2009

Concept Learning By Example Decomposition

Sameer Joshi
University of Central Florida

CONCEPT LEARNING BY EXAMPLE DECOMPOSITION

by

SAMEER JOSHI
M.S., University of Central Florida, 2008

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2008

Major Professor: Charles E. Hughes

ABSTRACT

For efficient understanding and prediction in natural systems, even in artificially closed ones, we usually need to consider a number of factors that may combine in simple or complex ways. Additionally, many modern scientific disciplines face increasingly large datasets from which to extract knowledge (for example, genomics). Thus to learn all but the most trivial regularities in the natural world, we rely on different ways of simplifying the learning problem.

One simplifying technique that is highly pervasive in nature is to break down a large learning problem into smaller ones; to learn the smaller, more manageable problems; and then to recombine them to obtain the larger picture. It is widely accepted in machine learning that it is easier to learn several smaller decomposed concepts than a single large one. Though many machine learning methods exploit it, the process of decomposition of a learning problem has not been studied adequately from a theoretical perspective. Typically such decomposition of concepts is achieved in highly constrained environments, or aided by human experts.

In this work, we investigate concept learning by example decomposition in a general probably approximately correct (PAC) setting for Boolean learning. We develop sample complexity bounds for the different steps involved in the process. We formally show that if the cost of example partitioning is kept low then it is highly advantageous to learn by example decomposition. To demonstrate the efficacy of this framework, we interpret

the theory in the context of feature extraction. We discover that many vague concepts in feature extraction, starting with what exactly a feature is, can be formalized unambiguously by this new theory of feature extraction. We analyze some existing feature learning algorithms in light of this theory, and finally demonstrate its constructive nature by generating a new learning algorithm from theoretical results.

This work is first and foremost dedicated to my family, which has stood by me all these years and is center of my existence. It is also dedicated to my friends, especially Ravi, whose arguments convinced me more often than I'd ever admit to him.

ACKNOWLEDGMENTS

I would like to acknowledge all my teachers who helped me understand the world better. I would particularly like to acknowledge the help and guidance provided by Dr. Hughes, the EECS patron saint of waifs.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS / ABBREVIATIONS

PAC	Probable Approximately Correct

CHAPTER 1: INTRODUCTION

Learning is a central part of the cognitive process. We acquire all our knowledge from learning. Since evolution is a learning algorithm, even instincts such as eating, or the fight or flight response, are learned and then imprinted onto our genetic code. It is no surprise then that the process of learning has occupied a central role in computer and cognitive science research. In recent years the study of learning has increasingly fallen under the rubric of computer science rather than mathematics or statistics alone. New research fields such as computational learning theory have expanded the paradigm from interpolation or induction by introducing novel complexity measures, new paradigms to look at learning, and tying abstract theory to concrete algorithms. This has facilitated a more applied approach.

This chapter starts by considering what learning is, and how it has been considered in computer science. We look at some factors that make learning a hard problem in the natural world, and introduce our approach to its simplification. In this thesis we present a formal analysis of a major means of simplification of learning, the decomposition of the learning problem.

What is Learning

Learning involves acquiring knowledge from the external world through experience. Broadly speaking, learning may be defined as predicting future events based on past observations. The learner observes the external world until a pattern emerges; this pattern is then internalized in some form, like a rule, an equation, or a philosophy. Furthermore,

the learner may continue to refine the learned idea by continuing to observe. If possible, the learner may test the learned idea in the external world. Modifying the idea leads to modification of behavior, and this correspondence may be used to refine the idea. If we stratify this idea into temporal steps, we have

1. Acquire experience
2. Use this experience to form an idea
3. Modify behavior based on this new knowledge, influencing future experience
4. Repeat the steps

This process may be formalized mathematically. The learner becomes an algorithm. The experience from the external world may be quantified as well. The idea to be learned becomes a hypothesis the learning algorithm postulates. The steps may be rewritten as

1. Acquire samples from a dataset
2. Induce hypothesis based on these samples
3. Test hypothesis by predicting values of samples from dataset
4. Continue until desired accuracy has been achieved.

Of course, the learned hypothesis may not predict the values from the dataset perfectly. Even if the algorithm is working perfectly thus far, there is no guarantee that there would not arise a sample from the dataset that would throw the hypothesis off, necessitating its modification. Thus the algorithm learns only approximately. Also, there is no guarantee that the algorithm would ever learn the required hypothesis, though it is reasonable to think that an algorithm of sufficient power would learn eventually, given enough samples. These ideas are formalized in the probably approximately correct (PAC)

framework [Val84]. Other fields in computer science and statistics such as computational learning theory and statistical learning theory are closely related, and in many ways equivalent for the purposes of this thesis. In the next section, we discuss PAC learning in greater detail.

