subset), and each gating network receives as inputs the concatenation of inputs to each expert and the normalized output from each expert. The target outputs at every expert and gating level are identical to one another and to the overall target. In the current implementation, only SM networks use the reformulated targets from clustering; Chapter 5, however, documents experiments with both clustered and non-clustered intermediate concepts. Thus, a training exemplar is a set of subnetwork outputs, concatenated with the total input to all experts in the *domain* of the moderator (i.e., the subtree rooted at that moderator). Training a moderator means revising its internal weights to approximate a mixture function.

4.2.2 Learning Procedures for Multi-strategy HME

HME networks are trained using an *interleaved* update algorithm that computes the error function at the topmost gating network and propagates credit down through the hierarchy, on every top-down pass (a single training *epoch*). This generic procedure can be specialized to expectation-maximization (EM), gradient, and Markov chain Monte Carlo (MCMC) learning algorithms.

Jordan and Jacobs studied EM-based learning in HME networks [JJ94]. The algorithm I use to implement learning techniques under the "HME, EM" column of my database is partly based the one described this paper. Gradient learning in HME is based on backpropagation of error using a hierarchical variant of the delta rule described in Appendix B; the probabilistic interpretation follows Bourlard and Morgan [BM94]. Finally, the MCMC method I use to perform Bayesian learning is the Metropolis algorithm for simulated annealing. This global optimization algorithm and its integration with hierarchical mixture models is also documented in Appendix B.

4.3 Composite Learning with Specialist-Moderator (SM) Networks

This section presents the SM network architecture and discusses its adaptation to multi-strategy learning, as an *integrative* method. The SM network, which was developed by Ray and Hsu [RH98, HR98a], is one of two mixture models that may be selected in my composite learning system. I review the construction of SM networks and, as for HME, consider how alternative optimization techniques such as MCMC methods may be applied, and discuss the incorporation of these methods into my database of models.

4.3.1 Adaptation of SM Networks to Multi-strategy Learning

Figure 8 shows an SM network with two layers of moderators. The primary novel contribution is the model's ability to turn attribute-based learning task decomposition to advantage in three ways:

1. **Reduced variance**. On decomposable time series learning problems, SM networks exhibit lower classification error than non-modular networks of comparable complexity. Chapter 5

reports results that demonstrate this in a manner very similar to that of Rueckl [RCK89] and Jordan *et al* [JJB91].

2. **Reduced computational complexity.** *Given* a target criterion (desired upper bound on classification error), SM networks require fewer trainable weights and fewer training cycles to achieve convergence on decomposable problems.
3. **Facility for multi-strategy learning**. My experimental results on several test beds, both real and synthetic, show that gains can be realized using specialists of *different types* within the SM network. These specialists are selected from the database documented in Chapter 3. The experimental findings are presented in Chapter 5.

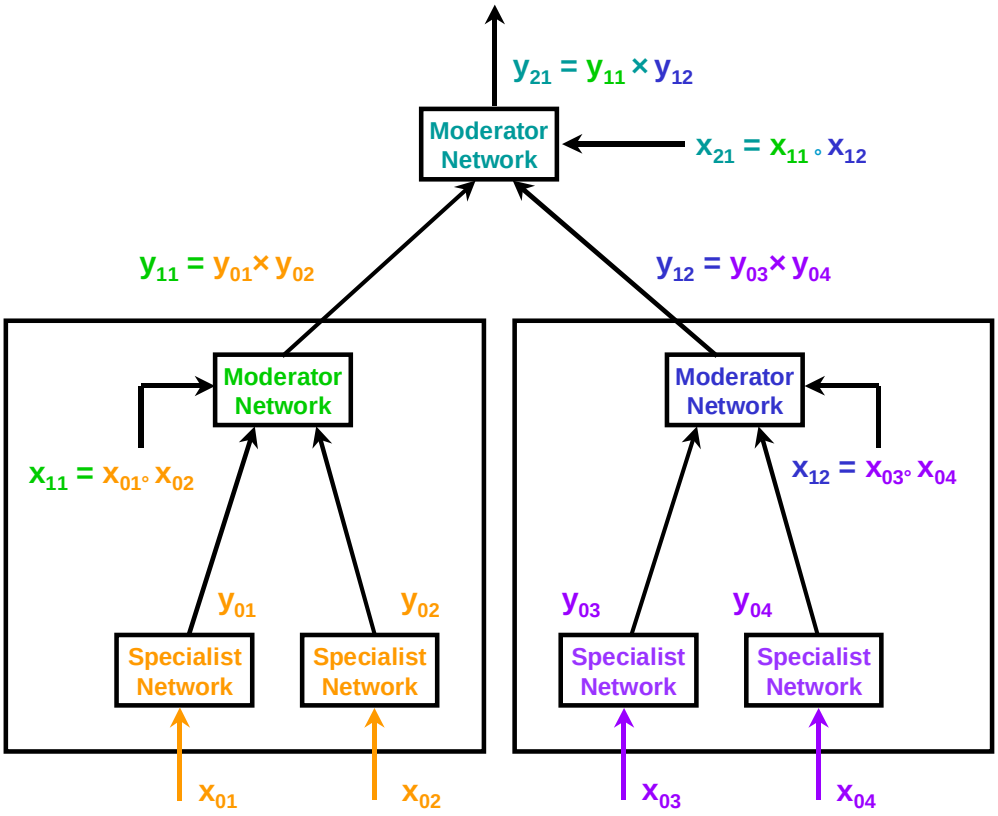

Figure 8. A Specialist-Moderator (SM) network

The main practical distinction between SM networks and HME are the ways in which each one achieves reduced variance and reduced computational complexity. SM networks trade more rapid *growth* in complexity for increased *resolution capability* and reduction of *localization error*. By exploiting differences among the problem definitions for each subnetwork, an SM network can distinguish among more concepts than its components, and to achieve higher classification accuracy than a comparable non-modular network. In time series learning

applications such as multimodal sensor integration, this localization error may be reduced in space or time [JJB91, SM93].

The construction of SM networks allows arbitrary real inputs to the expert (specialist) networks at the leaves of the mixture tree, but constructs higher level input and output attributes based upon x_{0j} (see Figure 6). This is one of the two main differences between SM networks and HME. The other is that training of the specialist-moderator network proceeds in a single bottom-up pass (i.e., a preorder traversal), while HME networks are trained iteratively, in a top-down fashion (i.e., one post-order traversal *per training cycle*, during the M step of EM). These algorithms can also be considered as proceeding in a breadth-first order: bottom-up for the specialist-moderator network, multiple up/down (estimation/maximization) passes for HME [JJ94]. The construction algorithm for SM networks is as follows:

Notation

D	training set	$\boldsymbol{a}_{lj}^{S}$	specialist part of input j at l
A	set of input attributes	$\boldsymbol{a}_{lj}^{M}$	moderator part of input j at l
B	set of output attributes	$\boldsymbol{B}_l$	complex output attribute vector at l
F	constructive induction algorithm	$\boldsymbol{b}_{lj}$	complex output attribute j at l
n	number of exemplars	x_{lj}	reformulation of $(\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n)})$ by $\boldsymbol{a}_{lj}$ (a set of vectors)
$\boldsymbol{x}^{(i)}$	input vector i		
$\boldsymbol{y}^{(i)}$	output vector i	y_{lj}	reformulation of $(\boldsymbol{y}^{(1)}, ..., \boldsymbol{y}^{(n)})$ by $\boldsymbol{b}_{lj}$
$x_j^{(i)}$	value j in input vector i	f_{lj}	S-M network j at level l
$y_j^{(i)}$	value j in output vector i	N_{lj}	number of children of f_{lj}
I	number of input channels	I_{lj}	number of inputs to f_{lj}
O	number of output channels	O_{lj}	number of outputs from f_{lj}
H	height of the S-M tree	$S_l[j]$	start index for children of moderator f_{lj}
N_l	number of networks at level l		
$\boldsymbol{A}_l$	complex input attribute vector at l	$E_l[j]$	end index for children of f_{lj}
$\boldsymbol{a}_{lj}$	complex input attribute j at l		

Given:

1. A training set $D = ((\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}))$ with input attributes $\boldsymbol{A} = (a_1, \ldots, a_I)$ such that $\boldsymbol{x}^{(i)} = (x_1^{(i)}, \ldots, x_I^{(i)})$
2. A constructive induction algorithm F such that $F(A, B, D) = (\boldsymbol{A_0}, \boldsymbol{B_0})$
3. N_{lj} for $1 \le l \le H, 1 \le j \le N_l$ (precomputed by dynamic programming)

Algorithm *SM-net:*

Using F on A and D, derive $\boldsymbol{A_0} = (\boldsymbol{a_{01}, \ldots, a_{0N_0}})$ and $\boldsymbol{B_0} = (\boldsymbol{b_{01}, \ldots, b_{0N_0}})$.

Let $(\boldsymbol{x_0^{(1)}, \ldots, x_0^{(n)}})$ be the new representation of the input data under $\boldsymbol{A_0}$, where $\boldsymbol{x_{0j}^{(i)}}: \boldsymbol{x^{(i)}} :: \boldsymbol{a_{0j}} : A$

Let $(\boldsymbol{y_0^{(1)}, \ldots, y_0^{(n)}})$ be the new representation of the output data under $\boldsymbol{B_0}$, where $\boldsymbol{y_{0j}^{(i)}}: \boldsymbol{y^{(i)}} :: \boldsymbol{b_{0j}} : B$

Train networks $(f_{01}, \ldots, f_{0N_0})$, using training inputs $\boldsymbol{x_{0j}^{(i)}}$ and desired outputs $\boldsymbol{y_{0j}^{(i)}}$ for specialist f_{0j}

for l := 1 to H // bottom-up

  for j := 1 to N_l

    $S_l[j]$ $:= \sum_{k=1}^{j-1} N_{lk} + 1$ // start index

    $E_l[j]$ $:= S_l[j] + N_{lj} - 1$ // end index

    $\boldsymbol{a_{lj}^S}$:= NULL // NULL $\circ$ $\boldsymbol{a_{l-1,k}} = \boldsymbol{a_{l-1,k}}$

    $\boldsymbol{x_{lj}^S}$:= NULL // NULL $\circ$ $\boldsymbol{x_{l-1,k}} = \boldsymbol{x_{l-1,k}}$

    $\boldsymbol{a_{lj}^M}$:= NULL // NULL $\circ$ $\boldsymbol{b_{l-1,k}} = \boldsymbol{b_{l-1,k}}$

    $\boldsymbol{x_{lj}^M}$:= NULL // NULL $\circ$ $\boldsymbol{y_{l-1,k}} = \boldsymbol{y_{l-1,k}}$

    $\boldsymbol{b_{lj}}$:= UNIT // UNIT $\times$ $\boldsymbol{b_{l-1,k}} = \boldsymbol{b_{l-1,k}}$

    $\boldsymbol{y_{lj}}$:= UNIT // UNIT $\times$ $\boldsymbol{y_{l-1,k}} = \boldsymbol{y_{l-1,k}}$

    for k := $S_l[j]$ to $E_l[j]$

      $\boldsymbol{a_{lj}^S}$:= $\boldsymbol{a_{lj}^S} \circ \boldsymbol{b_{l-1,k}}$ // specialist part

      $\boldsymbol{a_{lj}^M}$:= $\boldsymbol{a_{lj}^M} \circ \boldsymbol{a_{l-1,k}^M}$ // moderator part

      $\boldsymbol{b_{lj}}$:= $\boldsymbol{b_{lj}} \times \boldsymbol{b_{l-1,k}}$ // new output attribute

      for i := 1 to n

        $\boldsymbol{x_{lj}^{(i)\,S}}$:= $\boldsymbol{x_{lj}^{(i)\,S}} \circ \boldsymbol{y_{l-1,k}^{(i)}}$ // new input (S)

        $\boldsymbol{x_{lj}^{(i)\,M}}$:= $\boldsymbol{x_{lj}^{(i)\,M}} \circ \boldsymbol{x_{l-1,k}^{(i)\,M}}$ // new input (M)

        $\boldsymbol{y_{lj}^{(i)}}$:= $\boldsymbol{y_{lj}^{(i)}} \times \boldsymbol{y_{l-1,k}^{(i)}}$ // new output

      $\boldsymbol{a_{lj}}$:= $\boldsymbol{a_{lj}^S} \circ \boldsymbol{a_{lj}^M}$ // new input attribute

      for i := 1 to n

        $\boldsymbol{x_{lj}^{(i)}}$:= $\boldsymbol{x_{lj}^{(i)\,S}} \circ \boldsymbol{x_{lj}^{(i)\,M}}$ // new input

Train moderator network f_{lj} with

$x_{lj} = (\boldsymbol{x}_{lj}^{(1)}, \ldots, \boldsymbol{x}_{lj}^{(n)})$ and

$y_{lj} = (\boldsymbol{y}_{lj}^{(1)}, \ldots, \boldsymbol{y}_{lj}^{(n)})$

return $(f_{01}, \ldots, f_{0N_0}, \ldots, f_{H1})$

SM-net thus produces networks such as the one depicted in Figure 8. Network complexity is measured in the number of weights among generalized linear units in an ANN. **SM-net** produces moderator networks whose worst-case complexity is the product of that of their children. This growth is limited, however, because the tree height and maximum branch factor are typically (very) small constants. Relevant details of this combinatorial analysis appear in Appendix A.

Two determinants of the performance of an SM network are: the empirical likelihood of finding efficient factorizations of a data set *D*; and the difficulty of learning this factorization. The first quantity depends on many issues, the most important being the quality of *F*, the constructive induction algorithm. I consider the case where a good B_0 is already known or can be found by unsupervised learning methods, including knowledge-based constructive induction [Be90, Do96] and attribute-driven problem reformulation (e.g., subset selection [Ko95, KJ97] and partitioning). To address the second issue experimentally, I demonstrate that the achievable test error on efficient factorizations learned with an SM network is lower than that of non-modular feedforward or temporal ANNs (of comparable complexity) trained with the original data.

4.3.2 Learning Procedures for Multi-strategy SM Networks

SM networks are trained using a single-pass algorithm for updates in the overall network. That is, many training cycles or individual epochs (batch updates for backpropagation, EM steps, or candidate state transitions in MCMC learning) occur in order to complete the training for one subnetwork.

Gradient learning in SM networks was introduced in [RH98] and [HR98a]. The algorithm I use to implement learning techniques under the "SM, gradient" column of my database is based on this one. As does HME, SM also admits EM and MCMC learning for certain specialist architectures. Appendix B gives technical details of these implementations; Chapter 5, some important experimental results using these implementations.

4.4 Learning System Integration

The overall design of both hierarchical mixture models I have addressed here (modified HME and SM networks) is the result of a concurrent engineering process. The data fusion methodology complements the first two phases of my learning system – problem reformulation and multiple model selection. Based on an attribute-driven decomposition of a given time series learning problem, the mixture model must *distribute the workload* in a manner most appropriate to the characteristics of the data (as reflected in the way the problem was divided). Model selection not only chooses the specialist or expert network types independently for each subproblem, but also selects the most appropriate type of mixture model and training algorithm for the entire problem (i.e., all specialist and moderator subnetworks) as a function of the whole partition. It also provides feedback for the partition search, in order to limit the number of mixture models that are applied and eliminate the "bad splits" that do not balance the work evenly enough across mixture components. This section addresses the definition and utilization of "good splits" and the recognition of bad ones.

4.4.1 Interaction among Subproblems in Data Fusion

The objective of mixture modeling, according to Section 4.3.1, is to reduce variance and computational complexity and to facilitate multi-strategy learning. In order to reduce both variance (classification error) *and* complexity (required convergence time and network complexity), a reformulation of the problem must be exploited [Mi80, Be90, Hr92]. The solution I present through the modified HME and SM algorithms is to distribute the workload by maximizing the computational gain from specialists or experts (i.e., doing more of the work of learning at the lowest levels). This automatically reduces the difficulty of the *mixture estimation* task [DH73, CKS+88, JJ94]. The cost of this improvement is that the interaction among mixture components must be modeled. This is discussed in Section 3.4 and Appendix C.

4.4.2 Predicting Integrated Performance

Section 3.4.3 documents a combinatorial measure for *factorial* interaction and an information theoretic measure for probabilistic interaction that give rise, respectively, to the prescriptive metrics for SM networks and HME. In order to estimate the overall performace for a mixture model on an entire partition (without knowing in advance what the specialist and

moderator types are), these metrics must account for the precise mode of interaction that is exploited by the mixture. Appendix C documents the derivation of these distributional metrics.

4.5 Properties of Hierarchical Mixture Models

This section concludes the presentation of the HME and SM networks and the data fusion phase of composite learning. First, I list the criteria for network complexity and explain its relevance to performance evaluation as documented in Chapter 5. Second, I discuss how variance reduction is achieved in hierarchical mixtures and how this can be evaluated.

4.5.1 Network Complexity

To test the hypothesis that hierarchical mixtures reduce network complexity for decomposable learning problems, I first define a measure of complexity and identify other figures of merit for learning performance. These shall be held constant relative to network complexity (or vice versa). ANN, HMM, and Bayesian network complexity can all be defined in (slightly different) terms of graph complexity: that is, the number of trainable weights. The simplest measure for ANNs is the total number of connections between successive hidden layers (bipartite graph size in edges), plus the sizes of layers with bias parameters (bipartite graph size in vertices). For Bayesian networks, complexity grows exponentially with the number of values of an attribute as a base and the number of parents for a vertex (denoting a random variable) as an exponent. Network complexity is just one measure of performance. Classification accuracy on test input is, of course, an essential standard, and convergence time and number of exemplars needed to reach the target accuracy is also important in situated learning. My first method for evaluating the performance of a model is to set a target classification accuracy and compile the learning curves [Ka95] (accuracy versus training cycles for different numbers of exemplars) and complexity curves (accuracy versus training cycles for different numbers of trainable parameters). An alternative, when the achieved accuracies for the models being compared are far apart, is to plot them given fixed network complexity and training time.

4.5.2 Variance Reduction

Variance reduction in mixture modeling is achieved by combining multiple classifiers. Recent research has shown how inductive learning algorithms can be augmented by *aggregation mixtures* such as bootstrap aggregation (or *bagging*) [Br96], *stacking* [Wo92], and SM networks [HR98a, RH98], and by *partitioning mixtures* such as *boosting* [FS96] and HME [JJ94].

Aggregation uses (independent) differences in sampled input as given to each expert to estimate a mixture (by voting in bagging; by a mixture function in stacking). Partitioning mixtures are multi-pass and adjust the sample weights during learning.

5. Experimental Evaluation and Results

This chapter documents the evaluation of the time series learning system through experiments on both synthetic and real-world data sets. First, I present a learning test bed (wide-area crop condition monitoring) that demonstrates how time series can be heterogeneous, and how a hierarchical mixture model can be used to improve learning. This finding leads to further experiments on model integration and decomposition of learning tasks by attribute-driven constructive induction. Second, I report on experiments with synthetic data sets (corpora) that test the effectiveness of metrics for model selection. These synthetic corpora and two real-world corpora are used to calibrate the normalization model for metrics. Third, I document improvements to classification accuracy and learning efficiency based on attribute partitioning. I first report on performance gains as achieved through exhaustive enumeration of partitions – then examine the tradeoff between accuracy and efficiency when heuristic search is used. Fourth, I compare the performance of the integrated learning system to that of other mixture models and non-modular inductive learning techniques.

5.1 Hierarchical Mixtures and Decomposition of Learning Tasks

This section presents the results of experiments using hierarchical mixture models on time series with varying degrees of decomposability (see Section 1.4.3).

5.1.1 Proof-of-Concept: Multiple Models for Heterogeneous Time Series

The experiments first used to demonstrate heterogeneity in time series for this research were conducted using a real-world data set called the *corn condition monitoring* test bed. This test bed was developed specifically to demonstrate non-Markovity and heterogeneity in time series [HR98a, HGL+98].

Figure 9 depicts an (atemporal) spatially referenced data set for diagnosis in *precision agriculture*. The inputs are: yield monitor data, crop type, elevation data and crop management records; the learning target, *cause of observed low yield* (e.g., drought). Such classifiers may be used in *recommender* systems [RV97] (also called *normative* expert systems [He91]) to provide decision support for crop production planning in subsequent years. I collected biweekly remote sensing images and meteorological, hydrological, and crop-specific data for learning to classify

influents of *expected crop quality* (per farm) as *climatic* (drought, frost, etc.) or *non-climatic* (due to crop management decisions).

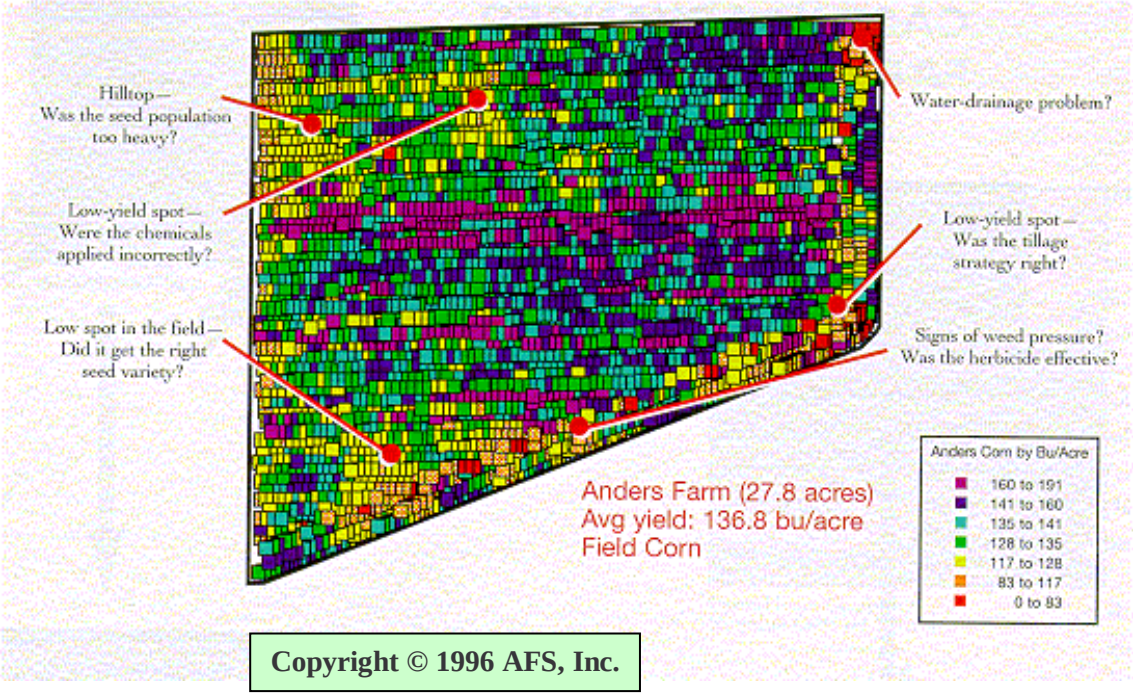

Figure 9. An agricultural decision support expert system

Figure 10 contains bar charts of the mean squared error from 125 training runs using ANNs of different configurations (5 architectures, 5 delay constant or momentum values for gradient learning, and 5 averaged runs per combination). On all runs, Jordan recurrent networks and time-delay neural networks failed to converge with momentum of 0.99, so the corresponding bars are omitted. Cross validation results indicate that overtraining on this data set is minimal. As a preliminary study, I used a gamma network to select the correct classifier (if any) for each exemplar from among the two best overall networks (input recurrent with momentum of 0.9 and TDNN with momentum of 0.7). The error rate was reduced by almost half, indicating that even with identical inputs and targets, a simple mixture model could reduce variance. These results are depicted in Figures 11 and 12.

This experiment illustrates the usefulness of learning task decomposition over heterogeneous time series. The improved learning results due to application of multiple models (TDNN and IR specialists) and a mixture model (the Gamma network moderator). Reports from the literature on common statistical models for time series [BJR94, GW94, Ne96] and experience with the (highly

heterogeneous) test bed domains documented here bears out the idea that "fitting the right tool to each job" is critical.

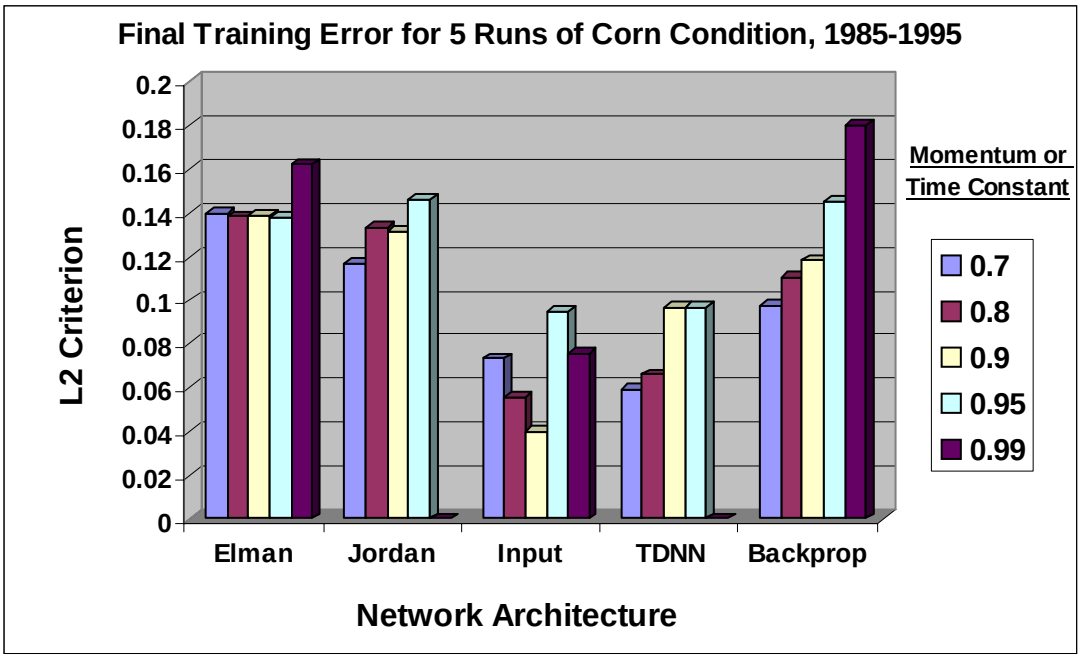

Figure 10. Performance of different learning architectures for crop condition monitoring

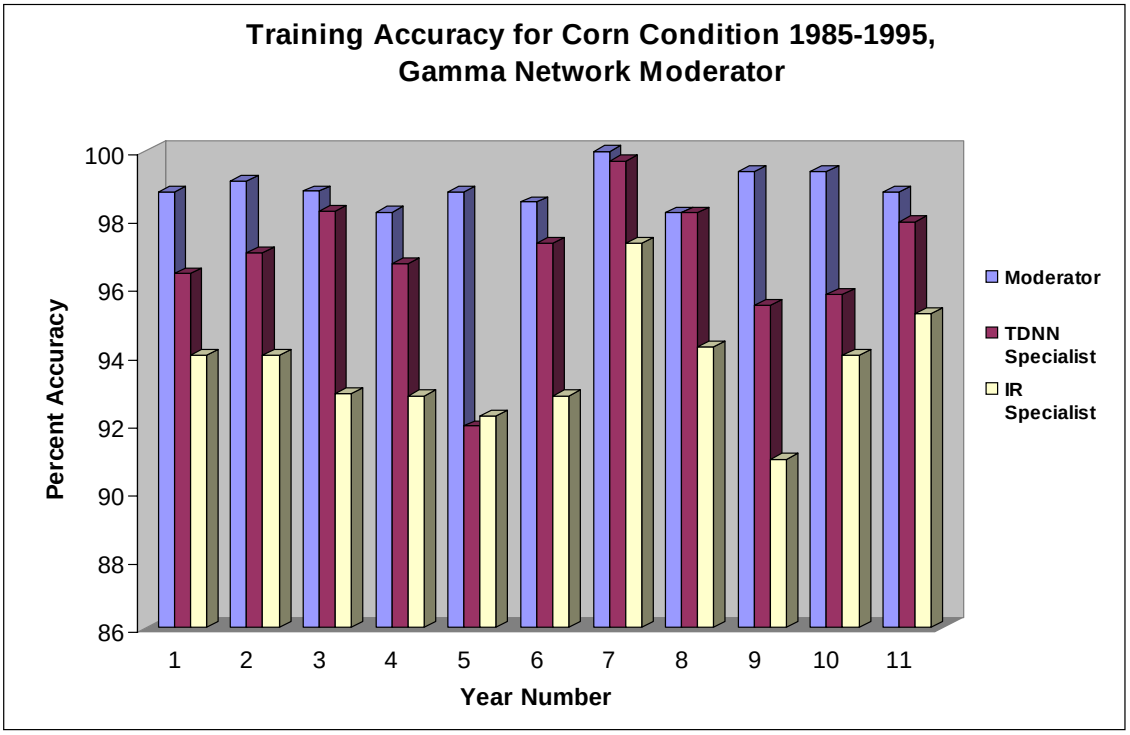

Figure 11. Training results for partitioned HME with gradient learning

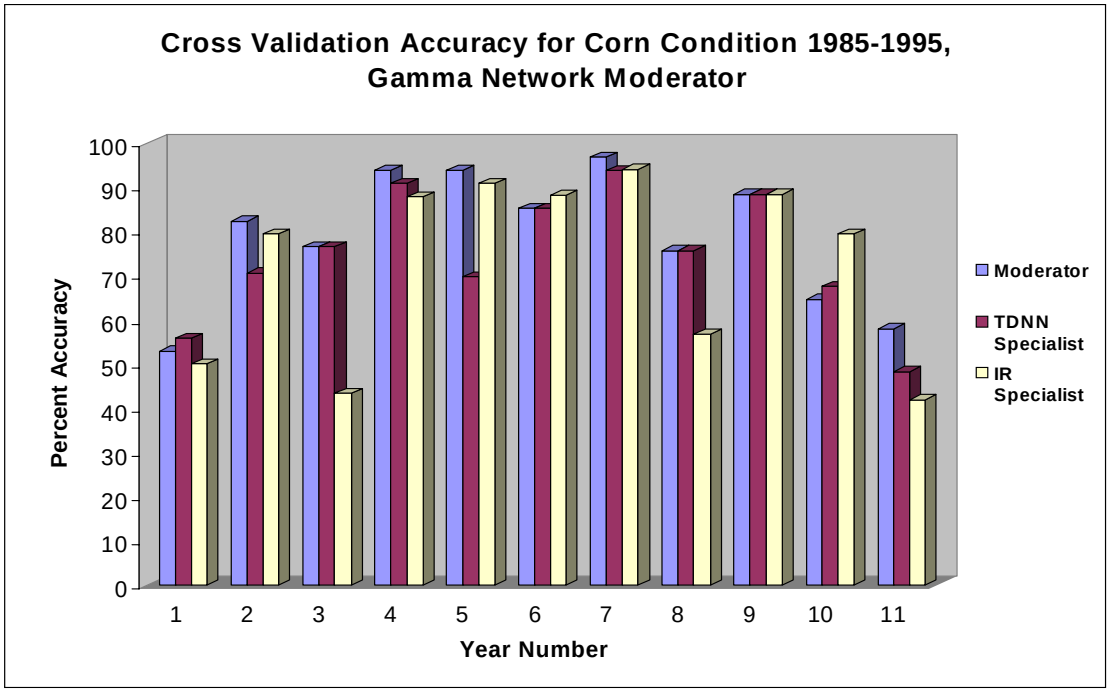

Figure 12. Cross validation results for partitioned HME with gradient learning

Research that is related to this dissertation [WS97, HGL+98] applies this methodology to specific problems in diagnostic monitoring for decision support (or *recommender*) systems [RV97].

5.1.2 Simulated and Actual Model Integration

Figure 13 visualizes a heterogeneous time series. The lines shown are phased *autocorrelograms*, or plots of autocorrelation shifted in time, for (subjective) weekly *crop condition* estimates, averaged from 1985-1995 for the state of Illinois. Each *point* represents the correlation between one week's mean estimate and the mean estimate for a subsequent week. Each *line* contains the correlation between values for a particular week and all subsequent weeks. The data is heterogeneous because it contains both an autoregressive pattern (the linear increments in autocorrelation for the first 10 weeks) and a moving average pattern (the larger, unevenly spaced increments from 0.4 to about 0.95 in the rightmost column). The autoregressive process, which can be represented by a time-delay model, expresses weather "memory" (correlating early and late drought); the moving average process, which can be represented by an exponential trace model, physiological damage from drought. Task decomposition can improve performance here, by isolating the AR and MA components for identification and application of

the correct specialized architecture (a time delay neural network [LWH90, Ha94] or simple recurrent network [El90, PL98], respectively).

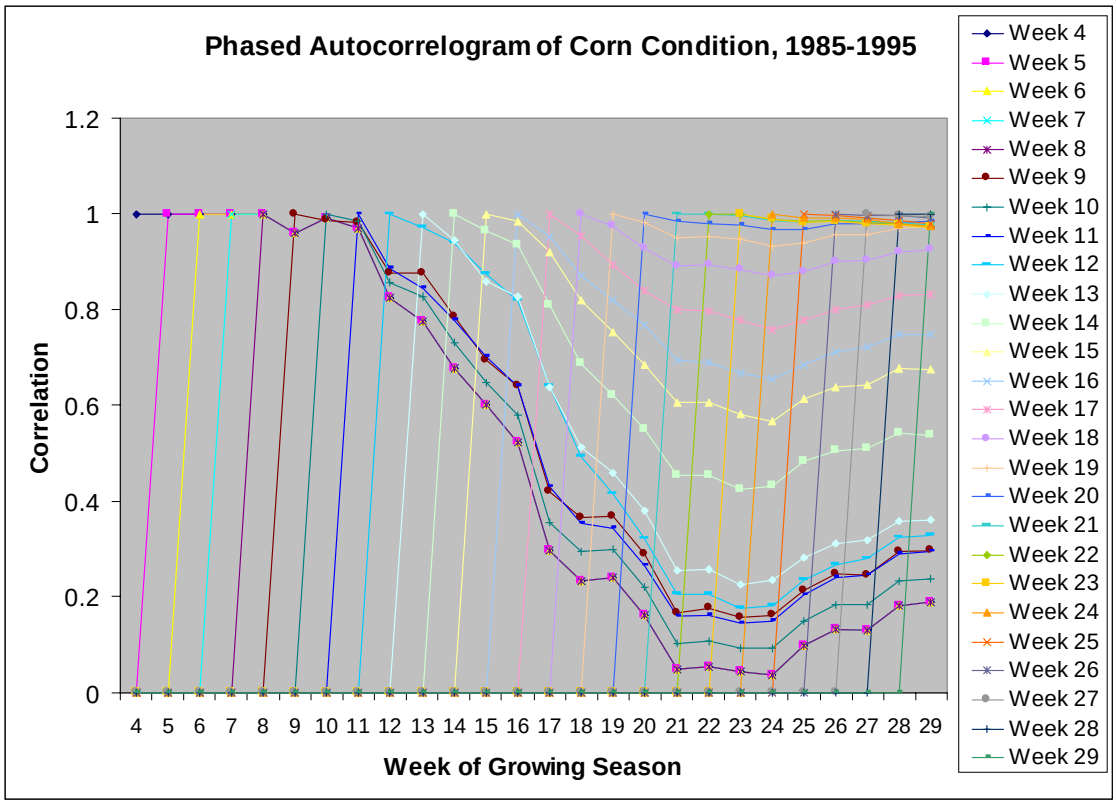

Figure 13. Phased autocorrelogram (plot of autocorrelation shifted over time) for crop condition (average quantized estimates)

Figure 14 shows a single line of this autocorrelogram plotted against a correlogram between predicted and actual values for crop condition. 26 temporal ANNs (all of the input recurrent type) are trained to produce this plot. The first 25 are *predictor* ANNs, trained to predict input $X(t + k)$ from $X(t)$, $1 \leq t \leq 25$, $1 \leq k \leq 25$. I then train a single input recurrent ANN to map $X(t + k)$ to a discrete predicted value of $Y(t + k)$. This could also be a nominal class such as {*very poor, poor, fair, good, very good*}, which is the learning target in Figures 2 and 3. We can think of this as *a predictive evaluation*, or *simulation*, model. The plot shows that recurrent ANNs can be expected to outperform linear prediction methods (and certainly outperform naïve linear or quadratic regression, which invariably predicts no change in the condition from one week to the next) in the "middle to distant future". This is important because the utility of near-term predictions tends to be lower for decision support systems [RN95].

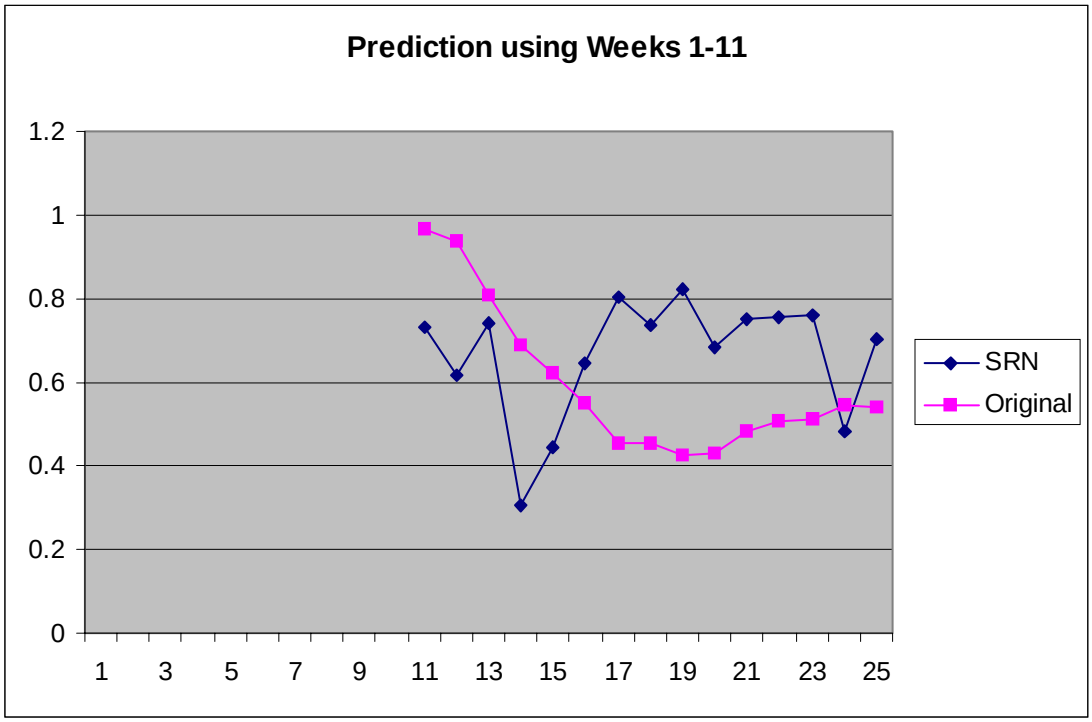

Figure 14. Predictive simulation for crop condition, using precipitation, temperature, working days, and maturity level up to Week 11, inclusive

5.1.3 Hierarchical Mixtures for Sensor Fusion

This section documents a sensor fusion experiment as applied to *musical tune classification*. First, I explain the choice of an experimental testbed for the moderator network. In my experiments using hierarchical mixtures, I focused primarily on *classification* of time series. The reasons for this restriction are that:

1. Classification of signals from multiple sources (sensor modalities, transforms, etc.) showcases the data fusion capabilities of the architecture, and can be used to benchmark the learning algorithm in comparison to other methods for multimodal integration (usually of predictions on time series).
2. Signal processing can be used to synthesize $\boldsymbol{A}_0$, the reformulated input attributes; such preprocessing methods are well understood [Ha94]. I discuss the experimental usefulness of this facility below.
3. Simple conceptual clustering can then extract the equivalence classes (e.g., using *k*-means clustering or other vector quantization methods such as competitive ANNs).

My architecture thus addresses one of the key shortcomings of many current approaches to time series learning: the need for an explicit, formal model of inputs from different modalities. For example, the specialists at each leaf in the SM network might represent audio and infrared sensors in a industrial or military monitoring system [SM93]. The SM network model and learning algorithm, described in Chapter 4, capture this property by allocating different channels of input (collected in each complex input attribute) to every specialist. Other models that can be represented by SM architecture are hierarchies of decision-making committees [Bi95].

I used both simple feedforward ANNs (multilayer perceptrons) and simple recurrent networks (SRNs), both trained by gradient learning (error backpropagation). For more information on SRNs, also known as autoregressive models or exponential trace memories, I refer the interested reader to [MMR97] and [PL98]. Recurrent feedback allows temporal information to be extracted from different subsets of a multichannel time series. More important, it allows this information to be recombined (i.e., a composite or higher-level classifier to be learned), even when the temporal attributes do not "line up" perfectly. I tested the Elman, Jordan, and input recurrent varieties of SRNs [El90, PL98], and found the input recurrent networks to achieve higher performance (accuracy and convergence rate) for exponentially coded time series, both alone and as part of the specialist-moderator networks.

In time series learning, preprocessing of input signals is the typical method of reformulating the input attributes [Ha94]. The experimental learning task was classification of (preprocessed) digital audio sequences. For this purpose, a database of 89 stylized musical tunes was synthesized, each containing 3-6 segments or "words" from an identifiable class (e.g., falling tone, rising tone, flat tone, etc.) delimited by silence (2-5 gaps). The tunes belong to 16 predetermined overall concept classes, factorizable into 4 equivalence classes of *frequency* and 4 of *rhythm*. Figure 15 shows this factorization. The learning task was to identify the overall concept class (among 16), given a preprocessed, multichannel frequency and rhythm signal. The training data consists of 73 tunes, with one randomly selected exemplar of each class being held out to obtain 16 cross-validation tunes. The numbers inside each circle in Figure 15 show the number of members from the overall 89-tune data set that belong to each class.

The input data was generated as follows. First, digital audio was recorded of the tunes being played by one of the authors. These samples were preprocessed using a simple autocorrelation

technique to find a coarse estimate of the *fundamental frequency* [BMB93]. This signal was used to produce the *frequency component*, an exponential trace of a tune over 7 input channels (essentially, a 7-note scale). The other group of input attributes is the *rhythm component*, containing 2 channels: the position in the tune (i.e., a time parameter ranging from 1 to 11) and a binary sound-gap indicator.

Figure 15 depicts non-modular and specialist-moderator architectures for learning the musical tune classification database. The non-modular network receives all 9 channels of input and is trained using the overall concept class. The first-level (leaf) networks in the specialist-moderator network receive *specialized* inputs: the frequency component only or the rhythm component only. The concatenation of frequency and rhythm components (i.e., the entire input) is given as input to the moderator network, and the target of the moderator network is the Cartesian product of its children's targets. Learning performance for these alternative network organizations is shown in Figure 16. Appendix A.3 documents some combinatorial properties relevant to this construction and this experiment.