The PAC Framework

To address the questions posed in this thesis, we introduce a formal model for concept learning by example decomposition in this thesis. This model builds upon the PAC framework and its variants [Val84, Vap82, Hau92].

Learning involves looking at some data and forming a general model, or hypothesis, for the purposes of classification or regression. The learning algorithm is provided with a data set, called training data, through which the algorithm forms a hypothesis. The hypothesis is used to predict or classify (and is hence tested by) another data set called testing data.

Consider a set X called the instance space. X provides the input data for a learning algorithm. For example, X may be the set of all English words. A concept to be learned would be a subset of X that exemplifies some property. For example, the set of palindromes is a concept over the instance space of all English words. For learning, a sample of m examples is drawn from $X \times Y$ according to a probability distribution D, where $Y = \{0, 1\}$. A concept c is a function

$c: X \rightarrow \{0,1\}$.

A concept class C is a collection of concepts over X. A target concept $c_t \in C$ correctly classifies all learning examples drawn from $X \times Y$. The task of a learner L is to model the target concept as closely as possible [Ale99] by forming a hypothesis h $\in H$, where H is the space of all possible hypotheses. The hypothesis space may be understood as all possible concepts that L may determine.

In the PAC framework, the task of the learner is to find a good approximation for the target concept $c_t : X \to Y$ drawn from the concept space C. Training data $z = \{(x_1, y_1), (x_2, y_2), \ldots (x_m, y_m)\}$ consisting of examples for c_t are drawn from $X \times Y$ according to some probability distribution D and presented to the learner. Based on this data, the learner picks a hypothesis $h : X \to Y$ from the hypothesis space H so as to minimize some measure of expected error with respect to D.

All learning algorithms are blind except for a factor called bias, which is the set of all factors that collectively influence hypothesis selection [Utg86]. Inductive bias may be understood as assumptions about the target hypothesis that aid in its selection. An example is parsimony, where the smallest hypothesis among a selection is favored.

We formally define learning in the PAC framework as introduced in [Hau90]. Let k be the representation size of a sample. For each $k \geq 1$ let C^k be a set of concepts over the instance space X. A concept class is $C := \{C^k\}_{k \geq 1}$. Similarly H^k for $k \geq 1$ is a set of hypotheses, and $H := \{H^k\}_{k \geq 1}$ is the hypothesis space.

Definition 1 (PAC Learnable). *Let C and H be defined over X. If concept class C is* ***PAC learnable*** *within confidence parameter δ and accuracy parameter ε by hypothesis space H, then there exists an algorithm L and a polynomial $p(\cdot,\cdot,\cdot)$ with the property that when L is presented with a set M samples of any $c_t \in C^k$ picked using an arbitrary distribution D, L returns a hypothesis $L(M) \in H^k$ and*

$$0 < \delta < 1, 0 < \varepsilon < 1, k \geq 1 [\text{"whenever"} m \geq p(k, \frac{1}{\varepsilon}, \frac{1}{\delta}), D^m \{er_D(L(M)) < \varepsilon\} > 1 - \delta]$$

Where $m = |M|$, and $er_D(L(M))$ is the error of machine L on M.

PAC learnability measures a property of the problem being learned, in this case the concept class *C*. It simply means that if *m* is large enough, then there would exist an algorithm which would be able to learn *C* within some parameters. Here, *L* is a learning algorithm that is presented a sample set *M* of size *m* picked from the concept space *C*. Upon processing this sample set, *L* returns a hypothesis *L(M)* explaining it. Thus *L* has 'learned' something about the concept which was sampled, and represented this knowledge in *L(M)*. This hypothesis may not have perfectly learned the concept under question, and there is a chance that *L* may not have learned anything at all. These factors are represented by the error and confidence parameters. Also, the difficulty of learning is proportional to the size of each individual sample in the sample set provided for learning, given by *k*. *C* is PAC learnable if some *L* can learn it with *m* samples, where *m* is bound by a polynomial function over *k*, the error, and the confidence parameters.

VC Dimension

We use some concepts from statistical learning theory [Foe94] in this thesis. An important concept, widely used in machine learning, is that of the Vapnik-Chervonenkis dimension (VC dimension) [Vap98]. The VC dimension is a statistical measure of the capacity of a classification algorithm. Roughly speaking, it measures the 'power' of a classification algorithm. The task of learning a concept *c,* given a sat of samples, may be viewed as distinguishing which samples belong to the concept and which do not. Thus all learning problems in the PAC framework can posed as classification problems and all learning algorithms are also classification algorithms, making the VC dimension a useful measure of the power of any PAC algorithm.