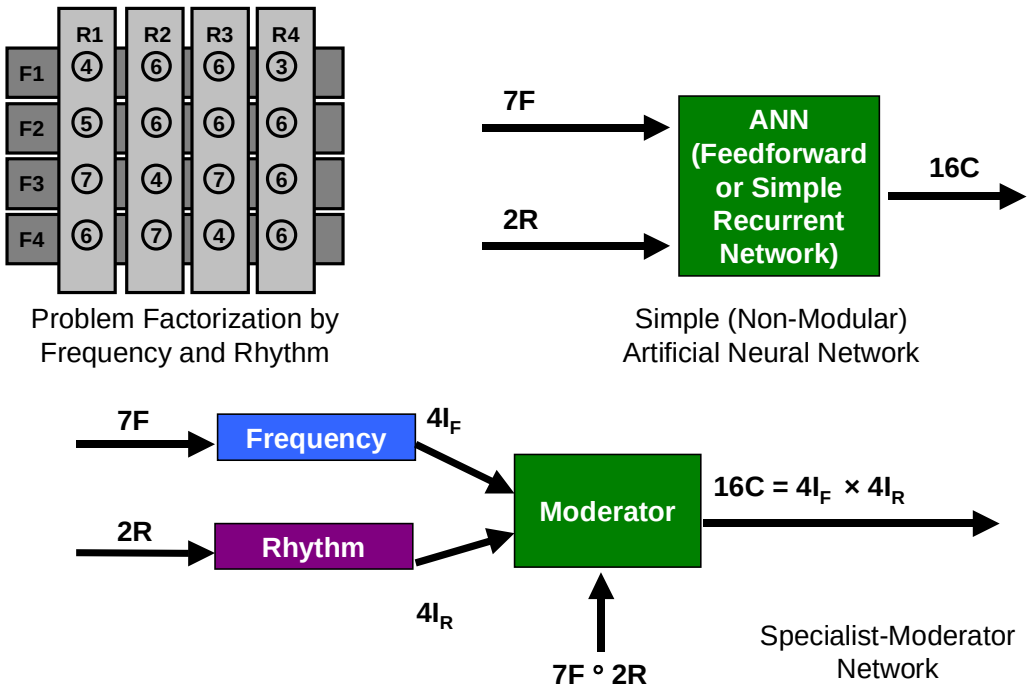

Figure 15. Organization of the musical tune classification experiment

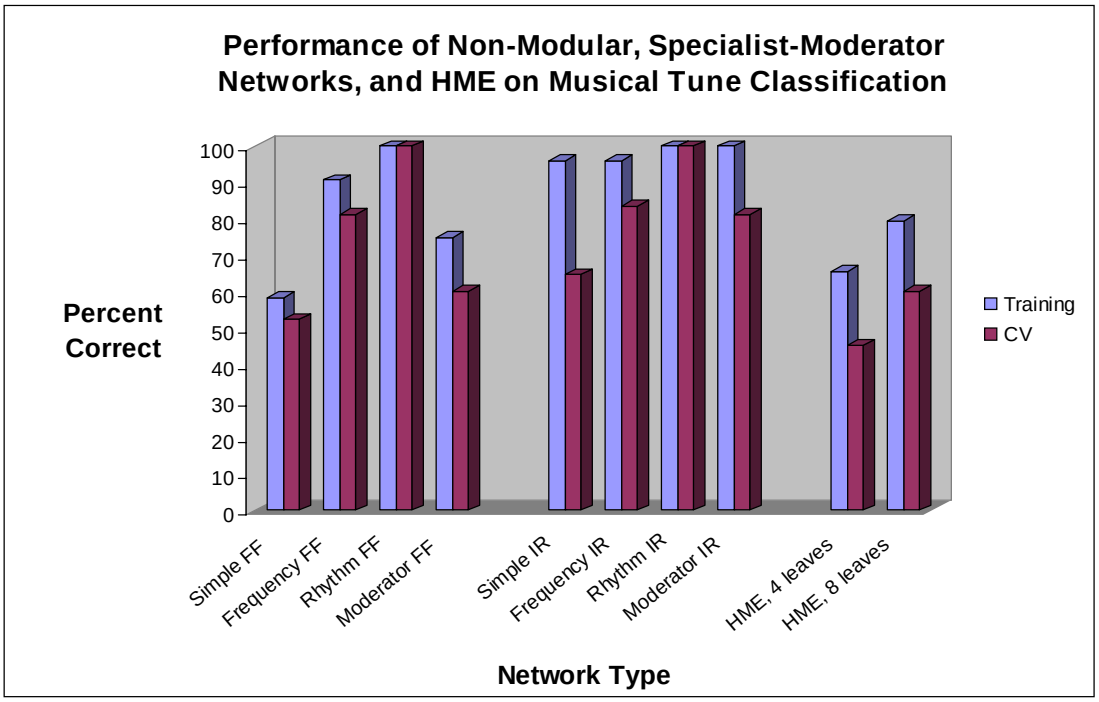

Figure 16. Performance (classification accuracy) of learning systems on the musical tune classification problem

5.2 Metric-Based Model Selection

This section presents results that illustrate the benefits of metric-based for model selection and compare with to other quantitative methods (such as naïve enumeration of learning configurations).

5.2.1 Selecting Learning Architectures

Figure 17 depicts a plot of the TDNN architectural metric curve for a series of 10000 independent, identically distributed, Uniform (0, 2) random variables. I use a discrete uniform distribution as a baseline because the unconditioned entropy is maximized. The variables are iid (an "order-0 Markov process") to provide a baseline for memory forms in time series learning. The definition of, and rationale for, the convolutional-code based metric for TDNNs, SRNs (specifically, input recurrent networks), and Gamma memories is given in Appendix C. As we would expect, the conditional entropy drops to nil as the number of delay lines goes to about 11 (i.e., with a window "depth" of 11, any sequence can be uniquely identified – note that this does not tell us what the generalization capability will be). The value for 0 delay lines is about lg(3) * 10000, also as expected.

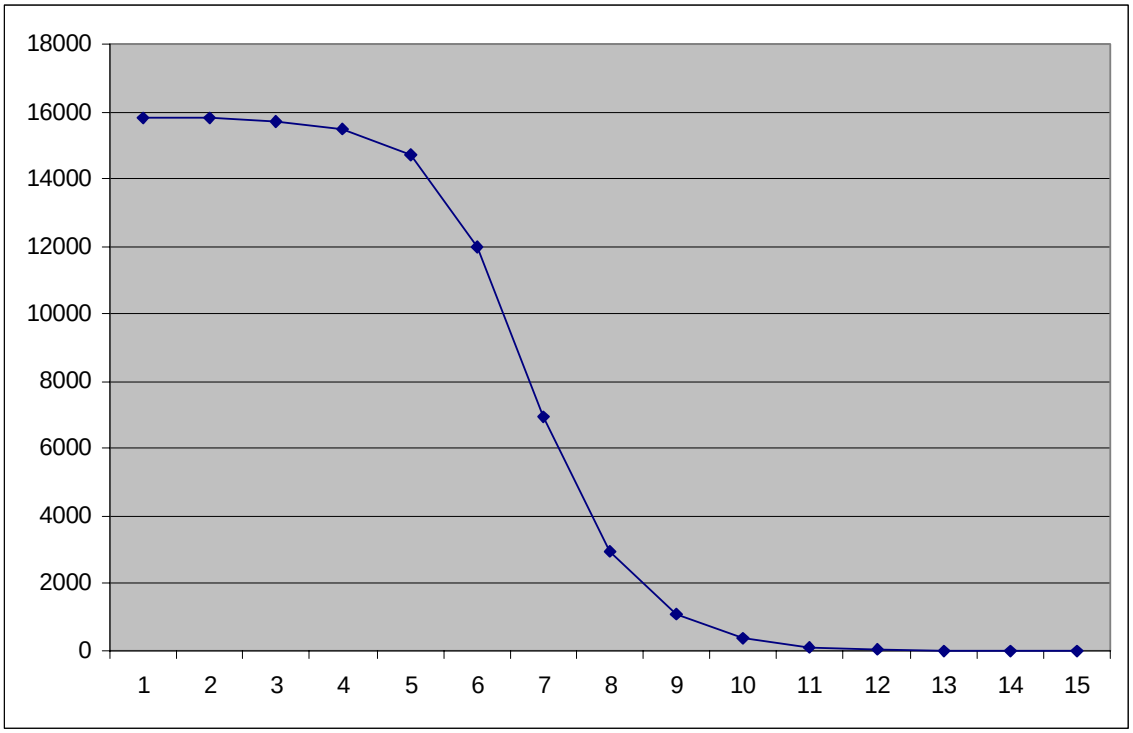

Figure 17. TDNN architectural metric for 10000 i.i.d. Uniform (0, 2) data points

5.2.2 Selecting Mixture Models

Figure 18 shows the classification accuracy in percent for moderator output for the concept:

$$\begin{aligned} \mathbf{Y} &= \prod_{i=1}^{k} Y_i \\ &= Y_1 \times Y_2 \times \ldots \times Y_k \\ Y_i &= X_{i1} \oplus X_{i2} \oplus \ldots \oplus X_{in_i} \\ X_{ij} &\in \mathbf{H} \equiv \{0, 1\} \end{aligned}$$

documented in Appendix D. All mixture models are trained using 24 hidden units, distributed across all specialists and moderators. When used as a heuristic evaluation function for partition search, the HME metric documented in Appendix C finds the best partition for the 5-attribute problem (shown below) as well as 6, 7, and 8, with no backtracking, and indicates that an HME-type mixture should be used.

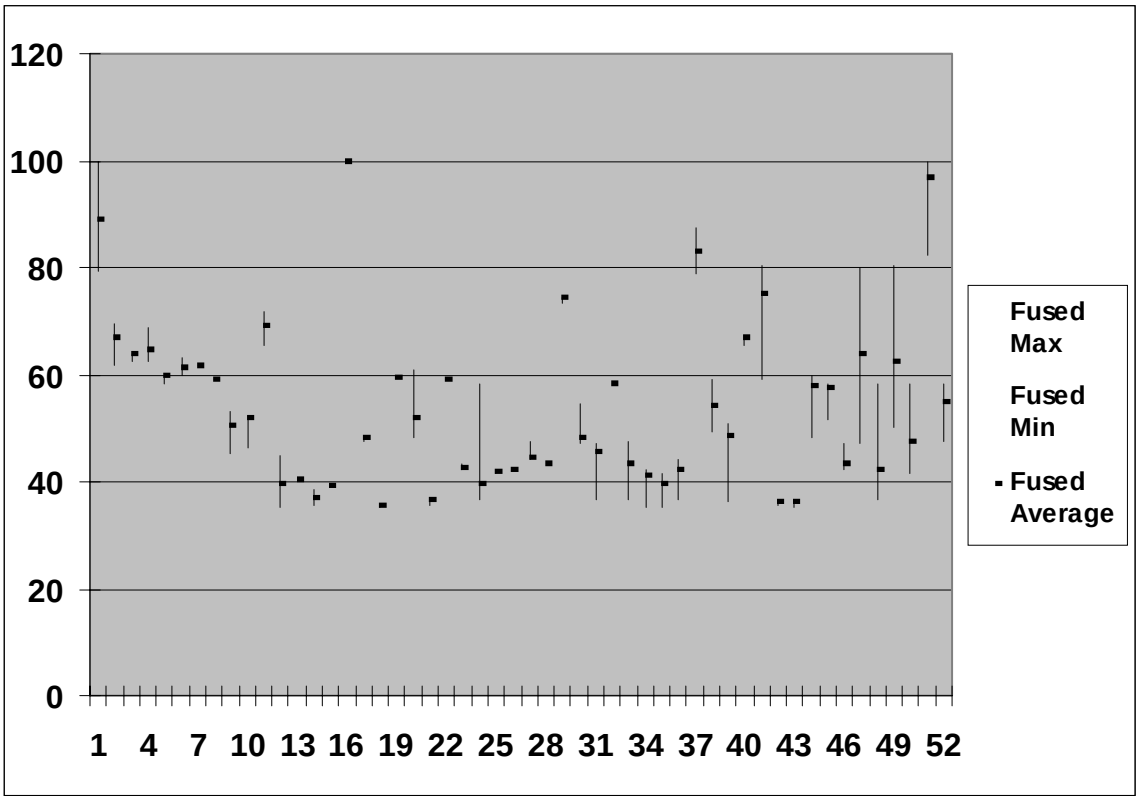

Figure 18. Min-max-average plot of classification accuracy for a partitioned, 5-attribute modular parity problem

5.3 Partition Search

This section presents the results for the attribute partition search algorithm given in Chapter 2 and for which an evaluation function is derived in Appendix C.

5.3.1 Improvements in Classification Accuracy

This section documents improvements in classification accuracy as achieved by attribute partitioning. Figure 19 shows how the optimal partition {{1,2,3}{4,5}} for the concept:

$$parity(x_1, x_2, x_3) \times parity(x_4, x_5)$$

as defined in Section 5.2.2 achieves the best specialist performance for any size-2 partition.

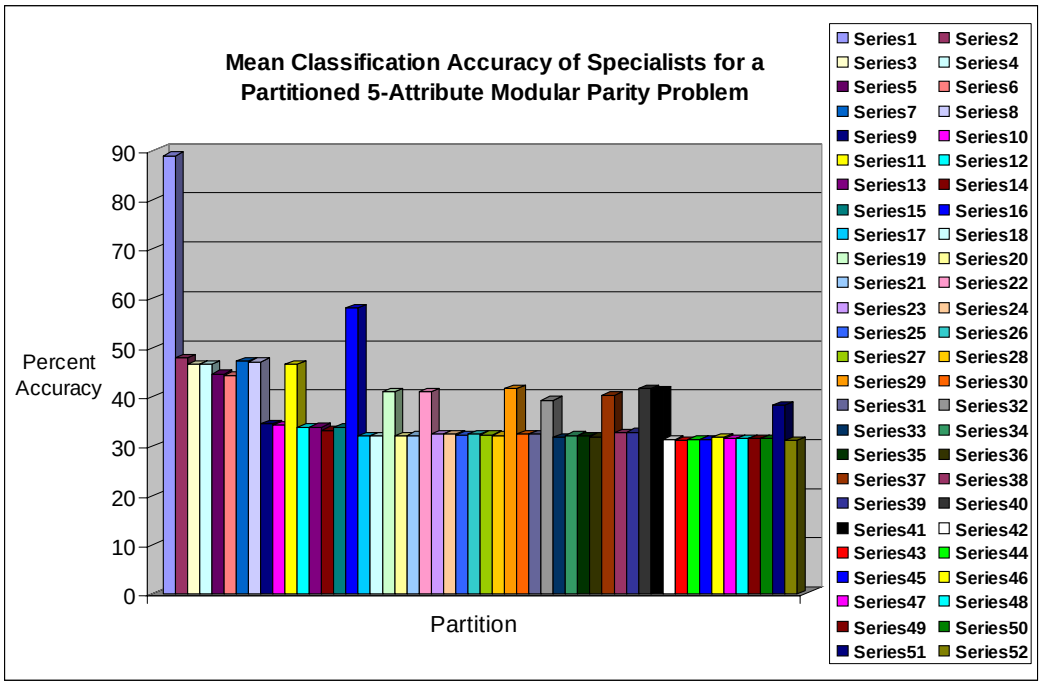

Figure 19. Specialist performance for all possible partitions of a 5-set

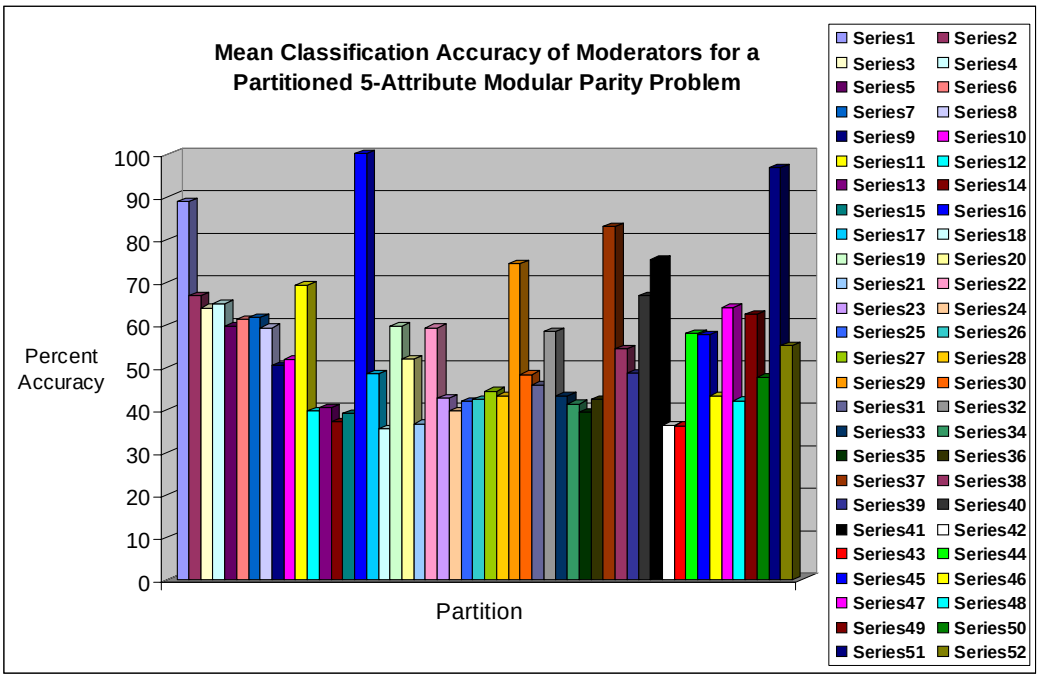

Figure 20. Moderator performance for all possible partitions of a 5-set

Figure 20 shows how this allows it to achieve the best moderator performance overall. Empirically, "good splits" (especially descendants and ancestors of the optimal one, i.e., members of its schema [BGH89]) tend to perform well.

5.3.2 Improvements in Learning Efficiency

n	2^n	B_n	Partitioning Runtime (s)	MMI Runtime (s)	FS Runtime (s)	Peak MMI Memory (KB)	Peak FS Memory (KB)
0	1	1	1	1	1	1040	1040
1	2	1	1	1	1	1050	1040
2	4	2	1	1	1	1060	1040
3	8	5	2	1	1	1070	1040
4	16	15	4	1	1	1080	1040
5	32	52	16	1	2	1100	1100
6	64	203	77	4	5	2200	1200
7	128	877	391	10	21	8600	1600
8	256	4140	2154	28	~40	31000	2800
9	512	21147	13454	87	~80	91600	20000
10	1024	115975	98108	281	~1500	374000	~300000

Table 4. Empirical performance statistics (time/space complexity) for metrics and (naïve) partition search

Table 4 shows the problem size, runtime, and memory consumption for the distributional metrics, documented in Appendix C.2.1. The MMI metric is the evaluation function for partition search. These values are graphed in Figure 21.

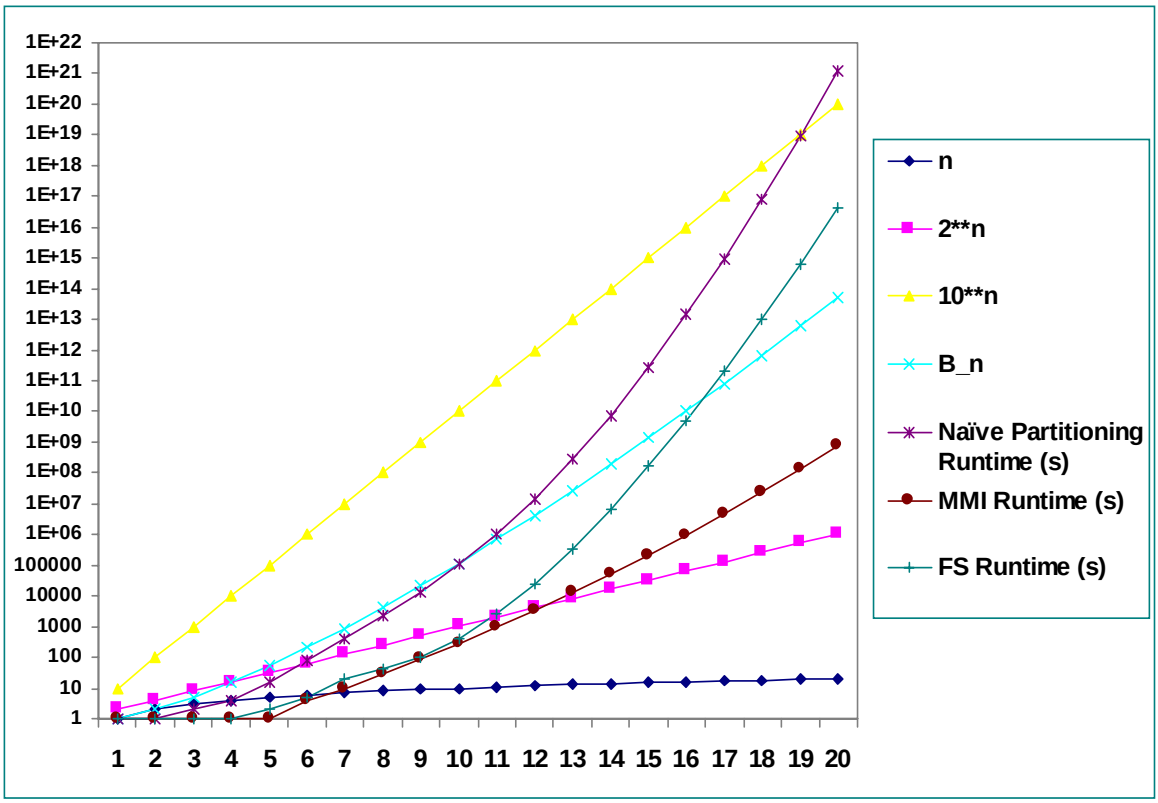

Figure 21. Logarithmic graph of problem size and running time (attribute parttitioning and distributional metrics)

5.4 Integrated Learning System: Comparisons

This section concludes the presentation of experimental results with comparisons to existing inductive learning systems, traditional regression-based methods as adapted to time series prediction, and non-modular probabilistic networks (both atemporal and ARIMA-type ANNs).

5.4.1 Other Inducers

Table 5 lists performance statistics (classification accuracy and running time) using atemporal inducers such as *ID3*, *C5.0*, Naïve Bayes, *IBL*, and *PEBLS* on the corn condition monitoring problem (babycorn data set) described in Section 5.1.1. The darkly-shaded rows in Table 5 denote partitioning mixtures, including the *multi-strategy HME* model that I used (this corresponds to the "HME, gradient" entry in Table 1 of Chapter 3). The lightly-shaded rows denote aggregating mixtures (in this case, bagged versions of the atemporal inducers).

	Classification Accuracy (%)							
	Training				Cross Validation			
Inducer	**Min**	**Mean**	**Max**	**StdDev**	**Min**	**Mean**	**Max**	**StdDev**
ID3	100.0	100.0	100.0	0.00	33.3	55.6	82.4	17.51
ID3, bagged	99.7	99.9	100.0	0.15	30.3	58.2	88.2	18.30
ID3, boosted	100.0	100.0	100.0	0.00	33.3	55.6	82.4	17.51
C5.0	90.7	91.7	93.2	0.75	38.7	58.7	81.8	14.30
C5.0, boosted	98.8	99.7	100.0	0.40	38.7	60.9	79.4	13.06
IBL	93.4	94.7	96.7	0.80	33.3	59.2	73.5	11.91
Discrete Naïve-Bayes	74.0	77.4	81.8	2.16	38.7	68.4	96.7	22.85
DNB, bagged	73.4	76.8	80.9	2.35	38.7	70.8	93.9	19.63
DNB, boosted	76.7	78.7	81.5	1.83	38.7	69.7	96.7	21.92
PEBLS	91.6	94.2	96.4	1.68	27.3	58.1	76.5	14.24
IR Expert	*91.0*	*93.7*	*97.2*	*1.67*	*41.9*	*72.8*	*94.1*	*20.45*
TDNN Expert	*91.9*	*96.8*	*99.7*	*2.02*	*48.4*	*74.8*	*93.8*	*14.40*
MS-HME	*98.2*	*98.9*	*100.0*	*0.54*	*52.9*	*79.0*	*96.9*	*14.99*

Table 5. Performance of a HME-type mixture model compared with compared with that of other inducers on the crop condition monitoring problem

	Classification Accuracy (%)							
	Training				Cross Validation			
Inducer	**Min**	**Mean**	**StdDev**	**Max**	**Min**	**Mean**	**StdDev**	**Max**
ID3	99.4	99.4	0.09	99.6	46.6	63.4	5.67	73.2
ID3, bagged	99.4	99.4	0.09	99.6	48.6	63.4	5.55	74.0
ID3, boosted	99.4	99.4	0.09	99.6	53.4	66.6	4.85	83.6
C5.0	95.0	95.8	0.64	96.3	67.1	77.1	3.41	84.9
C5.0, boosted	94.4	98.9	1.11	99.6	57.5	77.5	5.57	89
IBL	92.7	94.0	1.02	95.6	41.1	52.7	4.88	62.3
Discrete Naïve-Bayes	93.8	95.6	0.78	96.3	41.1	59.6	4.79	67.1
DNB, bagged	93.4	94.6	0.79	96.3	47.9	60.8	4.19	67.1
DNB, boosted	93.8	94.4	0.47	96.5	45.2	58.3	5.34	69.2
PEBLS	72.6	76.8	1.67	84.2	30.8	42.5	4.71	56.8
SM net, FF	*74.9*	*74.9*	*0.00*	*74.9*	*60.2*	*60.2*	*0.00*	*60.2*
SM net, IR	*100.0*	*100.0*	*0.00*	*100.0*	*81.3*	*81.3*	*0.00*	*81.3*

Table 6. Performance of an SM network mixture model compared with that of other inducers on the musical tune classification problem

Table 6 lists performance statistics for the musical tune classification problem (S4 data set) described in Section 5.1.2. The non-ANN inducers tested are all part of the *MLC++* package [KSD96].

5.4.2 Non-Modular Probabilistic Networks

This section summarizes the performance of the modified hierarchical mixture (specialist-moderator networks of feedforward and input recurrent ANNs) used on the time series classification problem as described in Section 5.1.2, as compared to non-modular regression models (feedforward and input recurrent ANNs trained by delta rule).

Network type	Size (units per layer)	Number of weights	Max epochs
Simple (overall)	9-48-16	1200	4000
Rhythm Specialist	9-16-4[8]	208	2000
Frequency Specialist	7-16-4	176	2000
Moderator (overall)	17-24-16	792	2000

Table 7. Design of non-modular and specialist-moderator ANNs

Table 7 shows the respective sizes of a feedforward ANN (see Figure 15) and the components of a specialist-moderator network of feedforward ANNs, along with the number of training cycles (epochs) allocated to each network. Note that the total number of learning weights is about the same for the simple network and the entire specialist-moderator network (the last three rows). Note also that the overall computational cost (as opposed to wall clock time) is equalized, because the specialists can be trained concurrently. The same network sizes and epochs were allocated for the SRNs and specialist-moderator networks of SRNs.

Design	Network Type	Training MSE	Training Accuracy	CV MSE	CV Accuracy
Feedfwd.	*Simple*	*0.0575*	*344/589 (58.40%)*	*0.0728*	*67/128 (52.44%)*
Feedfwd.	Rhythm	0.0716	534/589 (90.66%)	0.1530	104/128 (81.25%)
Feedfwd.	Frequency	0.0001	589/589 (100.0%)	0.0033	128/128 (100.0%)
Feedfwd.	*Moderator*	*0.0323*	*441/589 (74.87%)*	*0.0554*	*77/128 (60.16%)*
Input rec.	*Simple*	*0.0167*	*566/589 (96.10%)*	*0.0717*	*83/128 (64.84%)*
Input rec.	Rhythm	0.0653	565/589 (95.93%)	0.1912	107/128 (83.59%)
Input rec.	Frequency	0.0015	589/589 (100.0%)	0.0031	128/128 (100.0%)
Input rec.	*Moderator*	*0.0013*	*589/589 (100.0%)*	*0.0425*	*104/128 (81.25%)*

Table 8. Performance of non-modular and specialist-moderator networks.

Table 8 shows the performance of the non-modular (simple feedforward and input recurrent) ANNs compared to their specialist-moderator counterparts. Each tune is coded using between 5 and 11 exemplars, for a total of 589 training and 128 cross validation exemplars (73 training and

[8] Even though cluster definition (to obtain the intermediate concepts, or classification targets, for the rhythm specialist) was performed using only 2 attributes, experiments with attribute subset selection (in addition to partitioning) showed a slight increase in performance as "frequency-relevant" attributes were added. Therefore, all 9 attributes were used as input (in *supervised mode only*) to the rhythm specialists.

16 cross validation tunes). The italicized networks have 16 targets; the specialists, 4 each. Prediction accuracy is measured in the number of individual exemplars classified correctly (in a 1-of-4 or 1-of-16 coding [Sa98]). Significant overtraining was detected only in the frequency specialists. This did not, however, affect classification accuracy for my data set. The results illustrate that input recurrent networks (simple, specialist, and moderator) are more capable of generalizing over the temporally coded music data than are feedforward ANNs. The advantage of the specialist-moderator architecture is demonstrated by the higher accuracy of the moderator test predictions (100% on the training set and 81.25% or 15 of 16 tunes on the cross validation set, the highest among the learning algorithms tested).

Network Type	Levels	Weights	Expert	Fusion	Epochs
HME, 4 leaves	2	1296	GLIM (tanh)	GLIM (linear)	4000
HME, 8 leaves	3	1332	GLIM (tanh)	GLIM (linear)	4000
S-M net, 2 leaves	1	1176	FF or IR	FF or IR	4000

Table 9. Design parameters for HME and specialist-moderator networks.

Table 9 summarizes the topology of two hierarchical mixtures of experts I constructed for the musical tune classification problem. Each is designed to have approximately the same network complexity (number of learning parameters, i.e., expert and gating weights) as the specialist-moderator network (which is shown in Figure 15 and is similar in type to those documented in Chapters 3 and 4). The table also indicates what output nonlinearities are used. The HME design above is consistent with that described by Jordan and Jacobs for regression problems. For purposes of standardized comparison, however, I use a gradient learning algorithm (the same as in all of the specialist-moderator networks) instead of the EM algorithm with *iteratively reweighted least squares* (*IRLS*) used in Jordan and Jacobs's HME implementation [JJ94].

Design	Training MSE	Training Acc.	CV MSE	CV Accuracy
HME, 4 leaves	0.0576	387/589 (65.71%)	0.0771	58/128 (45.31%)
HME, 8 leaves	0.0395	468/589 (79.46%)	0.0610	77/128 (60.16%)
S-M net, FF	0.0323	441/589 (74.87%)	0.0554	77/128 (60.16%)
S-M net, IR	0.0013	589/589 (100.0%)	0.0425	104/128 (81.25%)

Table 10. Performance of HME and specialist-moderator networks.

As Table 10 shows, the HME algorithm with 8 leaves outperforms the version with 4 and is comparable to the specialist-moderator network of feedforward networks. It is, however, outperformed by the specialist-moderator network of input recurrent networks. This is significant because incorporating recurrence into HME requires nontrivial modifications to the algorithm. Equally important, I expect that a hierarchy of input recurrent expert and gating networks, with identical outputs and the original input presented to each, would incur excessive overhead due to its complexity. Given my uniform complexity restrictions, it would then be impractical to build a 3-level or even 2-level tree.

5.4.3 Knowledge Based Decomposition

In the musical tune classification problem, the intermediate targets are equivalence classes $I_F = \{ F_1, F_2, F_3, F_4 \}$ and $I_R = \{ R_1, R_2, R_3, R_4 \}$. This 4-by-4 factorization was discovered using competitive clustering by Gaussian radial-basis functions (RBFs) [Ha94, RH98]. In this experiment, the frequency and rhythm partitioning of *input* is self-evident in the signal processing construction, so the *subdivision of input* is known (note, however, that the intermediate targets are *not* known in advance). When the input subdivision is also unknown, automatic *subset selection* methods can be used to automatically determine which inputs are *relevant* to a particular specialist [KJ97].

6. Analysis and Conclusions

This dissertation has presented: a wrapper system for decomposition of inductive time series learning tasks by attribute partitioning; a metric-based procedure for coarse-grained selection of multiple models for subproblems, and a hierarchical mixture model for integration of trained submodels. The overall product is an integrated, multi-strategy learning system for decomposable time series, which combines unsupervised learning (constructive induction), supervised learning (using temporal, probabilistic networks), and model selection. This chapter reviews the system and assesses its theoretical and practical relevance. First, I characterize the attribute partitioning-based decomposition system as a *wrapper* for supervised inductive learning. I review the benefits for time series learning as documented in Chapter 5, and outline some promising topics of continued research in this direction. Next, I evaluate the empirical results on attribute partitioning, and contrast the improvements in classification accuracy with their computational costs. I discuss the techniques, such as heuristic search, that I have applied to make the most of this tradeoff; the obstacles that remain; and future work that addresses some of these obstacles. I then survey the results regarding the use of multiple models in time series learning – namely, the effectiveness of: subproblem definition, metric-based model selection, and hierarchical mixtures of temporal probabilistic networks. Finally, I document the ways in which my approach has been applied to real time series, and may be of future use in analysis of large-scale, heterogeneous time series.

6.1 Interpretation of Empirical Results

This section analyzes the experimental results reported in the previous chapters, especially Chapter 5. It begins with a discussion of the main findings and their ramifications, continues with an account of the design choices and tradeoffs incurred, and concludes with a brief investigation into the general properties of the test beds used in this dissertation.

6.1.1 Scientific Significance

The experimental results described in Chapter 5 bear out the following hypotheses:

1. *[Chapter 2] Decomposition of learning tasks by attribute partitioning can be useful in reducing variance when computational resources are limited (i.e., a consistent bound is imposed on network complexity and time until convergence, or – more accurately – both).*

2. *Conversely, when desired classification accuracy is specified, this type of decomposition can reduce the complexity of the model needed to achieve the target.*

3. *[Chapter 2] Using exhaustive enumeration of partitions, the optimal partition (with respect to multiple models) can be identified for a particular learning model using statistically sufficient number of tests.* In practice, this is typically not a very high number, but the number of partitions grows superexponentially as a function of the number of attributes.

4. *[Chapter 3] This method can be extended to testing all configurations of the learning models in a multi-strategy learning system.* In practice, this is typically a very high number not only because of the moderately large number of configurations involved for a single problem definition, but because the growth of configurations is exponential in the number of subsets of the partition.

5. *[Chapter 4] Adaptation of HME to partitioned concept learning problems is effective for decomposable time series (such as the corn condition monitoring test bed); it achieves data fusion through specialization of expert networks to specific types of embedded temporal patterns, or memory forms.*

6. *[Chapter 4] SM networks can be used as an alternative hierarchical mixture model when there is a high degree of factorial structure in the learning problem (such as in the musical tune classification test bed).*

7. *[Chapter 4] Hierarchical mixture models, as applied in this system, support multi-strategy learning from time series, using multiple types of temporal probabilistic networks.*

8. *[Chapter 2] Partition evaluation can be made much more efficient by casting it as a state space search problem and using a heuristic evaluation function.* This allows data sets with many more input attributes to be decomposed, although complexity is reduced only to an exponential function of the number of attributes in the worst-case.

9. *[Chapter 3] The architectural metrics are positively correlated with learning performance by a particular configuration of learning architecture (for a learning problem defined on a particular subset of a time series).* This makes them approximate indicators of the suitability

of the corresponding architecture and the assumption that the learning problem adheres to its memory form. Thus, architectural metrics may be used for partial model selection.

10. *[Chapter 3] The distributional metrics for hierarchical mixture models are positively correlated with learning performance by a particular learning method (for a learning problem defined on a particular partitioning of a time series).* This makes them approximate indicators of the suitability of the corresponding mixture model and the assumption that the learning problem adheres to its characteristics (with respect to interaction among subproblems). Thus, distributional metrics may be used for partial model selection.

The findings in this list support the design philosophy that *modular* learning [JJ94, RH98] is beneficial when there exists a hierarchical decomposition that reduces variance without incurring excessive complexity. In the specialist-moderator framework, for instance, this means that the problem factorization is highly efficient. In time series classification test beds such as the musical tune classification problem, the factorization is optimal, and the specialist-moderator framework is shown to outperform other learning architectures of comparable complexity. The results in Section 5.4 also show that it is sometimes preferable to use specialist-moderator networks for data fusion instead of mixture models where all complex input attributes and intermediate targets are identical. This case occurs when: some decomposition of a learning task contains subproblems *that are easier to solve in isolation* (all other things being equal); these subproblems can be extracted through attribute partitioning and cluster definition; and the intermediate outputs can be recombined with a hierarchical mixture model. An important part of subproblem definition is the model selection step, which associates the "input-output" specification[9] with a mixture model type and a learning architecture for the subproblem. Chapter 5 gives examples heterogeneous time series for which this step can be completed manually or automatically.

6.1.2 Tradeoffs

A number of important performance tradeoffs were assessed in this dissertation. These include:

[9] Such a specification fully defines the instance space, or concept language, but is only a partial definition of the hypothesis language.

1. Bias/variance decomposition in terms of mixture modeling [GBD92, Fr98, Ro98]. The general design philosophy in attribute-driven problem decomposition is that, for heterogeneous time series, it is neither appropriate to use a "monolithic" model nor a traditional hierarchical mixture model. By "monolithic", I mean a non-modular learning model that is highly flexible but typically less tractable than one that employs some model selection (either adaptation or coarse-grained *technique selection* [EVA98]). Section 3.1.1 discusses this issue in more detail. By "traditional" mixture models, I mean those that adapt their components (experts or specialists) as part of the learning algorithm *primarily as a substitute for explicit decomposition* [Ro98]. Some mixture models, such as HME, are used in both supervised modes and with some cluster definition [Ha94, Am95, Bi95].

2. Efficiency versus accuracy in attribute partitioning-based decomposition. Optimal partitions are neither necessarily unique, nor does the state space always exhibit *locality* very well [Go89, RS90, Gi96]. One interesting line of future research is to apply genetic search [BGH89, Go89, LY93] to the attribute partitioning problem. This is a high-level change-of-representation problem that eases the bias/variance tradeoff at the level of supervised learning.

3. Efficiency versus accuracy in metric-based model selection. A highly accurate, but also very expensive, approach, is to try every configuration of model available [Gr92, Ko95]. Even when only a small or moderate constant number of configurations is available, problem decomposition confounds this approach because the interactions among subproblems are difficult to predict. Problems with such methods have been documented in other technique selection applications, such as compression of heterogeneous files [HZ95]. The alternatives are to: limit partitioning to a small number of subsets (because of the exponential growth in the number of configurations that must be considered); make biasing assumptions about subproblem interaction; or use a prescriptive metric to approximate or predict performance as is done here.

6.1.3 Representativeness of Test Beds

Of additional interest is the question: to what degree are the real and synthetic test beds used in experiments (and for high-level calibration of the model) representative of all time series?

Properties of interest that are captured by all test beds or represented by individuals are:

1. *Heterogeneity.* All of the real-world time series are heterogeneous, but this is due to various causes (multiple, interacting physical processes in the crop monitoring test bed; signal preprocessing in the musical tune classification test bed). Some of the synthetic data sets (such as the partitioning test bed) are heterogeneous, some (such as the calibration sets for architectural metrics) homogeneous.

2. *Decomposability.* All of the real-world test beds are decomposable *to some degree* by attribute partitioning. An interesting topic of future research is to consider alternative methods for problem decomposition (such as knowledge-based constructive induction [Gu91, Do96]). All heterogeneous, synthetic time series have decomposable problem definitions.

3. *Mixtures.* Each type of mixture model (with various prescribed training algorithms) is represented by one real-world data set and by several synthetic data sets.

4. *Embedded temporal patterns.* All of the *memory forms* in my implementation of the database of learning techniques are represented by various subproblems.

6.2 Synopsis of Novel Contributions

This section presents a synopsis of contributions to the theory of multi-strategy machine learning and its application time series analysis.

6.2.1 Advances in Quantitative Theory

The main contribution of this dissertation to the theory of *integrative methods* for supervised inductive learning (cf. [Ko95, KJ97], as depicted in Figure 22) is a new wrapper system for probabilistic networks (as depicted in Figure 23). This methodology is not specific to time series.

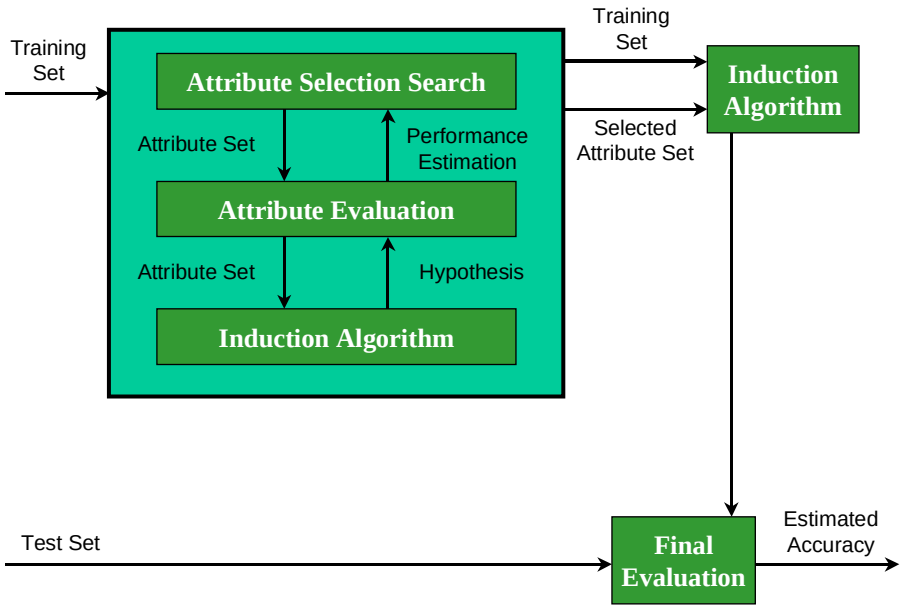

Figure 22. Wrapper systems for supervised inductive learning [Ko95, KJ97].

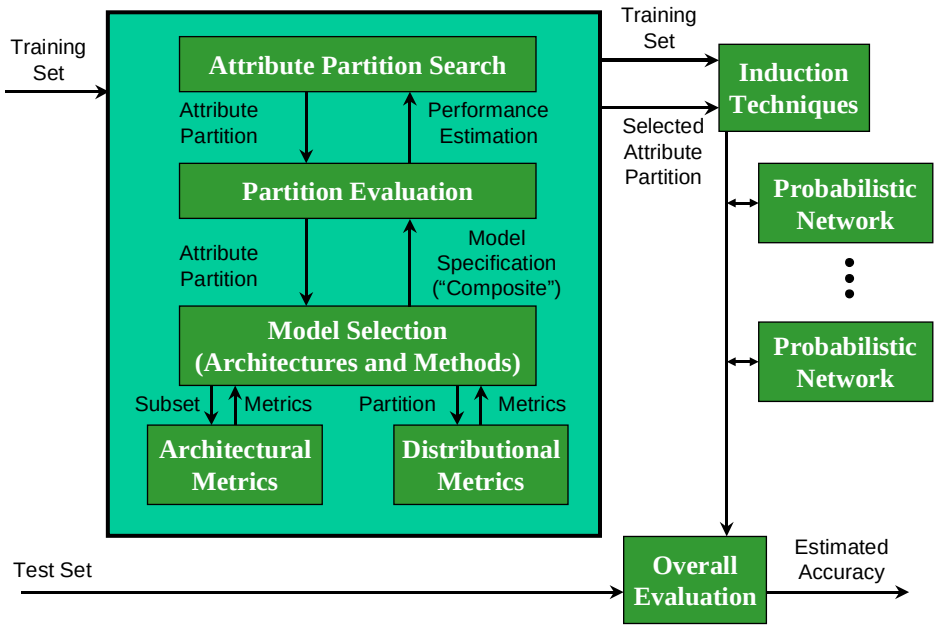

Figure 23. New wrapper system based on attribute partition search and metric-guided model selection.

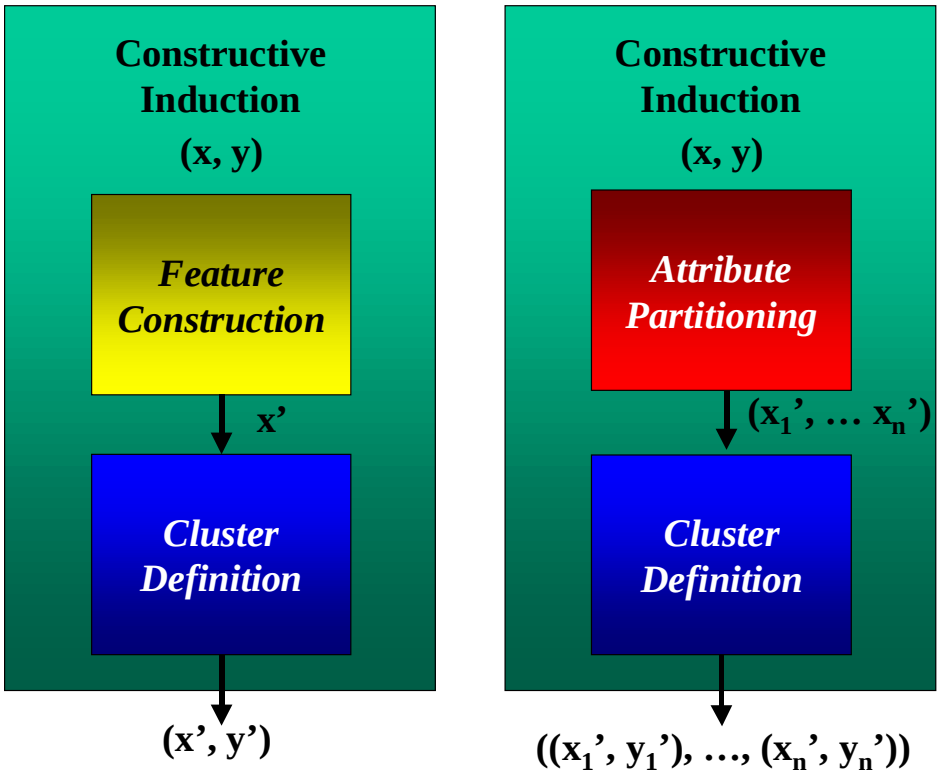

Figure 24. Traditional constructive induction and its adaptation to systematic (attribute-driven) problem decomposition.

A second major contribution of this work is the adaptation of constructive induction to problem decomposition (as depicted in Figure 24 and discussed in Section 2.2). Again, this extension of previous work is not specific to time series.

6.2.2 Summary of Ramifications and Significance

Heterogeneous time series learning problems are abundant in applications such as multimodal diagnosis and monitoring [BSCC89, HLB+96, HGL+98], multimodal sensor integration [SM93, Se98], multimodal human-computer interaction [Hu98], integrative knowledge based simulation [WS97], and multiagent problem solving [GD88]. This dissertation focuses on the first three categories of problems, though it may extend to certain problems in the last two categories.