To understand the VC dimension we have to first consider the concept of 'shattering' [Vap98]. A classification model with a parameter vector V (i.e. the parameters used to obtain a specific configuration of the model) is said to shatter a set of data points $\{x_1, x_2, \ldots x_n\}$, if for any placement of these points, there exists a V that correctly classifies all of them.

The VC dimension of a learning algorithm L is the cardinality of the largest set of points L can shatter. The VC dimension of L is related to how complicated L can be.

Kolmogorov Complexity

The Kolmogorov complexity [LV97] of an object is a measure of the computational resources it takes to specify that object. Usually, the object under consideration is a string, and the Kolmogorov complexity of a string may be understood as the size of the

smallest program that would generate that string. This program is written in some fixed universal programming language.

There is no way to compute the Kolmogorov complexity of a string. Therefore it is a purely theoretical measure. However, generalized versions, or approximations to Kolmogorov complexity exist. An example of this is Kolmogorov-Levin complexity (sometimes called Levin complexity). It is a resource bounded generalization of Kolmogorov complexity. It penalizes a slow program by adding the logarithm of its running time to the program's length. This leads to a computable, though sometimes intractable in practice, version of Kolmogorov complexity.

Learning in a Complex World

The natural world is a complex place with many variables, factors, and dependencies. The complexity of learning increases exponentially as we add to it. For example, consider the difference in difficulty of learning a Boolean function *f*(*A,B*) -> *C* versus another *g*(*A, B, C, D, E, F*) -> *G* based on their truth tables. The former requires just 4 examples, while the latter requires 16 times more examples to sift through. The number of functions possible on an input of *n* variables is *2^n^n*. So the number of hypotheses a machine has to sort through is even greater.

There are many factors that make learning hard, not the least of which is that the difficulty of the learning task increases exponentially with the size of the task. Given this growth in complexity, how is it possible to learn efficiently in this system?

Simplification of Learning

Most learning methods rely on constraining the learning task though some means to simplify learning. Such heuristics are usually domain specific in nature, and are often specified by human experts. In this work, we study a general way of simplifying learning by breaking it into smaller tasks. Learning by concept decomposition is a pervasive process. In fact, we may even see it present in general scientific methodology. For example, rather than discover the entire body of physics in one go, we isolate and study subsystems of the natural universe, and then combine the knowledge gained with preexisting knowledge.

Learning by decomposition of concepts is a pervasive process in computer science. Of the various kinds of decompositions possible, we consider *example* decomposition, where each example in the training data z is split into sub-examples $x_i = (x_{i1}, x_{i2}, \ldots x_{in})$, and each sub-example with its own label is used to learn a subconcept. Once all the subconcepts have been learned, they can be reassembled to yield the target concept c_t.

For example, in face recognition, each example may consist of an image of a face. This image may be divided into smaller images, with each smaller image consisting of a feature, like nose, eyes, etc.

In this work, we present our research on probably approximately correct (PAC) learning of Boolean concepts by decomposing the examples presented to the learner. While learning by decomposition is used often, the subject has not received broad theoretical

treatment. Many questions remain open. The first obvious question is, what exactly is decomposition? What all may we decompose? How can we learn to partition the example into sub-examples? Then how do we learn the subconcepts? How can the learned subconcepts be reassembled? What are the advantages? And perhaps the most important question is, when will the savings obtained be greater than the overhead costs? A systematic study of the process of PAC learning by example decomposition yields insights into the answers to these questions. We provide a framework for example decomposition, and provide upper limits on its sample complexity. We also develop conditions under which learning by decomposition is advantageous.

CHAPTER 2: RELATED WORK

As we mentioned before, learning by concept decomposition is a pervasive phenomenon in nature and thus shows up in many sciences, and various forms of reasoning. So naturally, there exists a large body of algorithmic and experimental work that deals with problem decomposition directly or indirectly. Some of these approaches may be seen as special cases of the model presented in this thesis. Others are not directly relevant. Some of this work is not in machine learning, but operates on classes of problems that are decomposable, and thus is included here. Without attempting to be exhaustive, we overview some of the major contributions to developing the theory of decomposition.

Pattern Theory

The premise behind pattern theory [Gre07, Mum96] is that the universe can be expressed in a language of patterns. It postulates that compositional representations of the universe can be formed, and that these representations are commonly found in nature [Ale99]. That is, we can combine simple primitives according to some rules to form increasingly complex primitives and systems. Pattern theory inherently lends itself to a decompositional nature, because learning a compositional representation may be done best through isolating the components and the rules that combine them. This paradigm of learning in pattern theory has not received direct treatment in the PAC framework, though there are many examples of learning algorithms developed under its rubric. Feature extraction using pattern theory is discussed in [RNGG94] where the authors introduce a complexity measure called Decomposed Function Cardinality, and a

decomposition algorithm to minimize this measure. Another example of a pattern theoretic application to knowledge discovery using pattern theory may be found in [Gol95]. In this paper, the authors recursively decompose a function to its atomic elements. Pattern Theory allows extrapolation on available information based on the inherent structure in the data; it ports over to Knowledge Discovery in Databases naturally.