Multimodality is a general property of data such that it is generated by multiple sources. It may originate from the use of *multiple models* in diagnosis [WCB86, Mi93], hybrid temporal and atemporal inference [Sh95], qualitative and numerical data [He91, Gr98], integration of sensors

and laboratory measurements with subjective evaluations [BSCC89, RN95, Gr98], and other similar phenomena. For example, sensors (artificial or natural) may be tuned to or configured for different *modes of perception*, in which case the tasks of learning, representation, and integration are called *multisensory* [SM93]. Multisensory integration is a central problem in design of intelligent alarms [HLB+96], where false positives and localization error may be reduced by combining multiple percepts. This phenomenon has been observed and studied in neurobiology [SM93, Se98] and is beginning to be investigated through simulation in computational neuroscience [Se98]. Finally, multimodality in human-computer intelligent interaction is a natural consequence of the modes of communication used by humans: speech, handwriting, gestures, and facial expression, among others. For example, lipreading as an enhancement to continuous speech recognition from audio is a topic of current research [Hu98].

In nearly all cases, multimodal time series are inherently heterogeneous. That is, because multiple, different sources *generated* the data, there are typically different temporal patterns embedded in this data. For a given learning problem as defined on such time series, these pattern types are often recognizable. If so, they may be exploited by separation of the problem to obtain subproblems – in which case the learning problem is referred to as *decomposable*. The objective is to find *homogeneous* time series, those with only one dominant embedded temporal pattern, typically originating from a single input source. This is where attribute partitioning methods may be useful; they decompose time series data into homogeneous parts, along groups of attributes.

Identifying homogeneous subsets of a time series data set is only a partial solution. When such partitions are achievable, it is still necessary in most cases to select a suitable model (hypothesis language) for *each* subset. In real-world heterogeneous time series, the description of the suitable model tends to vary along rather coarse parameters, such as the degree of autoregressive versus moving average characteristics [Mo94, Ch96, MMR97]. This is borne out by experimental evidence, as documented in Chapter 5.

It is typically feasible to combine the subset selection and model selection methods presented in this research whenever three conditions are met. All three of these conditions are attainable by refinement of metrics, compilation of training corpora for metrics, mixture modeling, clustering, and preprocessing of attributes, in roughly descending order of importance. This research is

foremost concerned with the first two issues and to a limited extent with the remaining three, but all are addressed.

First, the training data must be sufficiently homogeneous *with respect to the set of identifiable high-level temporal patterns* for a "dominant pattern type" to be unequivocally indicated. This is an issue of metric design and population of the database of learning techniques. Second, there nust be sufficient *representative data* for training the model selection mechanism based on these patterns. This is an issue of corpus design and metric normalization. Third, a supplementary unsupervised learning mechanism is required for determination of intermediate targets *given* an attribute partition, and a data fusion mechanism is required for recombining trained models for each component of the partition. This is somewhat data-driven, and is addressed by clustering and mixture models, respectively.

The key contribution of subset partitioning as applied to heterogeneous time series learning is its capability to effectively decompose learning tasks. The quality of decompositions depends primarily on its homogeneity. The benefits of a homogeneous partition are threefold: first, it is easier to fit the most appropriate model to each subtask; second, it automatically groups attributes into the appropriate subsets; third, the data fusion problem is simplified (i.e., there is less work for the mixture model to perform).

Fitting the most appropriate model to each subtask tends to reduce network complexity and classification error, as is demonstrated in Chapter 5. It also provides support for multi-strategy learning methods [HGL+98]. Grouping of attributes is a localized form of *relevance determination.* That is, the attributes are adjudged to "belong together" if and only if they are mutually cohesive and relevant to their common intermediate target, *and* if their grouping entails a tractable data fusion problem (as documented in Appendix D.2.1). The last benefit is a consequence of a criterion that minimizes the *inefficiency* of a partitioning, as described in Section 2.4 and in Chapter 4.

Heterogeneity is especially salient in time series learning because:

1. Many problems that can be cast as time series analysis involve learning from multiple sources of observations. This includes diagnosis and monitoring based on multiple models, multisensor integration, and multimodal HCI.

2. The performance element for an intelligent time series analysis system often necessitates models for different aspects of change among input variables [Do96, Io96].
3. Many time series learning problems, such as crop monitoring, also possess a spatial aspect. These *spatiotemporal* learning problems may exhibit heterogeneity because of the location of sensors. Related problems include: adaptive remote sensing, mobile robotics, and distributed agents including some Internet agents [Sa97].

6.3 Future Work

Subsequent work following this dissertation will examine some potential solutions to some problems that it has recognized, but not directly addressed. These include: further improvement of performance in the real-world test beds studied; extension of the lessons learned and performance gains achieved for these test beds, to larger-scale intelligent systems applications; and generalization of the results from synthetic and real data sets to other domains. This section presents some of the feasible research programs that arise as a result of the findings in this dissertation.

6.3.1 Improving Performance in Test Bed Domains

I have presented an algorithm for combining data from multiple input sources (sensors, specialists with different concentrations, etc.) and a modular, recurrent, artificial neural network for time series learning. This method can be extended beyond probabilistic networks, but for clarity of exposition and experimental standardization, I have focused on recurrent ANNs as the characteristic architecture for time series learning. Fusion of time series classifiers showcases the strengths of both mixture models because there are many *preprocessing methods* that produce reformulated input. The characteristic applications in this area are monitoring, prediction, and control – problems which often involve continuous output. For clarity, however, I have focused on discrete classification.

6.3.2 Extended Applications

One example of an extended learning test bed, which is related to the wide-area corn condition monitoring test bed that I developed, is the analysis of large geospatial databases for precision agriculture. A wealth of remote sensing, simulation, laboratory, and historical data has recently become available for computational studies. This data is collected at a much finer level of spatial granularity than that used in my pilot experiment for condition monitoring. One

important use of this data is the generation of *spatiotemporal statistics* [HR98a], approximated variables mapped over space and time, such as soil fertility, plant available water, and expected yield. Estimation of spatiotemporal statistics for agricultural applications varies greatly in difficulty, but can nearly always be enhanced using large geospatial databases.

Database Name	Database Type	Temporal Granularity (days)	Spatial Granularity (m^2)	Data Points Per Year[10]
Soil samples	**Map**	$\mathbf{1.40 \times 10^1}$	$\mathbf{2.0 \times 10^3}$	$\mathbf{2 \times 10^4}$
Elevation[11]	Map	2.52×10^2	1.0×10^1	3×10^5
Aerial image	Map/Sensor	7.00×10^0	1.0×10^{-2}	9×10^9
Yield[11]	Map/Sensor	2.52×10^2	2.0×10^2	1×10^4
Near infrared	Map/Sensor	1.40×10^1	1.0×10^0	5×10^7
Soil types	*Map/Computed*	1.40×10^1	2.0×10^2	2×10^5
Soil fertility	*Map/Computed*	1.40×10^1	1.0×10^1	5×10^6
Plant-available moisture	*Map/Computed*	1.00×10^0	1.0×10^0	7×10^8
Vegetative index[11]	*Map/Computed*	7.00×10^0	1.0×10^0	9×10^7
Nutrient uptake	*Map/Simulation*	1.00×10^0	1.0×10^0	7×10^8
Corn growth	*Map/Simulation*	1.00×10^0	1.0×10^0	7×10^8
Soybean growth	*Map/Simulation*	1.00×10^0	1.0×10^0	7×10^8
Planting density	Map/Historical	2.52×10^2	1.0×10^2	3×10^4
Tillage	Map/Historical	2.52×10^2	1.0×10^2	3×10^4
Fertilizer Application	**Map/Historical**	$\mathbf{1.26 \times 10^2}$	$\mathbf{1.0 \times 10^6}$	$\mathbf{5 \times 10^0}$
Chemical Application	**Map/Historical**	$\mathbf{1.40 \times 10^1}$	$\mathbf{1.0 \times 10^4}$	$\mathbf{5 \times 10^3}$
Soil composition	Map/Laboratory	1.40×10^1	2.0×10^3	2×10^4
Precipitation	Sensor	1.10×10^{-2}	1.0×10^6	6×10^4
Temperature	Sensor	1.10×10^{-2}	1.0×10^6	6×10^4
Evapo-transpiration	Sensor	1.00×10^0	1.0×10^6	7×10^2
Solar radiation	Sensor	1.00×10^0	1.0×10^6	7×10^2
Neutron (moisture) probe	Sensor	1.40×10^1	1.0×10^4	5×10^3
Agronomic Estimates	Historical	1.40×10^1	1.0×10^6	5×10^1
Pedology	Laboratory	1.40×10^1	2.0×10^3	2×10^4

Table 11. Geospatial databases available for time series learning reseearch with applications to large-scale precision agriculture.

[10] Order-of-magnitude estimates, assuming data is collected over a 36-week season from a 2.59 km^2 field.
[11] Acquired through satellite triangulation (e.g., GPS) or very-high resolution satellite radiometry (e.g., NOAA-11).

Table 11 lists maps, sensor data, historical records, and laboratory data that is available to the principal investigators through the Department of Crop Sciences, the Williams field (an experimental field in East Central Illinois), and the Illinois State Water Survey in Champaign, Illinois. The Williams field is situated on a 1-square-mile, or 2.59-square-kilometer, plot and is co-managed by one of the principal investigators.

Map-referenced data may be produced using manual probing, remote sensing, application of computational geometry algorithms, simulation [JK86], or records of crop management. Items shown in *italics* are the result of computation-intensive analysis of measurements (from both on-site and remote sensors). Items shown in **boldface** are typical quantities that a *recommender* (or *decision-support*) system [RV97] can be used to *prescribe*, or recommend, based on other spatiotemporal statistics from these databases.

6.3.3 Other Domains

An important topic that I continue to investigate is the process of automating task decomposition for model selection. I have used similar learning architectures and algorithms for each subproblem in our modular decomposition. I have shown how the quality of generalization achieved by a mixture of classifiers can benefit from the ability to identify the "right tool" for each job. The findings I report here, however, only demonstrate the improvement for a very limited set of real-world problems, and a (relatively) small range of stochastic process models. This needs to be greatly expanded (through collection of much more extensive corpora) to form any definitive conclusions regarding the efficacy of the coarse-grained model selection approach. The relation of model selection to attribute formation and data fusion in time series is an area of continuing research [HR98a]. A key question I will continue to investigate is: how does attribute partitioning-based decomposition support *relevance determination* in a modular learning architecture?

A. Combinatorial Analyses

This appendix briefly presents some illustrative combinatorial results and statistics that are useful in benchmarking components of the learning system and assessing its computational bottlenecks. The primary intractable problem (aside from the training algorithm in the actual supervised learning phase) is attribute partition evaluation. Another important bottleneck is evaluation of composites. Using the state space search formulation introduced in Chapter 2, I show below that the asymptotic running time for partition evaluation is only improved from superexponential to exponential. For attribute-driven problem decompositions, however, I argue that this improvement is of practical significance. Using the metric-based model selection algorithm introduced in Chapter 3, I show how significant savings can be attained, by using approximations of model performance instead of exhaustively testing configurations.

1. Growth of B_n and $S(n,2)$

N	B_n	S(n,2)
1	1	0
2	2	1
3	5	3
4	15	7
5	52	15
6	203	31
7	877	63
8	4140	127
9	21147	255
10	115975	511
11	678950	1023
12	4213597	2047
13	2.76E+07	4095
14	1.91E+08	8191
15	1.38E+09	16383
16	1.05E+10	32767
17	8.29E+10	65535
18	6.82E+11	131071
19	5.83E+12	262142
20	5.17E+13	524287
25	4.64E+18	16777215
50	1.86E+47	5.63E+14
100	4.76E+115	6.34E+29
500	1.61E+843	1.64E+150
1000	–	5.36E+300

Table 12. Bell numbers and number of size-2 partitions as a function of set size *n*.

Table 12 illustrates the growth of the Bell numbers as a function of the set size, n. As noted in Chapter 2, the growth of B_n is $\omega(2^n)$ and $o(n!)$. This makes enumeration of all states in the search space of partitions an infeasible prospect for n larger than about 20. Furthermore, direct evaluation of each state requires several complete experiments (the alternative being to use distributional metrics, as documented in Chapter 5). *Each* of these must run for as long as it takes the training algorithm to converge [Ne96, Hi98]. Finally, to gather enough data points (i.e., statistics on classification accuracy) to differentiate the confidence intervals for model performance may actually require many experiments, if indeed it can be done at all [Gr92, Ko95]. I addressed each of these problems by casting partition evaluation as a heuristic search: (as described in Chapter 2). The complexity is reduced to the maximum breadth (diameter) of the state space. As for the heuristic evaluation function for evaluation of each state, I isolated this subproblem (which is still essential to the success of partition search) and deferred it until the model selection stage (as described in Section 2.4 and Chapter 3).

A limitation of simple heuristic search is that, in order to expand the "frontier" of vertices to be visited (the *OPEN* list), all children of the current candidates (the *CANDIDATES* list as it is called in Section 2.2.1.2) must be evaluated [BF81, Wi92, RN95]. Unfortunately, this means that in the first step of partition search, all of the size-2 partitions must be evaluated. The number of such partitions is $S(n, 2) = 2^{n-1} - 1$. If the heuristic is accurate, this quantity tends to dominate the number of partitions to be evaluated on subsequent iterations of an informed search algorithm (as opposed to breadth-first search, for example). For example, observe that each size-3 partition entails a size-2 split of one subset (of size at most $n - 1$), and so on. Although a heuristic evaluation function can be deceived, a good function will typically not have an expanding frontier (i.e., expand more than $\Theta(S(n, 2))$ members before finding an optimal partition[12]). This can be explicitly enforced by constraining the *CANDIDATES* list to contain a constant-width set of vertices, as in the case of beam search. Kohavi [Ko95, KJ97] uses a recency criterion for termination (i.e., if the *BEST* vertex has not changed in a pre-set number of iterations, stop searching – in this case, do not test any further subdivisions of the current *BEST* partition).

[12] If the heuristic is admissible, this (locally) optim*al* solution is also the (global) optim*um* [Bo??]. Typically, however, we are dealing with neither an admissible heuristic *nor* one that is always close to h*; nontrivial admissible heuristics are hard to keep close to the true value, and vice versa.

As Table 12 also shows, even the first layer of the state space (which must always be expanded) grows exponentially. In practice, my system can handle at least twice as many attributes using partition search as with attribute enumeration. Exponential running time means partition search is swamped out at around 20 attributes on a fast desktop workstation, perhaps 50 for high-performance computers with the highest currently available throughput; by contrast, the current limit is approximately 11 and 21 for naïve enumeration.

Partition Number	Partition Members			Partition Number	Partition Members				
1	1 2 3 4 5			**27**	1 5	2 4	3		
2	1	2 3 4 5		**28**	1 5	2	3 4		
3	2	1 3 4 5		**29**	1	2 3	4 5		
4	3	1 2 4 5		**30**	1	2 4	3 5		
5	4	1 2 3 5		**31**	1	2 5	3 4		
6	5	1 2 3 4		**32**	1 2 3	4	5		
7	1 2	3 4 5		**33**	1 2 4	3	5		
8	1 3	2 4 5		**34**	1 2 5	3	4		
9	1 4	2 3 5		**35**	1 3 5	2	4		
10	1 5	2 3 4		**36**	1 3 4	2	5		
11	2 3	1 4 5		**37**	1 4 5	2	3		
12	2 4	1 3 5		**38**	1	2 3 4	5		
13	2 5	1 3 4		**39**	1	2 3 5	4		
14	3 5	1 2 4		**40**	1	2 4 5	3		
15	3 4	1 2 5		**41**	1	2	3 4 5		
16	4 5	1 2 3		**42**	1 2	3	4	5	
17	1 2	3 4	5	**43**	1 3	2	4	5	
18	1 2	3 5	4	**44**	1 4	2	3	5	
19	1 2	3	4 5	**45**	1 5	2	3	4	
20	1 3	2 4	5	**46**	1	2 3	4	5	
21	1 3	2 5	4	**47**	1	2 4	3	5	
22	1 3	2	4 5	**48**	1	2 5	3	4	
23	1 4	2 3	5	**49**	1	2	3 4	5	
24	1 4	2 5	3	**50**	1	2	3 5	4	
25	1 4	2	3 5	**51**	1	2	3	4 5	
26	1 5	2 3	4	**52**	1	2	3	4	5

Table 13. Partitions of a 5-attribute data set

Table 13 shows partitions of a 5-attribute data set.

2. Theoretical Speedup due to Prescriptive Metrics

The naïve method for selecting a learning technique from a database of learning combinations is to test every configuration. This may, furthermore, involve multiple tests for each pair of combinations in order to judge between them (cf. [Gr92], [Ko95], [KSD96]). Considering only

one combination at a time, the "try and see" method entails $O((r \bullet c)^k)$ tests for r possible choices of learning algorithm, possible choices, c_a possible choices of training algorithm, and c_m possible choices of mixture model. In the current design, $r = 3$, $c_a = 3$, $c_m = 2$, and $c = c_a \bullet c_m = 6$. Because my experiments usually use only one type of training algorithm at a time (as opposed to using the distributional metric to select it), we can suppose that $r = 3$, $c_a = 1$, $c_m = 2$, and $c = c_a \bullet c_m = 2$. For a size-4 partition, however, this still means $6^4 = 1296$ combinations.

More realistically, we can constrain the mixture model, training algorithm, or both to be a function of an entire partition. Because the distributional metrics are computed over the entire partition, this is a natural assumption. There will then be only c choices for learning methods, but still r^k for the learning architecture. The number of combinations to examine is then $O(r^k \bullet c)$, which, while significant smaller, is still a substantial number of experiments ($3^4 \bullet 2 = 162$ in the case of a size-4 partition).

If a 2-D lookup table is used, however, evaluation can be performed on rows and columns of the table independently. This reduces the number of tests to $O(k(r + c_a + c_m))$, or $4 \bullet (3 + 1 + 2) = 24$. The empirical speedup is even more significant, if metric-based model selection is used, because the time to evaluate a single configuration is typically much less than the convergence time for that configuration (especially for MCMC methods [Ne96]). Thus, the organization of learning architectures and methods into a database provides significant computational savings.

3. Factorization properties

Definition 1: A *factorization* of a data set D under an output attribute $\boldsymbol{b}_{lj}$ ($l \geq 0$) is the set of equivalence classes of points in D distinguishable by $\boldsymbol{b}_{lj}$.

For example, Figure 25 shows a factorization of a set of 15 points using two attributes $\boldsymbol{b}_{l-1,1}$ and $\boldsymbol{b}_{l-1,2}$. Each induces a factorization of size 3. $\boldsymbol{b}_{l1}$, their parent in the specialist-moderator tree, induces a factorization of size 7 (because two of the intersections among equivalence classes of the children are empty). This is efficient because $7 > 3 + 3$.

Definition 2: An *inefficient factorization* of D under a nonleaf attribute $\boldsymbol{b}_{lj}$ ($l \geq 1$) is a factorization whose size is less than or equal to the sum of its children's.

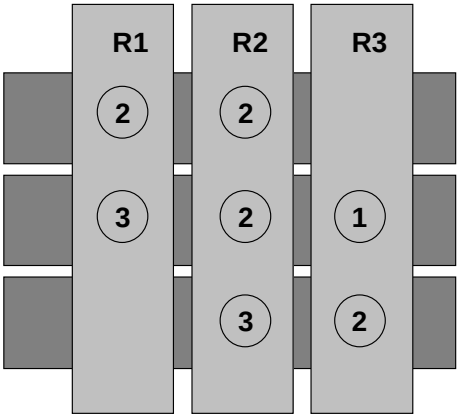

Figure 25. Hypothetical Factorization of a Data Set Using Two Attributes

Definition 3: A *possible factorization* by $\boldsymbol{b_{lj}}$ ($l \geq 1$) is one of the $2^{p_{lj}}$ that $\boldsymbol{b_{lj}}$ can induce under an *arbitrary* data set *D*, where $p_{lj} = \prod_{k=S_l[j]}^{E_l[j]} p_{l-1,k}$ and $p_{0j} = O_{0j}$.

Lemma: The number of possible factorizations by a nonleaf output attribute $\boldsymbol{b_{lj}}$ ($l \geq 1$) that are inefficient is $N_{lj}^{bad} = \sum_{i=0}^{s_{lj}} \sum^{p_{lj}} \sum \sum \sum^{i} \sum$ where $s_{lj} = \sum_{k=S_l[j]}^{E_l[j]} p_{l-1,k}$.

Theorem: Let $f_{l-1,k}$ $(S_l[j] \leq k \leq E_l[j], l \geq 1)$ be a child of f_{lj} and let $2^{p_{l-1,k}}$ denote the number of possible factorizations it induces.

$$\lim_{p_{l-1,S_l[j]} \to \infty, \ldots, p_{l-1,E_l[j]} \to \infty} \frac{\sum N_{lj}^{bad} \sum}{\sum 2^{p_{lj}} \sum} = 0$$

Definition 4: An *orthogonal* factorization of *D* under a nonleaf output $\boldsymbol{b_{lj}}$ ($l \geq 1$) is one whose size is equal to the product of the sizes of its children's.

Property 1: Among factorizations of a data set in any specialist-moderator decomposition, factorization size is maximized in the orthogonal case.

For purposes of generalization, maximizing the number of discriminable classes is not necessarily the goal. However, suppose that a set of overall target classes (such as those found by conceptual clustering using the original attributes) is known. *Given* this set and a hierarchically decomposable model, it is best to dichotomize as cleanly (orthogonally) as possible. This process is subject to constraints of network complexity *and* learning complexity of the induced attributes (i.e., whether the subnetworks can be trained efficiently).

Definition 5: A *perfect hypercubic* factorization of D under $\boldsymbol{b}_{ij}$ is one that is orthogonal and whose descendants' at each level are equal in size.

Table 14 gives statistics on *square* factorizations (perfect hypercubic factorizations using 2 children).

m	n	$N^{bad}(m,n)$	2^{mn}	% inefficient
2	2	16	16	100
3	3	466	512	91
4	4	39203	65536	59.8
5	5	7119516	33554432	21.2
6	6	2241812648	68719476736	3.3

Table 14. Number of possible and inefficient square factorizations.

Property 2: Perfect hypercubic factorizations minimize the sum of factorization size among child attributes, given the factorization size of the parent.

Although minimum total factorization size among children does not guarantee minimum network complexity, our examples below show that it is a good empirical indicator.

This analysis leaves two practical questions to be answered:

1. *What is the empirical likelihood of finding efficient factorizations of D?*

This depends on many issues, the most important being the quality of *F*, the constructive induction algorithm. I consider the case where a good B_0 is already known or can be found by knowledge-based inductive learning [Be90].

2. What is the difficulty of learning a factorization of D even if it is efficient?

The results for the musical tune classification problem, reported in Chapter 5, demonstrate that the experimental difficulty of training a specialist-moderator network on efficient factorizations is lower than that of training a non-modular feedforward or temporal ANN. By "difficulty" I mean *achievable test error given a consistent limit on network complexity and training time.* In future work, I will investigate the computational learning theoretic properties of specialist-moderator networks, but these are beyond the scope of this dissertation.

B. Implementation of Learning Architectures and Methods

This appendix presents salient implementation details for the time series learning architectures, training algorithms, and hierarchical mixture models used in this dissertation.