Feature Extraction

Feature extraction [HS03, GE06] is the process of generating a set of characteristic attributes from a given dataset. As such, feature extraction is very close to learning by decomposition, because each feature may be considered a decomposed subpart of the problem being learned. As it stands today, feature extraction is primarily an empirical science, with little theoretical background. Most of the theoretical work pertains to individual algorithms [HKCWL03, ZKF02, MM05] or low-level feature extraction [Now77, Foe94]. Baxter [Bax00] shows an example of how feature extraction may be considered in the PAC framework. Though primarily thought of as an image-processing field, feature extraction is a commonly occurring process across various fields. Most science involves the extraction of abstract features by looking at raw data, and then finding interconnections among those features. We discuss feature extraction in greater detail as a case study in chapter 4.

Clustering

Clustering [Rom04] is the partitioning by classification of a data set into different subsets, so that the data in each subset share some common trait. This trait is expressed as

proximity according to some defined distance measure. Thus clustering may be seen as decomposition of data into smaller spaces. The clusters may be seen as decomposed elements of a larger dataset. Conceptual clustering [MS84, SM86] takes another step towards machine learning by concept description for each generated class. Here, a descriptive concept is generated for each cluster. Usually, conceptual clustering algorithms also form hierarchical structures relating the concepts. A variety of methods have been developed where the description may rely on logic (e.g., [Fis87]), or probabilistic mechanisms (e.g., [TB01]).

Meta Learning

Since meta learning [BK90, Mau05] involves learning many smaller concepts while gaining global bias, it touches upon learning by decomposition. Baxter [Bax00] shows that a learner embedded in an environment of related tasks can automatically acquire bias. This is directly relevant to learning by example decomposition. The details are elaborated upon later in this thesis in chapter 5.

Dynamic Programming

Dynamic programming [CLR90, Rom04] was introduced in its modern form by Bellman [Bel57] in the 1950s. It is an example of how decomposition is formalized and used in computer science. Dynamic programming is a problem solving methodology that solves a large problem by finding optimal solutions to its subproblems. Dynamic programming works on problems exhibiting the properties of optimal substructure and overlapping subproblems. Optimal substructure means that an optimal solution to a subproblem would form part of an optimal solution to the global problem. Overlapping subproblems means

that the same subproblems may be used to solve different larger problems, i.e., they are reused. Since dynamic programming involves finding subproblems to a larger problem, it works by decomposition of the task. However, dynamic programming is not a learning algorithm. It also works in a more restricted domain than that presented in this paper. Interestingly, overlapping substructure in a problem is a factor that contributes to the savings obtained in learning by decomposition. We discuss this idea further in chapter 5 under the heading 'reuse of subconcepts'.

Domain Decomposition

Domain decomposition [CM94, Qua92] is a method that solves a boundary value problem by splitting it into smaller boundary value problems. A boundary value problem for an ordinary differential equation or a partial differential equation consists of the equation and its boundary conditions. Since any physical differential equation would have a boundary value problem, they occur prolifically in physics. If we can decompose the domain into sub-domains, large savings in the size of the problem are obtained. In [Chan87] some preconditions for domain decomposition are discussed. Domain decomposition is interesting to mention here, not only because it relies on splitting a larger problem into smaller ones, but also because of its heavy correlation with the natural sciences. It hints that decomposability is an inherent characteristic in natural representations, thus supporting the case for learning by decomposition.

Divide-and-Conquer Learning

Dietterich [Die00] introduced the term divide-and-conquer learning and outlined some research questions in the field. This methodology seeks to decompose large input sets

into smaller more manageable ones. Either the input set, or individual examples may be divided. Divide-and-conquer is different from decomposition. In the former, division is a design decision, and can be varied by the learner. In the latter, however, decomposability is a property of the concept being decomposed and the lines along which to decompose must be learned.

Separate and Conquer Learning

Separate and conquer learning [Fur99] was introduced in [Mic69] as the covering strategy. This strategy involves recursively searching for rules to explain subsets of the training instances until each example is covered by at least one rule. It is possible for separate and conquer to involve example decomposition, but it corresponds more closely to a specific methodology for domain decomposition

CHAPTER 3: FRAMEWORK FOR CONCEPT DECOMPOSITION

The complexity of learning increases with the representational size of a concept. The combinatorial nature of adding variables to a system makes learning infeasible fairly quickly in anything but the most trivial systems. In nature, complex learning is usually achieved by learning sub-parts of the problem separately, and then combining them together. So it is of much interest to formally study the process of learning by decomposition.