1. Time Series Learning Architectures

This section defines the underlying mathematical models for the *memory forms* studied, and which are used to populate the database of learning techniques described in Chapter 3. Each memory form corresponds to a row of Table 1 in Chapter 3. The implementation platforms are also briefly summarized.

1.1 Artificial Neural Networks

The artificial neural networks used for experimentation in this dissertation were implemented primarily using *NeuroSolutions v3.00, 3.01,* and *3.02* [PL98], which I used to collect results on temporal ANNs, unless otherwise noted. I implemented wrappers (e.g., for metric-based model selection as described in Section 5.2) and custom automation (e.g., for exhaustive partition evaluation as described in Section 5.3) using *Microsoft Visual C++ 5.0* and *Visual Basic for Applications* under *Windows NT 4.0*. Data preprocessing (encoding, partitioning) and postprocessing (discretization of intermediate outputs for moderator networks, counting the number of correctly classified exemplars) was implemented in C++ (*Microsoft Visual C++* for *Windows NT* and *GNU C++* for *Linux*). In many cases this code was integrated with or built upon that for metric-based model selection and partition evaluation (see Appendix C). Preliminary experiments testing the ability of simple (specifically, Elman) recurrent networks to predict various stochastic processes (such as those generated by Reber grammars and hidden Markov models [RK93, Hs95]) were implemented in *MATLAB* (versions 4 and 5) using the neural networks toolbox.

Adjustment of tunable parameters *other* than attribute partitioning (subset membership for each input) was primarily performed by hand, and automated in a few select cases. I implemented such automation mostly for synthesizing data in evaluation experiments as documented in Chapter 5 and Appendix D, using hybrids of scripting languages such as Visual Basic, the *NeuroSolutions* macro language [PL98], and Perl, along with some standalone C++ programs. Parameter tuning for neural networks consisted of:

1. The number of hidden units, which was tuned by hand to a consistent baseline and normalized (see Section B.3 below) for components of a hierarchical mixture
2. The step size, also tuned by hand[13]
3. The momentum values and time constants (see Section 5.1)

1.1.1 Simple Recurrent Networks

The term simple recurrent network refers to the family of artificial neural networks that contains *recurrent feedback*, or connections from one layer to an earlier one (according to the feedforward data flow). They are called *simple* because the network dynamics do not, in general, provide a facility for adapting the weights (i.e., decay values). The weights are therefore considered constants relative to the training algorithm (but can still be treated as high-level, tunable parameters using a wrapper for the supervised learning component). Other types of recurrent networks, such as *partially* and *fully recurrent* networks, have trainable recurrent weights.

Jordan networks, whose dynamics were first elucidated by Jordan [Jo87], contain recurrent connections from the output to *context elements* with exponential decay. Similarly, Elman networks are recurrent networks with connections from the first hidden layer to the context elements [El90, PL98]. Finally, input recurrent networks are those with connections from input to context elements [RH98]. Input recurrent networks are a type of *moving average* model previously studied under the term *exponential trace memory* [Mo94, MMR97, PL98].

In linear systems the use of the past of the input signal creates what is called the moving average (MA) models. These are best at representing signals that have a spectrum with sharp valleys and broad peaks [BD87, PL98]. The use of past values of the *output* generates a memory form corresponding to *autoregressive* (AR) models. These models are best at representing signals that have broad valleys and sharp spectral peaks [BD87, PL98]. In the case of nonlinear systems, such as neural nets, the MA and AR topologies are nonlinear (NMA and NAR, respectively). The Jordan network is a restricted case of an NAR(1) model, while the input recurrent network is a restricted case of NMA. Elman networks do not have a counterpart in linear system theory.

[13] Experiments with step size adaptation algorithms (such as Delta-Bar-Delta [Ha94, PL98] and exponential adjustment as used in the MATLAB Neural Networks toolbox) showed that extant procedures are generally too insensitive to use as wrappers for performance tuning, on arbitrary time series learning problems.

These simple recurrent network topologies have different processing power, but the question of which one performs best for a given problem is a coarse-grained model selection problem.

Neural networks with context elements can be analytically characterized for the case of linear processing elements, in which case the context elements are equivalent to a very simple lowpass filter [PL98]. A lowpass filter creates an output that is a weighted (average) value of some of its more recent past inputs. In the case of the Jordan context unit, the output is obtained by summing the past values multiplied by the cumulative decay τ^{t-k}, a scalar:

$$y_i(t) = \sum_{k=0}^{t} x_i(t)\tau^{t-k}$$

The x_i value is the i^{th} input in the case of input recurrent networks, output from i^{th} unit in the first hidden layer in the case of Elman networks, and output from the i^{th} unit from the output layer in the case of Jordan networks.

I conducted preliminary experiments using Elman, Jordan, and input recurrent networks on the musical tune classification and the crop condition monitoring data sets. The results indicate that for *unpartitioned* data (i.e., non-modular learning), the input recurrent network type tends to outperform Elman and Jordan networks of comparable complexity. This suggests that in these particular cases, the data are most effectively assumed to originate from MA processes than from AR or Elman-type processes. More specifically, they are more strongly attuned to the *exponential trace* memory form. In the crop monitoring test bed, however, non-exponential patterns can be observed in visualizations such as the phased correlogram (Figures 13 and 14, in Chapter 5). The positive learning results from the multi-strategy, hierarchical mixture model (pseudo-HME of TDNN and input recurrent specialists with a Gamma network moderator) provides evidence that these patterns conform to different memory forms (in this case, two different MA processes – one exponential, one non-exponential).

1.1.2 Time-Delay Neural Networks

Time-delay neural networks (TDNNs) are an alternative type of AR model that expresses future values of ANN elements as a linear recurrence over past values [LWH90, Mo94]. This is implemented using memory buffers at the input and hidden layers (associated with the units rather than weights). The *delay* represents the number of discrete time units of memory that the

model can represent, a quantity also known as *depth* [Ha94, Hs95, MMR97, PL98]. TDNNs can be thought of as having as many "copies" of a hidden or input unit as there are delays [Ha94]. Data is propagated from one copy to the next in a cascaded (serial) delay line; the acronym *TDNN* has therefore also been used to mean *tapped delay-line neural network* [MMR97, PL98]. The TDNN architecture has the simplest mathematical description in terms of convolutional codes, as given in Appendix C.

1.1.3 Gamma Networks

Gamma networks (ANNs whose elements are generalized temporal units called *Gamma memories*) are a type of ARMA model [DP92, Mo94, PL98]. They express both *depth* (through a delay-line-based mechanism that represents the MA part of the pattern-generating process) and *resolution* (through an exponential decay-based mechanism that represents the AR part) [Mo94, MMR97]. The combination of both tapped delay lines and exponential traces in a Gamma network makes the model more general and flexible, but also increases the number of degrees of freedom. The nonlinear dynamics of a Gamma network are extremely complex – relatively more so than for a comparably-sized SRN or TDNN [DP92]. Gamma networks commonly *require* fewer trainable parameters to acquire a general ARMA process than a pure AR or MA model; however, the added complexity means that they also tend to require more updates to converge [DP92, Mo94, MMR97]. Furthermore, the complexity is aggravated when global optimization is used; the extant research on ARMA models suggests that extension to Bayesian learning, for example, poses difficulties [PL98, Wa98]. In future work, I intend to investigate integrative models (ARMA and ARIMA) [Ch96] with variational and MCMC learning [Ne96, Jo97a], especially in the capacity of data fusion.

1.2 Bayesian Networks

1.2.1 Temporal Naïve Bayes

The implementation of the naïve Bayes learning architecture is based entirely on that given in *MLC++*, Kohavi *et al*'s machine learning library in C++ [KS96, KSD96]. Though I installed tested under both the *Microsoft Windows NT 4.0* and *RedHat Linux 5.1* platforms, I conducted most experiments using the Linux version, for efficiency's sake. In several exploratory experiments, I constructed artificial features to test the capability of discrete naïve Bayes to acquire simple memory forms (such as M-of-N and parity *through time*). The results are reported in Appendix D.

1.2.2 Hidden Markov Models

My original implementation of HMM learning used the Viterbi algorithm [Le89, CLR90, BM94] and was written in MATLAB [Hs95]; a port to GNU C++ was used for additional experiments [Hs95]. Parameter learning with gradient and EM learning rules can be performed using an ANN representation (wherein HMM parameters are encoded in ANN weights, then interpreted after training). This dualization was documented by Bourlard and Morgan [BM94] and is similar to several discussed by Ackley, Hinton, and Sejnowski [AHS85], Neal [Ne92, Ne93], and Myllymäki [My95, Hs97].

2. Training Algorithms

This section defines the training algorithms for the types of *target distribution* studied, and which are used to populate the database of learning techniques. Each training algorithm corresponds to a single column (subheading of a learning method) of Table 1 in Chapter 3.

2.1 Gradient Optimization

Gradient learning is a basic optimization technique [BF81, Wi93] adapted to parameter estimation in network models [MP69, MR86]. Its advantages are that it is simple to implement and highly general (over families of probabilistic networks – i.e., learning architectures). Its disadvantages are that it is, in general, slow to converge [MR86, JJ94] and, by definition, susceptible to local optima [MR86, Ha94, Bi95]. The implementations tested in this dissertation were implemented first using *MATLAB 4* (using the Elman network code in the Neural Network toolbox), then in *NeuroSolutions* [PL98]. The exact gradient learning algorithm used was *backpropagation with momentum* [Ha94, PL98], though experiments with simple Step, Delta-Bar-Delta, and Quickprop learning rules were conducted. These generally resulted in worse performance on the data sets tested, which were generally heterogeneous time series or subsets thereof. Batch update was used in all cases, as incremental (online) learning produced generally poorer performance as well.

2.2 Expectation-Maximization (EM)

As mentioned in Section B.1.2 above, the Viterbi algorithm (a graph optimization algorithm for probabilistic network learning [Le89, CLR90]) was implemented for experiments on HMMs.

As EM does, the Viterbi algorithm estimates the maximum likelihood path through a state transition model. The primary difference is that the Viterbi algorithm is designed for the backward problem (maximum likelihood estimation) and learning by iterative refinement requires an update step. In EM, this is called the "maximize" step (the acronym *EM* is also taken to stand for *Estimate-and-Maximize*) [Le89]. Variants of Viterbi used to test the efficacy of naïve Bayes (the learning rule, not the learning architecture) on simple HMMs were implemented using C++ and MS Excel [Hs95]. Experiments using this combination on the crop condition monitoring data set indicated that gradient learning (MAP estimation) was preferable in that case.

2.3 Markov chain Monte Carlo (MCMC) Methods

MCMC refers to a family of algorithms that estimates network parameters by integrating over the conditional distribution of models given observed data [Ne96]. This integration is performed using Monte Carlo techniques (also known as *random sampling*) and the distribution sampled from is a Markov chain of network configurations (states of a large stochastic model). The MCMC algorithm implemented in this dissertation is the Metropolis algorithm for simulated annealing, documented in [KSV83]. I implemented simulated annealing using large-scale modifications to *NeuroSolutions* breadboards (network and training algorithm specifications) [PL98]. These modifications were based on Neal's implementation of the Metropolis algorithm [Ne96, Fr98] and required extensive use of the dynamic link library (DLL) integration feature of *NeuroSolutions* [PL98].

3. Mixture Models

This section defines the mathematical models for the *hierarchical mixture models* studied, and which are used to populate the database of learning techniques. Each mixture type corresponds to a group of 3 columns (main heading of a learning method) of Table 1 in Chapter 3. Together with the choice of training algorithm, the choice of mixture model determines the *learning method* to be used (as prescribed by the distributional metrics). This specification, plus that of the learning architecture (as prescribed by the architectural metrics), forms a *composite* as described in Section 3.1 and Appendix C.

3.1 Specialist-Moderator (SM)

The specialist-moderator network was implemented using *NeuroSolutions* 3.0, 3.01, and 3.02 [PL98] with data fusion being performed first by hand, then using automation scripts written in *Visual Basic for Applications* (*VBA*). Specialist outputs were typically passed through a winner-take-all filter that converted real-value intermediate output to 1-of-C coded [Sa98] (also known as a locally coded [KJ97]) output. This provides input to the moderator that is sparse, discrete, and conforms to the construction algorithm for SM networks, **Select-Net**, given in Section 4.3. Experiments on the musical tune classification data set showed that winner-take-all prefiltering for moderator networks resulted in better performance (classification accuracy using the same-sized moderator networks) than raw intermediate outputs from the specialists.

3.2 Hierarchical Mixtures of Experts (HME)

The hierarchical mixture model (a variant of the HME architecture of Jordan et al [JJB91, JJNH91, JJ94]) was also implemented using *NeuroSolutions* 3.0, 3.01, and 3.02 with data fusion being performed first by hand, then using automation scripts written in *Visual Basic for Applications* (*VBA*). Experiments using winner-take-all prefiltering and unfiltered intermediate targets were inconclusive for synthetic data sets and for the crop condition monitoring data set. My conjecture is that for most HME-type applications, unfiltered intermediate data will tend to perform slightly better, because this is most consistent with the original design of the gating networks [JJ94].

C. Metrics

This appendix gives empirical and mathematical background for the architectural and distributional metrics, presents the design rationale for each one to show how it was derived, and explains how the individual metrics are computed.

1. Architectural: Predicting Performance of Learning Models

As explained in Section 1.1 and Section 3.2, the primary criterion used to characterize a stochastic process in my multi-strategy time series learning system is its *memory form*.

1.1 Temporal ANNs: Determining the Memory Form

To determine the memory form for temporal ANNs, I make use of two properties of statistical time series models. The first property is that the temporal pattern represented by a memory form can be described as a *convolutional code*. That is, past values of a time series are stored by a particular type of recurrent ANN, which transforms the original data into its internal representation. This transformation can be formally defined in terms of a *kernel function* that is convolved over the time series. This convolutional or functional definition is important because it yields a general mathematical characterization for individually weighted "windows" of past values (time delay or *resolution*) and nonlinear memories that "fade" smoothly (attenuated decay, or *depth*) [DP92, Mo94, PL98]. It is also important to metric-based model selection, because it concretely describes the transformed time series that we should evaluate, in order to compare memory forms and choose the most effective one. The second property is that a transformed time series can be evaluated by measuring the change in *conditional entropy* [CT91] for the stochastic process of which the training data is a sample. The entropy of the next value conditioned on past values of the *original* data should, in general, be higher than that of the next value conditioned on past values of the *transformed* data. This indicates that the memory form yields an improvement in predictive capability, which is ideally proportional to the expected performance of the model being evaluated.

1.1.1 Kernel Functions

Given an input sequence $\mathbf{x}(t)$ with components $\{\hat{\mathbf{x}}_i(t), 1 \le i \le n\}$, its convolution $\hat{\mathbf{x}}_i(t)$ with a kernel function $c_i(t)$ (specific to the i^{th} component of the model) is defined as follows:

$$\hat{\mathbf{x}}_i(t) = \sum_{k=0}^{t} c_i(t-k)\,\mathbf{x}(k)$$

(Each **x** or $\mathbf{x}_i$ value contains all the attributes in *one subset* of a partition.)

The memory form for a recurrent ANN is determined by its kernel function. For tapped delay-line memories (time-delay neural networks, or TDNNs), the kernel function is:

$$c_i(j) = \begin{cases} 1 & \text{for } j = i, 1 \le i \le d \\ 0 & \text{otherwise} \end{cases}$$
$$\hat{\mathbf{x}}_i(t) = \mathbf{x}(t-i), 1 \le i \le d$$

This kernel function is inefficient to compute, as a tapped-delay line can be implemented in linear space and linear time without convolution (which takes quadratic time in the straightforward implementation). The above characterization, however, is still useful because it captures the notion of *resolution*. TDNNs are *high-resolution, low-depth* models: they are flexible, nonlinear AR models that degrade totally when the required depth exceeds the number of memory state variables (delay buffer or "window" width) [Mo94, PL98].

For exponential trace memories (input recurrent networks), the kernel function is:

$$c_i(j) = (1-\mu_i)\mu_i{}^j$$
$$\hat{\mathbf{x}}_i(t) = (1-\mu_i)\mathbf{x}_i(t) + \mu_i \hat{\mathbf{x}}_i(t-1)$$

This kernel function expresses *depth* by introducing a "decay variable" or "exponential trace" $\mu_i \in [-1,1]$, for every model component. IR networks are *high-depth, low-resolution* models: they are flexible, nonlinear MA models that degrade gradually (how slowly depends on the decay variables, which can be adapted based on the training data) as the required depth grows [Mo94, PL98]. IR networks do *not* scale up in complexity with the required information content for successive elements of the input sequence; that is, they can store information further into the past, but this information degrades incrementally because it is stored using the same state variables.

Finally, the kernel function for gamma memories is:

$$c_i(j) = \begin{cases} \binom{j}{l_i} (1-\mu_i)^{l_i+1} \mu_i^{\,j-l_i} & \text{if } j \geq l_i \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{\mathbf{x}}_{\mu,j}(t) = (1-\mu)\hat{\mathbf{x}}_{\mu,j-1}(t) + (1-\mu)\hat{\mathbf{x}}_{\mu,j-1}(t)$$

$$\hat{\mathbf{x}}_{\mu,-1}(t) = \mathbf{x}(t+1) \quad \text{for } t \geq 0, \text{ and } \hat{\mathbf{x}}_{\mu,j}(0) = 0 \quad \text{for } j \geq 0$$

This kernel function expresses both resolution and depth, at a cost of much higher theoretical and empirical complexity (in terms of the number of degrees of freedom, convergence time, and the additional computation entailed by this more complex function). Gamma memories are even more flexible nonlinear ARMA models that trade this complexity against the ability to learn both exponential traces $\mu_i \in [0,1]$ and tapped delay-line weights $l_i \in \mathbf{N}$.

1.1.2 Conditional Entropy

The entropy *H(X)* of a random variable *X*, the joint entropy *H(X,Y)* of two random variables *X* and *Y*, and conditional entropy *H(X|Y)* are defined as follows [CT91]:

$$\begin{aligned} H(X) &=_{def} -E[\lg p(X)] \\ H(X,Y) &=_{def} -E[\lg p(X,Y)] \\ H(X \mid Y) &=_{def} \sum_{x \in \mathcal{X}} p(x) H(Y \mid X = x) \\ &= -E_{p(X,Y)}[\lg p(X \mid Y)] \\ &= H(X,Y) - H(Y) \qquad \text{(chain rule)} \end{aligned}$$

For a stochastic process (time-indexed sequence of random variables) *X(t)*, we are interested in the conditional entropy of the next value given earlier ones. This can be written as:

$$\begin{aligned} H_d &=_{def} H(X(t) \mid X_i(t), 1 \leq i \leq d) \\ &=_{def} H(X(t) \mid X_1(t), \ldots, X_d(t)) \end{aligned}$$

To measure the improvement due to convolution with a kernel function with *d* components, we can compute $\hat{H}_d$:

$$\hat{H}_d =_{def} H\left(X(t) \mid \hat{X}_i(t), 1 \leq i \leq d\right)$$

where $\hat{X}_i(t)$ is as defined above. An additional refinement that allows us to evaluate specific *subsets* of input data (recall that architectural metrics are used to determine the memory form for a single *subset* within an attribute partition) is to define H_d^s and $\hat{H}_d^s$ for a subset *s*:

$$H_d^s =_{def} H\left(X^s(t) \mid X_i^s(t), 1 \le i \le d\right)$$
$$\hat{H}_d^s =_{def} H\left(X^s(t) \mid \hat{X}_i^s(t), 1 \le i \le d\right)$$

Given a kernel function for a candidate learning architecture, I then define the architectural metric as follows:

$$M_R = \frac{H_d^s}{\hat{H}_d^s}$$

for a recurrent ANN of type $R \in \{TDNN, SRN, GAMMA\}$. Note that, because H_d^s is identical to the entropy of a depth-*d* tapped delay-line convolutional code (for the training data), the metric M_{TDNN} will always have a baseline value of 1. I adopt this convention merely to simplify the normalization process.

A final note: an assumption I have made here is that predictive capability is a good indicator of performance (classification accuracy) for a recurrent ANN. Although the merit of this assumption varies among time series classification problems [GW94, Mo94], I have found it to be reliable for the types of time series I have studied.

1.2 Temporal Naïve Bayes: Relevance-Based Evaluation Metrics

The memory form for a general naïve Bayesian classifier [Ko95, KSD96, KJ97], network [Pe88], or rule set [KSD96] cannot be defined as a convolutional code, so predicting the effectiveness of naïve Bayes (or approximation by MCMC methods [Hr90, Ne93]) is not as straightforward. In future work, I will investigate the application of relevance measures [He91, KJ97] to evaluation of temporal naïve Bayesian networks.

1.3 Hidden Markov Models: Test-Set Perplexity

The memory form for an *arbitrary* HMM is not easy to define as a convolutional code. In certain linear models, such as first-order HMMs for speech recognition [Le89], algorithms such as the Viterbi algorithm [Vi67] can be used to *decode the hidden state sequence* (i.e., solve the search-based inference [Pe88], or *backward*, problem [Le89]). I conducted preliminary experiments (reported in [Hs95]) on HMM parameter learning using the EM algorithm [DLR77, Le89] and gradient search by dualization to simple recurrent networks [BM94]. The results suggest that a good indicator of problem difficulty for a particular HMM architecture is the test-set perplexity [Le89]:

$$\frac{1}{n} \log p(x_1, x_2, \ldots, x_n)$$

which is an estimate of the true perplexity Q for observed data. Q can be defined in terms of a finite state model (an HMM [Le89, Ra90] or Reber diagram [RK96]) with states $s(t)$ and observations X(t):

$$Q(x(t) \mid s(t)) = 2^{H(x(t) \mid s(t))}$$
$$H(X(t) \mid s(t)) = \sum_{x \in X} p(x(t) \mid s(t)) \lg[p(x(t) \mid s(t))]$$

or over a state transition grammar G (defined over these states):

$$Q(G) = 2^{H(G)}$$
$$H(G) = \sum_{q} \pi(s(t)) H(X(t) \mid s(t))$$

The measure based upon G is defined over symbols (associated with each transition in an HMM) and is referred to as the *per-word* perplexity in speech recognition [Le89]. In future work, I will investigate the application of empirical perplexity measures [Le89, Ra90] to evaluation of HMMs for time series learning. The principle behind this approach is that, just as the ratio of conditional entropy for a convolutional code is a good indicator of predictive capability for a recurrent ANN model, so is the perplexity a good indicator of difficulty for a time series learning problem, *given a particular parametric model*. Given specific topologies of HMMs [Le89, Ra90, BM94] or partially observable Markov decision processes [BDKL92, DL95], these information theoretic measures indicate how appropriate each model is for the training data.