We first formally define PAC decomposability and study the case when it's advantageous. We then discuss the process of learning by decomposition and provide sample complexity bounds for each of the steps in the process. We combine these bounds to obtain the conditions for propitious concept decomposition.

Posing the Problem Formally

To decompose a learning problem, we have to split the learning task into smaller tasks, learn them separately, and then put them back together to form a coherent solution to the original problem. Thus, a given target concept c_t is decomposable if there exists an equivalent representation

$$c_t = f_a(c_1, c_2 \ldots, c_n).$$

It is desirable that decomposition does not introduce error or reduce the likelihood of learning the task by unacceptable amounts.

Definition 2 (PAC decomposable). *A target concept* $c_t \in C^k$, *where* $k \geq 1$, *is* ***PAC decomposable*** *within confidence parameter* δ *and accuracy parameter* ε *if*

• There exists a PAC learning algorithm L that splits c_t into $c_1, c_2 \ldots, c_n$ such that $c_t = f_a(c_1, c_2 \ldots, c_n)$ within accuracy and confidence parameters ε_L and δ_L respectively.

• $f_a, c_1, c_2 \ldots, c_n$ are PAC learnable within accuracy and confidence parameters ε_f, δ_f and $\varepsilon_1, \delta_1, \varepsilon_2, \delta_2, \ldots, \varepsilon_n, \delta_n$.

• $(1-\varepsilon_L)(1-\varepsilon_f)\prod_{i=1}^{n}(1-\varepsilon_i) \leq 1-\varepsilon$

• $(1-\delta_L)(1-\delta_f)\prod_{i=1}^{n}(1-\delta_i) \leq 1-\delta$

We may introduce the additional constraint that such decomposition should benefit the learning process by reducing its complexity. Though any complexity measure may be studied, we consider the benefits of concept decomposition on sample complexity, which is the implicit complexity measure for the rest of the thesis. For a subconcept to be discoverable within given confidence and error parameters there must be a sufficient amount of information about it present in the input examples. Not only that, this information must be less than the information required to learn c_t, or there is no reduction in sample complexity gained by decomposition. So for a subconcept $c_i \in c_t$ with given accuracy parameter ε_i and confidence parameter δ_i, the sample complexity must be less than the sample complexity for learning c_t. In fact, the combined sample complexity of the subconcepts $c_1, c_2 \ldots, c_n$, the cost of decomposing the examples, and f_a must be less

than the sample complexity for c_t. Let S_c be the smallest set of examples that allows us to learn a concept c, and the cardinality of a set S be given by $|S|$. Note that this lower bound $|S_c|$ is the sample complexity of c, i.e. the cost of learning c given some accuracy and confidence parameters, and size of samples. Then a concept is *propitiously* decomposable if

$$c_t = f_a(c_1, c_2 \ldots, c_n), \text{"such that"} | S_{c_t} | \leq | \bigcup_{i=1}^{n} S_{c_i} \bigcup S_f \bigcup S_L |,$$

where $| S_L |$ is the cost of decomposing the examples. We union S_L, S_f, and S_c for the subconcepts before we take their cardinality because some samples may serve to learn multiple concepts. These samples have to be considered only once as they do not add to the sample complexity a second time. The cardinality of the union of all these sets gives us the cost of decomposing the examples, learning the subconcepts, and then putting them back together again, with no sample counted twice.

If we add this condition to PAC decomposability, we have the definition for propitious PAC decomposability.

Definition 3 (Propitiously PAC decomposable). *A target concept* $c_t \in C_k$, *where* $k \geq 1$, *is* ***propitiously PAC decomposable*** *if*

• There exists a PAC learning algorithm L that splits c_t into $c_1, c_2 \ldots, c_n$ such that

$c_t = f_a(c_1, c_2 \ldots, c_n)$

• $f, c_1, c_2 \ldots, c_n$ are PAC learnable

• $(1 - \varepsilon_L)(1 - \varepsilon_f) \prod_{i=1}^{n} (1 - \varepsilon_i) \leq 1 - \varepsilon_{c_t}$

• $(1-\delta_L)(1-\delta_f)\prod_{i=1}^{n}(1-\delta_i) \leq 1-\delta_{c_t}$

• $|S_{c_t}| \leq |\bigcup_{i=1}^{n} S_{c_i} \bigcup S_f \bigcup S_L|$

So a concept is PAC decomposable if it may be learned as subconcepts and then put together while staying within the confidence and error bounds. It is propitiously so if all this can be done so that the sample complexity of the learning problem is reduced. This, of course, is the main motivation. There are some important points to note about the definitions introduced above.