2. Distributional: Predicting Performance of Learning Methods

The learning methods being evaluated are: the hierarchical mixture model used to perform multi-strategy learning in the integrated, or composite, learning system, and the training algorithm used. This section presents the metrics for each.

2.1 Type of Hierarchical Mixture

The expected performance of a hierarchical mixture model is a *holistic* measurement; that is, it involves all of the subproblem definitions, the learning architecture used for each one, and even the training algorithm used. It must therefore take into account at least the subproblem definitions. I designed distributional metrics to evaluate only the subproblem definitions. This criterion has three benefits: first, it is consistent with the holistic function of mixture models; second, it is minimally complex, in that it omits less relevant issues such as the learning architecture for each subproblem from consideration; and third, it measures the quality of an attribute partition. The third property is very useful in heuristic search over attribute partitions: the distributional metric can thus serve double duty as an evaluation function for a partition (given a mixture model to be used) and for mixture model (given a partitioned data set). As a convention, I commit the choice of *partition* first, then the mixture model and training algorithm, then the learning architectures for each subset, with each selection being made subject to the previous choices.

2.1.1 Factorization Score

The distributional metric for specialist-moderator networks is the *factorization score.* This is an empirical measure of how evenly the learning problem is modularized; it is not specific to time series data. The score is a penalty function whose magnitude is proportional to the deviation from *perfect hypercubic* factorization. In Appendix A, a factorization is defined for a locally coded target $\boldsymbol{b}_{lj}$ ($l \geq 0$). $\boldsymbol{b}_{lj}$ is formed through cluster definition using a subset $\boldsymbol{a}_{lj}$ of a partition at level *l;* that is, the set of distinguishable classes depends on the *restricted view* through a subset of the original attributes. We can therefore characterize the restricted view by measuring the factorization size for an attribute subset. The most straightforward way to do this is through a naïve cluster definition algorithm that works as follows

Given: a set of overall target classes and an attribute partition

1. Sweep through the training data once for every subset.
2. If any two exemplars occur such that the same input (restricted to the attributes in the subset) is mapped to different output classes, *merge* the equivalence classes for these two output classes.

This algorithm is best implemented using a *union-find* data structure as described in Chapter 22 of Cormen, Leiserson, and Rivest [CLR90]. I implemented a union-find-based version of this algorithm, which requires less than half the running time required for the HME metric (described in Section 2.1.2 below) for small numbers of input attributes. As Table 4 in Chapter 5 shows, however, performance tends to be determined by memory consumption, with thrashing becoming a bottleneck for as few as 8 to 10 attributes.

If the number of distinguishable output classes for each subset $\boldsymbol{a}_i$, $1 \le i \le k$, is o_i, then all o_i are equal in the perfect hypercubic factorization. Let the product of o_i be N:

$$M_{FS} = -\sum_{i=1}^{k} \left| \lg \frac{\sum o_i}{\sqrt[k]{N}} \right|$$

$$N = \prod_{i=1}^{k} o_i$$

The metric imposes a penalty on every factorization (belonging to a single subset) that deviates from the "ideal case" [RH98]. For example, suppose a set of attributes is partitioned into three subsets, whose factorization sizes are 6, 6, and 6. Then N = 216 and M_{SM} = 0. If, for a different size-3 partition, the factorization sizes are 2, 18, and 6, N = 216, but $M_{SM} = -2 \lg 3 \approx -3.17$.

2.1.2 Modular Mutual Information Score

The distributional metric for HME-type networks is the *modular mutual information score*. This score measures mutual information across subsets of a partition [Jo97b]. It is directly proportional to the conditional mutual information of the desired output given each subset *by itself* (i.e., the mutual information between one subset and the target class, *given* all other subsets). It is inversely proportional to the difference between joint and total conditional mutual

information (i.e., shared information among all subsets). I define the first quantity as I_i for each subset $\boldsymbol{a}_i$, and the second quantity as I_{V} for an entire partition.

First, the *Kullback-Leibler distance* between two discrete probability distributions *p(X)* and *q(X)* is defined [CT91] as:

$$D(p \| q) =_{def} E_p \left[\lg \frac{p(X)}{q(X)} \right]$$
$$= \sum_{x \in \mathcal{X}} p(x) \lg \frac{p(x)}{q(x)}$$

The mutual information between discrete random variables *X* and *Y* is defined [CT91] as the Kullback-Leibler distance between joint and product distributions:

$$\begin{aligned}
I(X;Y) &=_{def} D(p(x,y) \| p(x)p(y)) \\
&= E_{p(x,y)} \left[\lg \frac{p(X,Y)}{p(X)p(Y)} \right] \\
&= \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p(x,y) \lg \frac{p(x,y)}{p(x)p(y)} \\
&= \sum_{x,y} p(x,y) \lg \frac{p(x \mid y)}{p(x)} \\
&= -\sum_{x,y} p(x,y) \lg p(x) + \sum_{x,y} p(x,y) \lg p(x \mid y) \\
&= -\sum_{x} p(x) \lg p(x) - \left[\sum_{x,y} p(x,y) \lg p(x \mid y) \right] \\
&= H(X) - H(X \mid Y) \\
&= H(X) + H(Y) - H(X,Y) \qquad \text{(chain rule)} \\
&= H(Y) - H(Y \mid X)
\end{aligned}$$

The conditional mutual information of *X* and *Y* given *Z* is defined [CT91] as the change in conditional entropy when the value of *Z* is known:

$$I(X;Y|Z) =_{def} H(X \mid Z) - H(X \mid Y, Z)$$
$$= H(Y \mid Z) - H(Y \mid X, Z)$$

I now define the *common information* of *X*, *Y*, and *Z* (the analogue of *k*-way intersection in set theory, except that it can have negative value):

$$\begin{aligned} I(X;Y;Z) &=_{def} I(X;Y) - I(X;Y \mid Z) \\ &= I(X;Z) - I(X;Z \mid Y) \\ &= I(Y;Z) - I(Y;Z \mid Y) \\ &= I(X;Z) - I(X;Z \mid Y) \\ &= I(Y;Z) - I(Y;Z \mid Y) \end{aligned}$$

The idea behind the modular mutual information score is that it should reward high conditional mutual information between an attribute subset and the desired output given other subsets (i.e., *each expert subnetwork will be allotted a large share of the work*). It should also penalize high common information (i.e., the gating network is allotted more work relative to the experts). Given these dicta, we can define the modular mutual information for a partition as follows:

$$\begin{aligned} I(\mathbf{X};\mathbf{Y}) &=_{def} D(p(x_1, x_2, \ldots, x_n, y) \| \; p(x_1)p(x_2)\ldots p(x_n)p(y)) \\ \mathbf{X} &= \{\mathbf{X}_1, \ldots, \mathbf{X}_k\} \\ \bigcap_{i=1}^{k} \mathbf{X}_i &= \varnothing \\ \bigcap_{i=1}^{k} \mathbf{X}_i &= \{X_1, X_2, \ldots, X_n\} \end{aligned}$$

which leads to the definition of I_i (modular mutual information) and I_∇ (modular common information):

$$\begin{aligned} I_i &=_{def} I(\mathbf{X}_i;\mathbf{Y} \mid \mathbf{X}_{\neq i}) \\ &=_{def} H(\mathbf{X};\mathbf{Y}) - H(\mathbf{X}_i \mid \mathbf{Y}, \mathbf{X}_1, \ldots, \mathbf{X}_{i-1}, \mathbf{X}_{i+1}, \ldots, \mathbf{X}_k) \\ I_\nabla &=_{def} I(\mathbf{X}_1;\mathbf{X}_2;\ldots;\mathbf{X}_k;\mathbf{Y}) \\ &=_{def} I(\mathbf{X};\mathbf{Y}) - \sum_{i=1}^{k} I_i \end{aligned}$$

Because the desired metric rewards high I_i and penalizes high I_∇, we can define:

$$\begin{aligned} M_{MMI} &= \left(\sum_{i=1}^{k} I_i\right) - I_\nabla \\ &= \left(\sum_{i=1}^{k} I_i\right) - I(\mathbf{X};\mathbf{Y}) - \sum_{i=1}^{k} I_i \\ &= 2\left(\sum_{i=1}^{k} I_i\right) - I(\mathbf{X};\mathbf{Y}) \end{aligned}$$

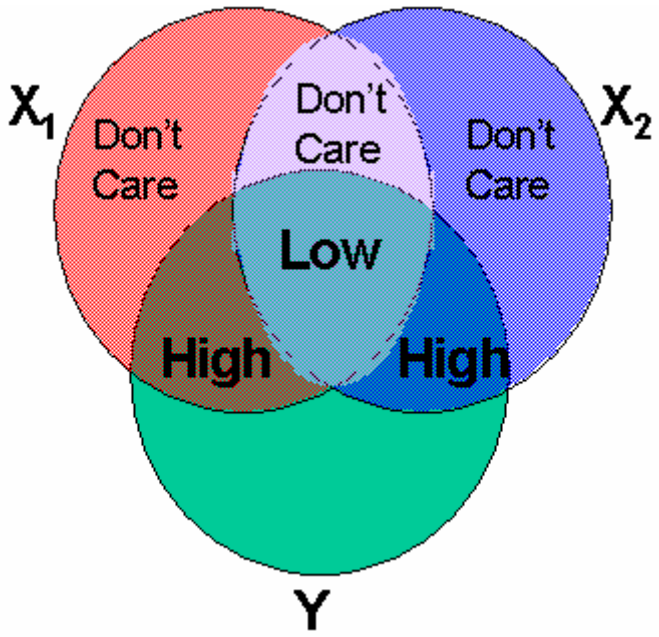

Figure 26. Modular mutual information score for a size-2 partition

Figure 26 depicts the modular mutual information criterion for a partition with 2 subsets $\mathbf{X_1}$ and $\mathbf{X_2}$, where **Y** denotes the desired output.

2.2 Algorithms

The architectural metrics highlight the strengths of each learning architecture by estimating the information gain from a memory form, and the distributional metrics for hierarchical mixture models highlight the strength of each organization by estimating the distribution of work. My preliminary design for distributional metrics for algorithms similarly attempts to estimate the benefits of using a particular type of local or global optimization. The algorithms studied in this dissertation include gradient (local optimization or delta-rule) learning, the EM algorithm (another type of local optimization), and MCMC methods (global stochastic optimization or Bayesian inference), though I have concentrated primarily on gradient algorithms. As for the TDNN architecture, I use gradient learning as a baseline, so its metric can be considered a constant (and need not be computed).

2.2.1 Value of Missing Data

A prototype distributional metric I considered for the EM algorithm is similar to a value-of-information (VOI) measure [RN95] that measures the expected information gain from

interpolation of missing data. The design rationale is that EM is the only local optimization algorithm available that can interpolate missing data, and should therefore be used when there is enough data missing for its approximation to be worth while. This metric has not yet been fully developed or evaluated, because VOI is a nonnegative measure, while EM does is not guaranteed to achieve improved learning through missing data estimation [DLR77, Ne93].

2.2.2 Sample Complexity

Finally, the distributional metric for MCMC methods (specifically, the Metropolis algorithm for simulated annealing [KGV83, Ne93, Ne96]) is based on the frequency of local optima. Sample complexity estimation is used in *convergence analysis* for MCMC methods [Gi96]. This metric has not yet been fully developed or evaluated. Some preliminary comparative experiments using gradient and MCMC learning, however, have shown that short-term convergence analysis methods, such as learning speed curves [Ka95, Pe97] can provide quantitative indicators of the necessity of global optimization.

D. Experimental Methodology

This appendix describes the experimental design for system evaluation, both component-wise and integrated, and the results of some additional experiments that support the ones described in Chapters 5 and 6.

1. Experiments using Metrics

My experimental approach to metric-based model selection and its evaluation builds on two research applications that I have investigated: selection of compression techniques for heterogeneous files, and selection of learning techniques (architectures, mixture models, and training algorithms) for heterogeneous time series. The latter application of metric-based model selection, which I refer to in this dissertation, as *composite learning*, is described in Chapter 3. This section reports some additional relevant findings from the two research efforts.

1.1 Techniques and Lessons Learned from Heterogeneous File Compression

Heterogeneous files are those that contain multiple types of data such as text, image, or audio. We have developed an experimental data compressor for that outperforms commercial, general-purpose compressors on heterogeneous files [HZ95]. It divides a file into fixed-length segments and empirically analyzes each (cf. [Sa89, HM91]) for its *file type* and dominant *redundancy type*. For example, *dictionary* algorithms such as Lempel-Ziv coding are most effective with frequent repetition of strings; *run length encoding*, on long runs of bits; and *statistical* algorithms such as *Huffman coding* and *arithmetic coding*, when there is nonuniform distribution among characters. These correspond to our *redundancy metrics*: string repetition ratio, average run length, and population standard deviation of ordinal character value. The normalization function over these metrics is calibrated on a corpus of homogeneous files. Using the metrics and file type, our system predicts, and applies, the most effective algorithm and update (e.g., paging) heuristic for the segment. In experiments on a second corpus of heterogeneous files, the system selected the best of the three available algorithms on about 98% of the segments, yielding significant performance wins on 95% of the test files [HZ95].

1.2 Adaptation to Learning from Heterogeneous Time Series

The analogy between compression and learning [Wa72] is especially strong for technique selection from a database of components. Compression algorithms correspond to network architectures in our framework; heuristics, to applicable methods (mixture models, learning algorithm, and hyperparameters for Bayesian learning). Metric-based file analysis for compression can be adapted to technique selection for heterogeneous time series learning. To select among network architectures, we use indicators of temporal patterns typical of each; similarly, to select among learning algorithms, we use predictors of their effectiveness. The analogy is completed by the process of segmenting the file (corresponding to problem decomposition by aggregation and synthesis of attributes) and concatenation of the compressed segments (corresponding to fusion of test predictions).

The compression/learning analogy also provides some guidelines for metric calibration. [HZ95] describes how multivariate Gamma distributions are empirically fitted for a homogeneous corpus of 50 representative files, in order to select the algorithm that corresponds to the dominant redundancy type. A similar nonlinear approximation procedure is applied to normalize architectural and distributional metrics (separately) for comparison purposes. Note that normalization is not needed when distributional metrics are being used to evaluate attribute partitions, even though these are the same metrics used to select hierarchical mixture models.

2. Corpora for Experimentation

This section briefly describes the collection and synthesis of representative test beds for testing specific learning components, isolated aspects of the composite learning system, and the overall system.

2.1 Desired Properties

As explained in Sections 1.1.4 and 1.4.3, this dissertation focuses on decomposable learning problems defined over heterogeneous time series. To briefly recap, a heterogeneous time series is one containing data from multiple sources [SM93], and typically contains different embedded temporal patterns (which can be formally characterized in terms of different memory forms [Mo94]). These sources can therefore be thought to correspond to different "pattern-generating" stochastic processes. A decomposable learning problem is one for which multiple subproblems

can be defined by systematic means (possibly based on heuristic search [BF81, Wi93, RN95, KJ97] or other approximation algorithms [CLR90]). Some specific properties that characterize most kinds of heterogeneous and decomposable time series, and are typically of interest for real-world data, are as follows:

1. *Heterogeneity*: multiple physical processes for which a stochastic process model is known, hypothesized, or can be hypothesized and tested
2. *Decomposability*: a known or hypothesized method for isolating one or more of these processes (often published in the literature of the application domain)
3. *Feasibility*: evidence that this process is reasonably "homogeneous" (in the ideal case, evidence that all the embedded processes are homogeneous)

These properties are present to some degree in the musical tune classification and crop condition monitoring test beds. They can also be simulated in synthetic data, and I have done so to a realistic extent.

2.1.1 Heterogeneity of Time Series

The crop condition monitoring test bed [HGL+98, HR98b] is heterogeneous in that:

1. Meteorological, hydrological, physiological, and agricultural processes represent highly disparate sources of data.
2. These processes are reflected in the observable phenomena (weather statistics, subjective estimates of condition) through different stochastic processes (see Section 5.1.2).
3. The scale and structure of spatiotemporal statistics varies greatly: temporal granularity, spatial granularity, and proportion of missing data all fluctuate from attribute to attribute.[14]

The musical tune classification test bed [HR98a, RH98] is heterogeneous in that:

[14] In this dissertation, I have applied simple averaging and downsampling methods to deal with this aspect of heterogeneity for this test bed. Adaptation to scale and structure of large-scale geospatial data, however, is an important topic for future work whereby this research may be refined and extended.

1. The signal preprocessing transforms produce training data that originates from different "sources" (algorithms) and is inherently multimodal. (There is also a natural embedding of the ideal attribute partition based on these transforms.)
2. The processes that each transform "extracts" are typically very different in terms of signal waveshape [RH98] and therefore evoke *different memory forms.*

2.1.2 Decomposability of Problems

The crop condition monitoring test bed [HGL+98, HR98b] is decomposable in that:

1. As the phased correlograms in Chapter 5 indicate, the memory forms manifest in different components of the time series (typically different weeks of the growing season and different magnitudes). The patterns also manifest to a certain extent within different attributes, although this effect (which prescribes the specialist-moderator network) is weaker than the "load balancing" effect.
2. As the comparative experiment using SRNs, TDNNs, and multilayer perceptrons (feedforward ANNs) and the pseudo-HME fusion experiments show, the embedded patterns can be isolated. Furthermore, use of different memory forms tends to distribute the computational workload.

The musical tune classification test bed [HR98a, RH98] is decomposable in that:

1. The problem is inherently "factorizable" as defined in Appendix A.
2. The factorizations lend themselves well to separate SRNs (in this case, the same species: input recurrent specialists and moderators).

2.2 Synthesis of Corpora

The synthesis of experimental corpora also emphasizes heterogeneity and decomposability, but additionally focuses on typically hard problems that have traditionally been mitigated by the use of constructive induction [Gu91, Do96, Io96, Pe97]. An example of this is the *modular parity* problem, a member of the *XOR/parity* family that is often used to demonstrate the limitations of certain inducers [Gu91, Pe97], most (in)famously the single-layer perceptron

[MP69]. Modular parity simply defines the target concept as a combination (Cartesian product) of parity functions defined on each subset, i.e.:

$$\begin{aligned}\mathbf{Y} &= \prod_{i=1}^{k} \mathbf{Y}_i \\ &= \mathbf{Y}_1 \times \mathbf{Y}_2 \times \ldots \times \mathbf{Y}_k \\ \mathbf{Y}_i &= X_{i1} \oplus X_{i2} \oplus \ldots \oplus X_{in_i} \\ X_{ij} &\in \mathbf{H} \equiv \{0,1\}\end{aligned}$$

2.3 Experimental Use of Corpora

I use the synthetic and real-world test data to: calibrate functions for metric normalization; experiment with metrics and learning components (especially mixture models); and evaluate partition search as compared to exhaustive enumeration.

2.3.1 Fitting Normalization Functions

Normalization functions are calibrated based on test sets, both by hand and by histogramming (as used in [HZ95]). Currently, the normalization corpora (for which each *set* of training data constitutes a single point) is of insufficient volume to perform systematic learning from data [Ri88]. In future work, I plan to collect and synthesize representative corpora for every combination of learning architecture and available method, to promote the validity of the metrics for all plausible configurations of "prescribed learning technique".

2.3.2 Testing Metrics and Learning Components

My general directive for experiments with distributional metrics (especially those for heirarchical mixture models) was to simultaneously use the metric to select a learning component and to evaluate a candidate partition. By "simultaneous" I mean that the same metric was used in both contexts without substantial additional computations (not that the choice was committed concurrently).

2.3.3 Testing Partition Search

I generated numerous synthetic test sets to evaluate the partition enumerator, and to compare various informed search algorithms over the partition state space. These test sets had the common property that they were of significant difficulty (e.g., modular parity); decomposable

into well-balanced modules, *provided* the partitioning algorithm was complete and empirically sound; and demonstrated how a problem could be sufficiently decomposable for a *fair* allotment of computational resources. The fairness criterion means that the same number of trainable weights is allotted throughout a modular network (i.e., a hierarchical mixture model) – thus, all non-modular networks are being compared to mixture models whose specialists and moderators have a total network complexity that is comparable. (This actually skews the balance in favor of the non-modular inducers, because it disregards the possibility that parallel processing can be used in concurrent training of "siblings" in a hierarchical mixture model.)

In some test sets I introduced a nontrivial, but tolerable, quantity of "mixing" or "crosstalk" among modules. For the most part, the modular networks showed graceful degradation with *at least* the same quality as non-modular networks; however, this noise only made the partitioning algorithm more difficult, so I omitted it from further experimentation. An interesting line of future work, however, is to examine the robustness and incrementality of modular networks [Hr96, RH98].

References

[AD91] H. Almuallim and T. G. Dietterich. Learning with Many Irrelevant Features. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-91)*, p. 129-134, Anaheim, CA. MIT Press, Cambridge, MA, 1991.

[AHS85] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A Learning Algorithm for Boltzmann Machines. *Cognitive Science*, 9:147-169, 1985.

[AKA91] D. W. Aha, D. Kibler, and M. K. Albert. Instance-Based Learning Algorithms. *Machine Learning*, 6:37-66.

[Am95] S.-I. Amari. Learning and Statistical Inference. In *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, editor, p. 522-526.

[BD87] P. J. Brockwell and R. A. Davis. *Time Series: Theory and Methods.* Springer-Verlag, New York, NY, 1987.

[BDKL92] K. Basye, T. Dean, J. Kirman, and M. Lejter. A Decision-Theoretic Approach to Planning, Perception, and Control. *IEEE Expert* 7(4):58-65, 1992.

[Be90] D. P. Benjamin, editor. *Change of Representation and Inductive Bias*. Kluwer Academic Publishers, Boston, 1990.

[BF81] A. Barr and E. A. Feigenbaum. Search, In *The Handbook of Artificial Intelligence, Volume 1*, p. 19-139. Addison-Wesley, Reading, MA, 1981.

[BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA, 1984.

[BGH89] L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier Systems and Genetic Algorithms. *Artificial Intelligence*, 40:235-282, 1989.

[Bi95] C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, UK, 1995.

[BJR94] G. E. P. Box, G. M. Jenkins, and G.C. Reinsel. *Time Series Analysis, Forecasting, and Control (3rd edition).* Holden-Day, San Fransisco, CA, 1994.

[BM94] H. A. Bourlard and N. Morgan. *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer Academic Publishers, Boston, MA, 1994.

[BMB93] J. W. Beauchamp, R. C. Maher, and R. Brown. Detection of Musical Pitch from Recorded Solo Performances. *In Proceedings of the 94th Convention of the Audio Engineering Society*, Berlin, Germany, 1993.

[Bo90] K. P. Bogart. *Introductory Combinatorics, 2nd Edition.* Harcourt Brace Jovanovich, Orlando, FL, 1990.

[BR92] A. L. Blum and R. L. Rivest. Training a 3-Node Neural Network is NP-Complete. *Neural Networks*, 5:117-127, 1992.

[Br96] L. Breiman. Bagging Predictors. *Machine Learning*, 1996.

[BSCC89] I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper, The *ALARM* Monitoring System: A Case Study With Two Probabilistic Inference Techniques for Belief Networks. In *Proceedings of ECAIM '89, the European Conference on AI in Medicine*, pages 247-256, 1989.

[Bu98] D. Bullock. Personal communication, 1998.

[Ca93] C. Cardie. Using Decision Trees to Improve Case-Based Learning. In Proceedings of the 10th International Conference on Machine Learning, Amherst, MA, p. 25-32. Morgan-Kaufmann, Los Altos, CA, 1993.

[CF82] P. R. Cohen and E. A. Feigenbaum. Learning and Inductive Inference, In *The Handbook of Artificial Intelligence, Volume 3*, p. 323-511. Addison-Wesley, Reading, MA, 1982.

[CH92] G. F. Cooper and E. Herskovits. A Bayesian Method for the Induction of Probabilistic Networks from Data. *Machine Learning*, 9(4):309-347, 1992.

[Ch96] C. Chatfield. *The Analysis of Time Series: An Introduction (5th edition)*. Chapman and Hall, London, 1996.

[CKS+93] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, D. Freeman. AUTOCLASS: A Bayesian Classification System. *In Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 316-321, 1993.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[Co90] G. Cooper. The Computational Complexity of Probabilistic Inference using Bayesian Belief Networks. *Artificial Intelligence*, 42:393-405.

[CT91] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley and Sons, New York, NY, 1991.

[DH73] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, NY, 1973.

[DL95] T. Dean and S.-H. Lin. Decomposition Techniques for Planning in Stochastic Domains. In *Procedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1995.

[DLR77] A. Dempster, N. Laird, and D. Rubin. Maximum Likelihood From Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society*, 39(Series B):1-38.

[Do96] S. K. Donoho. *Knowledge-Guided Constructive Induction.* Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1996.

[DP92] J. Principé and deVries. The Gamma Model – A New Neural Net Model for Temporal Processing. *Neural Networks*, 5:565-576, 1992.

[DR95] S. K. Donoho and L. A. Rendell. Rerepresenting and Restructuring Domain Theories: A Constructive Induction Approach. *Journal of Artificial Intelligence Research*, 2:411-446, 1995.

[El90] J. L. Elman. Finding Structure in Time. *Cognitive Science*, 14:179-211, 1990.

[EVA98] R. Engels, F. Verdenius, and D. Aha. *Joint AAAI-ICML Workshop on Methodology of Machine Learning: Task Decomposition, Problem Definition, and Technique Selection*, 1998.

[FD89] N. S. Flann and T. G. Dietterich. A Study of Explanation-Based Methods for Inductive Learning. *Machine Learning*, 4:187-226, reprinted in *Readings in Machine Learning*, J. W. Shavlik and T. G. Dietterich, editors. Morgan-Kaufmann, San Mateo, CA, 1990.

[Fr98] B. Frey. Personal communication, 1998.

[FS96] T. Freund and R. E. Schapire. Experiments with a New Boosting Algorithm. In *Proceedings of ICML-96*.

[GBD92] S. Geman, E. Bienenstock, and R. Doursat. Neural Networks and the Bias/Variance Dilemna. *Neural Computation*, 4:1-58, 1992.

[GD88] D. M. Gaba and A. deAnda. A Comprehensive Anesthesia Simulation Environment: Re-creating the Operating Room for Research and Training. *Anesthesia*, 69:387:394, 1988.

[Gi96] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors. *Markov Chain Monte Carlo in Practice*. Chapman and Hall, New York, NY, 1996.

[Go89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[Gr92] R. Greiner. Probabilistic Hill-Climbing: Theory and Applications. In *Proceedings of the 9th Canadian Conference on Artificial Intelligence*, p. 60-67, J. Glasgow and R. Hadley, editors. Morgan-Kaufmann, San Mateo, CA, 1992.

[Gr98] E. Grois. *Qualitative and Quantitative Refinement of Partially Specified Belief Networks by Means of Statistical Data Fusion.* Master's thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1998.

[Gu91] G. H. Gunsch. *Opportunistic Constructive Induction: Using Fragments of Domain Knowledge to Guide Construction.* Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1991.

[GW94] N. A. Gershenfeld and A. S. Weigend. The Future of Time Series: Learning and Understanding. In *Time Series Prediction: Forecasting the Future and Understanding the Past (Santa Fe Institute Studies in the Sciences of Complexity XV)*, A. S. Weigend and N. A. Gershenfeld, editors. Addison-Wesley, Reading, MA, 1994.

[Ha89] D. Haussler. Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework. *Artificial Intelligence*, 36:177-221, 1989.

[Ha94] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan College Publishing, New York, NY, 1994.

[Ha95] M. H. Hassoun. *Fundamentals of Artificial Neural Networks*. MIT Press, Cambridge, MA, 1995.

[HB95] E. Horvitz and M. Barry. Display of Information for Time-Critical Decision Making. In *Proceedings of the Eleventh International Conference on Uncertainty in Artificial Intelligence (UAI-95)*. Morgan-Kaufmann, San Mateo, CA, 1995.

[He91] D. A. Heckerman. *Probabilistic Similarity Networks*. MIT Press, Cambridge, MA, 1991.

[He96] D. A. Heckerman. *A Tutorial on Learning With Bayesian Networks*. Microsoft Research Technical Report 95-06, Revised June 1996.

[HGL+98] W. H. Hsu, N. D. Gettings, V. E. Lease, Y. Pan, and D. C. Wilkins. A New Approach to Multistrategy Learning from Heterogeneous Time Series. In *Proceedings of the International Workshop on Multistrategy Learning*, 1998.

[Hi97] G. Hinton. Towards Neurally Plausible Bayesian Networks. Plenary Talk, *International Conference on Neural Networks (ICNN-97)*, Houston, TX, 1997.

[Hj94] J. S. U. Hjorth. *Computer Intensive Statistical Methods: Validation, Model Selection and Bootstrap.* Chapman and Hall, London, UK, 1994.

[HLB+96] B. Hayes-Roth, J. E. Larsson, L. Brownston, D. Gaba, and B. Flanagan. *Guardian Project Home Page*, URL: http://www-ksl.stanford.edu/projects/guardian/index.html

[HM91] G. Held and T. R. Marshall. *Data Compression: Techniques and Applications, 3rd edition.* John Wiley and Sons, New York, NY, 1991.

[Hr90] T. Hrycej. Gibbs Sampling in Bayesian Networks. *Artificial Intelligence*, 46:351-363, 1990.

[Hr92] T. Hrycej. *Modular Learning in Neural Networks: A Modularized Approach to Neural Network Classification*. John Wiley and Sons, New York, NY, 1992.

[HR76] L. Hyafil and R. L. Rivest. *Constructing Optimal Binary Decision Trees is NP-Complete.* Information Processing Letters, 5:15-17, 1996.

[HR98a] W. H. Hsu and S. R. Ray. A New Mixture Model for Concept Learning From Time Series. In *Proceedings of the 1998 Joint AAAI-ICML Workshop on Time Series Analysis*, to appear.

[HR98b] W. H. Hsu and S. R. Ray. Quantitative Model Selection for Heterogeneous Time Series. In *Proceedings of the 1998 Joint AAAI-ICML Workshop on Methodology of Machine Learning*, to appear.

[Hs95] W. H. Hsu. Hidden Markov Model Learning With Elman Recurrent Networks. *Final Project Report, CS442 (Artificial Neural Networks)*. University of Illinois at Urbana-Champaign, unpublished, December, 1995.

[Hs97] W. H. Hsu. A Position Paper on Statistical Inference Techniques Which Integrate Bayesian and Stochastic Neural Network Models. In *Proceedings of the International Conference on Neural Networks (ICNN-97)*, Houston, TX, June, 1997.

[Hu98] T. S. Huang. Personal communication, February, 1998.

[HZ95] W. H. Hsu and A. E. Zwarico. Automatic Synthesis of Compression Techniques for Heterogeneous Files. *Software: Practice and Experience*, 25(10): 1097-1116, 1995.

[Io96] T. Ioerger. *Change of Representation in Machine Learning, and an Application to Protein Tertiary Structure Prediction*. Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1996.

[JJ93] M. I. Jordan and R. A. Jacobs. Supervised Learning and Divide-and-Conquer: A Statistical Approach. In *Proceedings of the Tenth International Conference on Machine Learning*. Amherst, MA, 1993.

[JJ94] M. I. Jordan and R. A. Jacobs. Hierarchical Mixtures of Experts and the EM Algorithm. *Neural Computation*, 6:181-214, 1994.

[JJB91] R. A. Jacobs, M. I. Jordan, and A. G. Barto. Task Decomposition Through Competition in a Modular Connectionist Architecture: The What and Where Vision Tasks. *Cognitive Science*, 15:219-250, 1991.

[JJNH91] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive Mixtures of Local Experts. *Neural Computation*, 3:79-87, 1991.

[JK86] C. A. Jones and J.R. Kiniry. *CERES-Maize: a Simulation Model of Maize Growth and Development*. Texas A&M Press. College Station, TX, 1986.

[JKP94] G. John, R. Kohavi, and K. Pfleger. Irrelevant Features and the Subset Selection Problem. In *Proceedings of the 11th International Conference on Machine Learning*, p. 121-129, New Brunswick, NJ. Morgan-Kaufmann, Los Altos, CA, 1994.

[Jo87] M. I. Jordan. Attractor Dynamics and Parallelism in a Connectionist Sequential Machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, p. 531-546. Erlbaum, Hillsdale, NJ, 1987.

[Jo97a] M. I. Jordan. *Approximate Inference via Variational Techniques.* Invited talk, International Conference on Uncertainty in Artificial Intelligence (UAI-97), August, 1997. URL: http://www.ai.mit.edu/projects/jordan.html.

[Jo97b] M. I. Jordan. Personal communication, August, 1997.

[Ka95] C. M. Kadie. *SEER: Maximum Likelihood Regression for Learning Speed Curves*. Ph.D. thesis, University of Illinois, 1995.

[KGV83] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671-680, 1983.

[Ki86] J. Kittler. *Feature Selection and Extraction.* Academic Press, New York, NY, 1986.

[Ki92] K. Kira. *New Approaches to Feature Selection, Instance-Based Learning, and Constructive Induction.* Master's thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1992.

[KJ97] R. Kohavi and G. H. John. Wrappers for Feature Subset Selection. *Artificial Intelligence, Special Issue on Relevance*, *97*(1-2):273-324, 1997.

[Ko90] T. Kohonen. The Self-Organizing Map. *Proceedings of the IEEE*, 78:1464-1480, 1990.

[Ko94] I. Kononenko. Estimating Attributes: Analysis and Extensions of *Relief*. In *Proceedings of the European Conference on Machine Learning*, F. Bergadano and L. De Raedt, editors. 1994.

[Ko95] R. Kohavi. *Wrappers for Performance Enhancement and Oblivious Decision Graphs.* Ph.D. thesis, Department of Computer Science, Stanford University, 1995.

[KS96] R. Kohavi and D. Sommerfield. *MLC++: Machine Learning Library in C++, Utilities v2.0.* URL: http://www.sgi.com/Technology/mlc.

[KSD96] R. Kohavi, D. Sommerfield, and J. Dougherty. Data Mining Using *MLC++*: A Machine Learning Library in C++. In *Tools with Artificial Intelligence*, p. 234-245, IEEE Computer Society Press, Rockville, MD, 1996. URL: http://www.sgi.com/Technology/mlc.

[KR92] K. Kira and L. A. Rendell. The Feature Selection Problem: Traditional Methods and a New Algorithm. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-92)*, p. 129-134, San Jose, CA. MIT Press, Cambridge, MA, 1992.

[KV91] M. Kearns and U. Vazirani. *Introduction to Computational Learning Theory.* MIT Press, Cambridge, MA, 1991.

[Le89] K.-F. Lee. *Automatic Speech Recognition: The Development of the SPHINX System*. Kluwer Academic Publishers, Boston, MA, 1989.

[LFL93] T. Li, L. Fang, and K. Q-Q. Li. Hierarchical Classification and Vector Quantization With Neural Trees. *Neurocomputing* 5:119-139, 1993.

[Lo95] D. Lowe. Radial Basis Function Networks. In *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, editor, p. 779-782.

[LWYB90] L. Liu, D. C. Wilkins, X. Ying, and Z. Bian. Minimum Error Tree Decomposition. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence (UAI-90)*, 1990.

[LWH90] K. J. Lang, A. H. Waibel, and G. E. Hinton. A Time-Delay Neural Network Architecture for Isolated Word Recognition. *Neural Networks* 3:23-43, 1990.

[LY97] Y. Liu, and X. Yao. Evolving Modular Neural Networks Which Generalise Well. In *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC-97)*, p. 605-610, Indianapolis, IA, 1997.

[Ma89] C. J. Matheus. *Feature Construction: An Analytical Framework and Application to Decision Trees*. Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1989.

[Mi83] R. S. Michalski. A Theory and Methodology of Inductive Learning. *Artificial Intelligence*, 20(2):111-161, reprinted in *Readings in Knowledge Acquisition and Learning*, B. G. Buchanan and D. C. Wilkins, editors. Morgan-Kaufmann, San Mateo, CA, 1993.

[Mi80] T. M. Mitchell. *The Need for Biases in Learning Generalizations.* Technical Report CBM-TR-117, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1980, reprinted in *Readings in Machine Learning*, J. W. Shavlik and T. G. Dietterich, editors. Morgan-Kaufmann, San Mateo, CA, 1990.

[Mi82] T. M. Mitchell. Generalization as Search. *Artificial Intelligence*, 18(2):203-226.

[Mi93] R. S. Michalski. Toward a Unified Theory of Learning: Multistrategy Task-Adaptive Learning. In *Readings in Knowledge Acquisition and Learning*, B. G. Buchanan and D. C. Wilkins, eds. Morgan-Kaufmann, San Mateo, CA, 1993.

[Mi97] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, NY, 1997.

[MMR97] K. Mehrotra, C. K. Mohan, and S. Ranka. *Elements of Artificial Neural Networks*. MIT Press, Cambridge, MA, 1997.

[MN83] P. McCullagh and J. A. Nelder. *Generalized Linear Models*. Chapman and Hall, London, 1983.

[Mo94] M. C. Mozer. Neural Net Architectures for Temporal Sequence Processing. In *Time Series Prediction: Forecasting the Future and Understanding the Past (Santa Fe Institute Studies in the Sciences of Complexity XV),* A. S. Weigend and N. A. Gershenfeld, editors. Addison-Wesley, Reading, MA, 1994.

[MP69] M. L. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry, first edition.* MIT Press, Cambridge, MA, 1969.

[MR86] J. L. McClelland and D. E. Rumelhart. *Parallel Distributed Processing*. MIT Press, Cambridge, MA, 1986.

[My95] P. Myllymäki. Mapping Bayesian Networks to Boltzmann Machines. In *Proceedings of Applied Decision Technologies 1995*, pages 269-280, 1995.

[Ne92] R. M. Neal. *Bayesian Training of Backpropagation Networks by the Hybrid Monte Carlo Method.* Technical Report CRG-TR-92-1, Department of Computer Science, University of Toronto, 1992.

[Ne93] R. M. Neal. *Probabilistic Inference Using Markov Chain Monte Carlo Methods.* Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto, 1993.

[Ne96] R. M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, New York, NY, 1996.

[Pe88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan-Kaufmann, San Mateo, CA, 1988.

[Pe95] J. Pearl. Bayesian Networks. In *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, editor, p. 149-153.

[Pe97] E. Pérez. *Learning Despite Complex Attribute Interaction: An Approach Based on Relational Operators*. Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1997.

[PL98] J. Principé, C. Lefebvre. *NeuroSolutions v3.02*, NeuroDimension, Gainesville, FL, 1998. URL: http://www.nd.com.

[Qu85] R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1:81-106, 1985.

[Ra90] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, reprinted in *Readings in Speech Recognition*, A. Waibel and K.-F. Lee, editors. Morgan Kaufmann, San Mateo, CA, 1990.

[RCK89] J. G. Rueckl, K. R. Cave, and S. M. Kosslyn. Why are "What" and "Where" Processed by Separate Cortical Visual Systems? A Computational Investigation. *Journal of Cognitive Neuroscience*, 1:171-186.

[RH98] S. R. Ray and W. H. Hsu. Self-Organized-Expert Modular Network for Classification of Spatiotemporal Sequences. *Journal of Intelligent Data Analysis*, to appear.

[Ri88] J. A. Rice. *Mathematical Statistics and Data Analysis*. Wadsworth and Brooks/Cole Advanced Books and Software, Pacific Grove, CA, 1988.

[RK96] S. R. Ray and H. Kargupta. A Temporal Sequence Processor Based on the Biological Reaction-Diffusion Process, *Complex Systems*, 9(4):305-327, 1996.

[RNH+98] C. E. Rasmussen, R. M. Neal, and G. Hinton. *Data for Evaluating Learning in Valid Experiments (DELVE)*. Department of Computer Science, University of Toronto, 1996. URL: http://www.cs.toronto.edu/~delve/delve.html.

[RN95] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[Ro98] D. Roth. Personal communication, 1998.

[RR93] H. Ragavan and L. A. Rendell. Lookahead Feature Construction for Learning Hard Concepts. In *Proceedings of the 1993 International Conference on Machine Learning (ICML-93)*, June, 1993.

[RS88] H. Ritter and K. Schulten. Kohonen's Self-Organizing Maps: Exploring Their Computational Capabilities. In *Proceedings of the International Conference on Neural Networks (ICNN-88)*, p. 109-116, San Diego, CA, 1988.

[RS90] L. A. Rendell and R. Seshu. Learning Hard Concepts Through Constructive Induction: Framework and Rationale. *Computational Intelligence*, 6:247-270, 1990.

[RV97] P. Resnick and H. R. Varian. Recommender Systems. *Communications of the ACM*, 40(3):56-58, 1997.

[Sa89] G. Salton. *Automatic Text Processing*. Addison Wesley, Reading, MA, 1989.

[Sa97] M. Sahami. Applications of Machine Learning to Information Access (AAAI Doctoral Consortium Abstract). In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, p. 816, Providence, RI, 1997.

[Sa98] W. S. Sarle, editor. *Neural Network FAQ*, periodic posting to the Usenet newsgroup *comp.ai.neural-nets*, URL: ftp://ftp.sas.com/pub/neural/FAQ.html

[Sc97] D. Schuurmans. A New Metric-Based Approach to Model Selection. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, p. 552-558.

[Se98] C. Seguin. *Models of Neurons in the Superior Colliculus and Unsupervised Learning of Parameters from Time Series*. Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1998.

[Sh95] Y. Shahar. *A Framework for Knowledge-Based Temporal Abstraction.* Stanford University, Knowledge Systems Laboratory Technical Report 95-29, 1995. URL: http://www-smi.stanford.edu/pubs/SMI_Abstracts/SMI-95-0567.html

[SM86] R. E. Stepp III and R. S. Michalski. Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects. In *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors. Morgan-Kaufmann, San Mateo, CA, 1986.

[SM93] B. Stein and M. A. Meredith. *The Merging of the Senses*. MIT Press, Cambridge, MA, 1993.

[St77] M. Stone. An Asymptotic Equivalence of Choice of Models by Cross-Validation and Akaike's Criterion. *Journal of the Royal Statistical Society Series B*, 39:44-47.

[Th96] S. Thrun. *Explanation-Based Neural Network Learning*. Kluwer Academic Publishers, Norwell, MA, 1996.

[TK94] H. M. Taylor and S. Karlin. An Introduction to Stochastic Modeling. Academic Press, San Diego, CA, 1984.

[TSN90] G. G. Towell, J. W. Shavlik, M. O. Noordewier. Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks. *In Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-90)*, pages 861-866, 1990.

[Vi67] A. J. Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory*, 13(2):260-269, 1967.

[Vi98] R. Vilalta. *On the Development of Inductive Learning Algorithms: Generating Flexible and Adaptable Concept Representations.* Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1998.

[Wa72] S. Watanabe. Pattern Recognition as Information Compression. In *Frontiers of Pattern Recognition*, S. Watanabe, editor. Academic Press, San Diego, CA, 1972.

[Wa85] S. Watanabe. *Pattern Recognition: Human and Mechanical*, John Wiley and Sons, New York, NY, 1985.

[Wa98] B. Wah. Personal communication, January, 1998.

[Wi93] P. H. Winston. *Artificial Intelligence, 3rd Edition.* Addison-Wesley, Reading, MA, 1993.

[WM94] J. Wnek and R. S. Michalski. Hypothesis-Driven Constructive Induction in AQ17-HCI: A Method and Experiments. *Machine Learning*, 14(2):139-168, 1994.

[Wo92] D. H. Wolpert. Stacked Generalization. *Neural Networks*, 5:241-259, 1992.

[WCB86] D. C. Wilkins, W. J. Clancey, and B. G. Buchanan, *An Overview of the Odysseus Learning Apprentice.* Kluwer Academic Press, New York, NY, 1986.

[WS97] D. C. Wilkins and J. A. Sniezek. *DC-ARM: Automation for Reduced Manning*. Knowledge Based Systems Laboratory Technical Report UIUC-BI-KBS-97-012. Beckman Institute, UIUC, 1997.

[WZ89] R. J. Williams and D. Zipser. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation* 1(2):270-280.

[ZMW93] X. Zhang, J. P. Mesirov, and D. L. Waltz. A Hybrid System for Protein Secondary Structure Prediction. Preprint, Journal of Molecular Biology, 1993.

Curriculum Vitae

William Henry Hsu was born on October 1, 1973 in Atlanta, Georgia. He graduated in June, 1989 from Severn School in Severna Park, Maryland, where he was a National Merit Scholar. In May, 1993, he was awarded the Outstanding Senior Award from the Department of Computer Science at the Johns Hopkins University in Baltimore, Maryland, and received dual bachelor of science degrees in Computer Science and Mathematical Sciences, with honors. He also received a concurrent Master of Science in Engineering from the Johns Hopkins University in May, 1993. After entering the graduate program in Computer Science at the University of Illinois at Urbana-Champaign, he joined the research group of Professor Sylvian R. Ray in 1996. He was awarded the Ph.D. degree in 1998 for his work on time series learning with probabilistic networks, an approach integrating constructive induction, model selection, and hierarchical mixture models for learning from heterogeneous time series. He has presented research papers at various scientific conferences and workshops on artificial intelligence, intelligent systems for molecular biology, and artificial neural networks. His research interests include machine learning and data mining, time series analysis, probabilistic reasoning for decision support and control automation, neural computation, and intelligent computer-assisted instruction.