• The task of learning c_t is replaced by many smaller learning tasks. These are the i subconcepts, f, and the concept learned by L.

• Computing the sample complexities of the subconcepts is not straightforward, as the same example may serve towards learning multiple subconcepts. Therefore we take the union of all the examples required for the subconcepts when computing sample complexity for the decomposed concept.

• L is the learning algorithm that decomposes the examples for learning the subconcepts. This may range from a trivial to a highly complex task. There would exist some natural boundaries along which an example might be decomposed. So problem specific bias would play an important role here. The question of what subconcepts are useful to learn,

and which are chimerical, is also intimately related to how the examples are decomposed. We consider this problem in greater detail in the discussion section.

• The function f_a provides the process of recombining the subconcepts into the original target concept, and must be learned. In many cases, f_a may be seen as reverse engineering the concept learned by L.

• In our definition, we assume errors to be multiplicative. But to be more precise they depend on exactly how the subconcepts are amalgamated, or f_a.

• $| S_c |= \min p_c(\cdot,\cdot,\cdot)$.

We consider learning Boolean concepts in this thesis. That is, for a given set of binary inputs, the output of the concept is either positive (1) or negative (0).

Analyzing the Process of Concept Decomposition

In this section, we discuss the sample complexity of learning subconcepts. Besides the usual PAC requirements, a subconcept must be isolated from the target concept, which poses its own restrictions.

Labeled examples are required for PAC learning. To discover a subconcept c_i, we have to isolate the examples that can help learn it, i.e. the examples that provide a positive or

negative example for c_i. Can the examples drawn from X for a target concept c_t be used to learn c_i? We have to determine how examples drawn from $X \times Y$ are relevant for learning $c_1, c_2 \ldots, c_n$.

Sample complexity for learning by concept decomposition may be analyzed under two different settings. The first is when it is possible to obtain labels for the subconcept-examples. This may correspond to a real world situation where the learner has some bias concerning the subconcepts. In this case, we assume that there are oracles present for the subconcepts to generate labels for examples. The other case is the stricter condition that there are no oracles present for learning the subconcepts; the only oracle generates labels for the target concept c_t. This corresponds to the real world situation where we are given a concept to learn, and without any a priori knowledge we must determine if the concept is decomposable, and then discover the decomposition and learn it without any extra help. To do so, we have to take the jump to unsupervised learning. In this thesis, we only consider the former case.

Here we analyze the sample complexity of learning by concept decomposition with oracles available for the decomposed subconcepts. Learning a concept by decomposition consists of the following steps

1. Divide the hypothesis space for the target concept into spaces for the subconcepts.
2. Learn each subconcept using its space.
3. Combine the learned subconcepts back together to form the target concept.

We discuss these steps in greater detail in the following sections.

Dividing the Hypothesis space

To learn a concept by decomposing it, we would first have an algorithm learn the divisions of the hypothesis space corresponding to the subconcepts. We discuss two ways of splitting the hypothesis space for the subconcepts. The first is when different subsets of the hypothesis space are applicable to different subconcepts. In this case, the hypothesis space has to be partitioned into subsets corresponding to subconcepts. The second case is the focus of this thesis, when a part of each example is applicable to a subconcept. Here, we have to partition each individual example into smaller substrings corresponding to subconcepts.

Domain Decomposition

Examples for the subconcepts are drawn from space $X \times \{0,1\}$. It is possible for some of these examples to be relevant for a particular subconcept. In this case each subconcept c_i has a probability distribution D_i on X. Define the *relevant set* for a subconcept as the subset of the sample space that is relevant to learning the subconcept. The elements of this set would have a non-zero probability of being drawn. Formally, the relevant set for a subconcept c_i is

$$R_i := \{x \mid x \in X, D_i(x) \neq 0\}.$$

If we knew the relevant sets, we could classify examples perfectly and we would have learned all the subconcepts. So the learning task becomes one of classifying each drawn example to a subconcept, until we have seen enough examples of each subconcept to have learned them all within the required accuracy and confidence parameters.

Example Decomposition

Consider an example $x \in X$ of length k. Since we are considering Boolean concepts in this thesis, x is a string of bits $b_1 b_2 \ldots b_k$. For a subconcept of c_t, it is possible that only a part of x would suffice. In this section we will consider the case where a substring of each example is relevant for a particular subconcept. The definition of relevant set changes in this case. The subset of bits in an example that provides an example for a subconcept forms its relevant set. We assume that these bits are always contiguous (discussed later as the assumption of contiguity) so the relevant set would, in fact, consist of the bits of a substring.

Definition 4 (Relevant set). *We say that bits* $\{b_a, b_{a+1} \ldots b_{a+u}\}$ *in an example* $x \in X$ *provide the relevant set,* R_i*, for a subconcept* c_i *if the substring in that location provides a relevant example for* c_i.

So the bits of substring of length u belong to the relevant set, $b_a, b_{a+1} \ldots b_{a+u} \in R_i$, for a subconcept c_i. We shall refer to the relevant substring location for c_i as x_{c_i}. The relevant sets of interest are the largest non-spurious sets within an example.

Example 1 *The HIV (Human Immunodeficiency virus) has a very short life cycle, which may be as short as 1.5 days. It also lacks proofreading enzymes to correct errors during the process of reverse transcription. These two factors give HIV a very high mutation rate. Thus a combination of three or four anti-retroviral drugs, called Highly Active Anti-*

Retroviral Therapy (HAART), is given to patients. This is more effective than trying one drug after another, to which the virus tends to develop quick immunity.

The reason for the success of this treatment may be understood through example decomposition. Since evolution is a learning process, the evolution of the HIV virus may be viewed as a 'learner' that is trying to model an immune virus. Consider HAART therapy consisting of four drugs, $d_1, d_2, d_3,$ *and* d_4*. In this case, the 'example' that the learner sees is* $d_1 d_2 d_3 d_4$*. If only the first drug is used, then the example the learner sees is* d_1*. Consider the treatment where a single drug is given, and changed to the next one only if treatment begins to fail. In this case, we have effectively decomposed the example into its four sub-examples. This allows the virus to learn through decomposition, greatly reducing the time it takes to develop immunity.*

We make the following assumptions while analyzing example decomposition in this thesis. Some of these assumptions reflect commonly encountered conditions in learning, and yet others simply facilitate analysis. A point to note is that these assumptions define the class of learning problems we consider. To apply this framework to a different class of problems, we would start simply by revising this assumption set.

• **Assumption of total relevance:** We assume that every subconcept $c_i \in c_t$ has a non-null relevant set in every example for c_t.

• **Assumption of consistent relevance:** We assume that the relevant set for a given subconcept $c_i \in c_t$ is consistently found in the same location in the string for every example for c_t.

• **Assumption of contiguity:** We assume that the bits in R_i for any subconcept c_i are contiguous, i.e. they belong to a single substring.

• **Assumption of non-overlap:** We assume that for some bit $b \in x$, where x is any example for c_t, if $b \in c_i$ and $b \in c_j$, then $i = j$.

In example decomposition, dividing the hypothesis space for the target concept into spaces for subconcepts involves dividing each example into substrings for the subconcepts. With the above assumptions, each example would be neatly divided into n non-overlapping substrings, one for each subconcept. Since the subconcepts come together to form a compete description of the target concept, there would be no part of the example left over in noiseless learning. So the task of dividing an example for n subconcepts is one of inserting $n-1$ markers in the string for the example. For a string of length k, the hypothesis space for inserting $n-1$ markers is $(n-1)\times(k-1)$. Assuming no other bias, this is also the sample complexity of example decomposition.

$$p_{ex}(k, \frac{1}{\varepsilon_{ex}}, \frac{1}{\delta_{ex}}) = ((n-1)\times(k-1)) \quad (1)$$

Learning the Subconcepts

Once the relevant substring R_i of example $x \in X$ has been established, it may be used to learn the subconcept c_i. At this point, this may be considered straightforward PAC learning. The upper limit on sample complexity of PAC learning a concept is given by

$$\frac{1}{\varepsilon}[(k+1)D_{VC}(C^{[k]})\ln(2)+\ln[\frac{1}{\delta}]]$$

Each example may be used to learn multiple subconcepts. In fact, with the assumption of total relevance (see previous subsection), each example may be used to learn every subconcept. For simplicity of analysis, we may assume that the average relevant substring for each subconcept is k/n in length for examples of length k and n subconcepts. We also assume that all the subconcepts need to be learned within the same accuracy and confidence parameters, ε_{sc} and δ_{sc}. Then the sample complexity of learning a subconcept is

$$p_{sc}(\frac{k}{n},\frac{1}{\varepsilon_{sc}},\frac{1}{\delta_{sc}})=\frac{1}{\varepsilon_{sc}}[(\frac{k}{n}+1)D_{VC}(C^{[\frac{k}{n}]})\ln(2)+\ln[\frac{1}{\delta_{sc}}]]. \quad (2)$$

With the assumption of total relevance, the sample complexity of learning all n subconcepts is also given by the above equation.

Amalgamation

The final task in learning by decomposition is to combine the learned subconcepts $c_1, c_2 \ldots, c_n$ to give the target concept c_t. This is achieved by combining the classifiers for

$c_1, c_2 \ldots, c_n$ conjunctively. So for an example x, if $(x_{c_1} \in c_1) \wedge (x_{c_2} \in c_2) \ldots \wedge (x_{c_n} \in c_n)$, then $x \in R_t$.

This sort of reassembly does not require any learning. So it does not influence the sample complexity of decomposition directly. However, in such a scheme, the error would be multiplicative. So,

$$(1-\varepsilon_{c_t}) = (1-\varepsilon_{sc})^n$$

$$\Rightarrow \sqrt[n]{(1-\varepsilon_{c_t})} = (1-\varepsilon_{sc}). \tag{3}$$

Similarly, for the confidence parameter, we have that

$$\sqrt[n]{(1-\delta_{c_t})} = (1-\delta_{sc}). \tag{4}$$

Conditions for Propitious Concept Decomposition

For a target concept to be propitiously decomposable, the aggregate sample complexity of decomposition, learning the decomposed subconcepts, and their amalgamation must be less than the sample complexity of learning the target concept without decomposition. So we have

$$p_{c_t}(k, \frac{1}{\varepsilon_{c_t}}, \frac{1}{\delta_{c_t}}) \geq p_{ex}(k, \frac{1}{\varepsilon_{ex}}, \frac{1}{\delta_{ex}}) + p_{sc}(\frac{k}{n}, \frac{1}{\varepsilon_{sc}}, \frac{1}{\delta_{sc}}). \tag{5}$$

Using equation 5 we can come up with the condition for propitious PAC decomposition in terms of the VC dimensions of the concept spaces.

Theorem 1. *Let a Boolean target concept* $c_t \in C^k$ *be learned within accuracy and confidence parameters* ε_{c_t} *and* δ_{c_t}, *using examples of size* k, *and by decomposition into* n *subconcepts of equal size, accuracy, and confidence parameters. Then the condition for propitious PAC decomposition is given by*

$$\frac{1}{\varepsilon_{c_t}}(k+1)D_{VC}(C^k) - \frac{1}{1-\sqrt[n]{(1-\varepsilon_{c_t})}}(\frac{k}{n}+1)D_{VC}(C^{\frac{k}{n}}) \geq$$

$$\frac{1}{\ln(2)}\left[(n-1)(k-1) + \frac{\ln(\delta_{c_t})}{\varepsilon_{c_t}} - \frac{\ln[1-\sqrt[n]{(1-\delta_{c_t})}]}{1-\sqrt[n]{(1-\varepsilon_{c_t})}}\right],$$

where $D_{VC}(C^k)$ is the VC dimension of C^k and $D_{VC}(C^{\frac{k}{n}})$ is the VC dimension of $C^{\frac{k}{n}}$.

Proof. Substituting equations 1 and 2 in equation 5, we get

$$\frac{1}{\varepsilon_{c_t}}[(k+1)D_{VC}(C^k)\ln(2) + \ln[\frac{1}{\delta_{c_t}}]] \geq$$

$$(n-1)(k-1) + \frac{1}{\varepsilon_{sc}} \times$$

$$\left[(\frac{k}{n}+1)D_{VC}(C^{\frac{k}{n}})\ln(2) + \ln[\frac{1}{\delta_{sc}}]\right].$$

Substituting from 3 and 4 in the above equation, we get

$$\frac{1}{\varepsilon_{c_t}}[(k+1)D_{VC}(C^k)\ln(2) + \ln[\frac{1}{\delta_{c_t}}]] \geq$$

$$(n-1)(k-1) + \frac{1}{1-\sqrt[n]{(1-\varepsilon_{c_t})}} \times$$

$$\left[(\frac{k}{n}+1) D_{VC}(C^{\frac{k}{n}}) \ln(2) + \ln[\frac{1}{1-\sqrt[n]{(1-\delta_{c_t})}}] \right]$$

$$\Rightarrow \frac{1}{\varepsilon_{c_t}}(k+1) D_{VC}(C^k) - \frac{1}{1-\sqrt[n]{(1-\varepsilon_{c_t})}}(\frac{k}{n}+1) D_{VC}(C^{\frac{k}{n}}) \geq$$

$$\frac{1}{\ln(2)}\left[(n-1)(k-1) + \frac{\ln(\delta_{c_t})}{\varepsilon_{c_t}} - \frac{\ln[1-\sqrt[n]{(1-\delta_{c_t})}]}{1-\sqrt[n]{(1-\varepsilon_{c_t})}} \right].$$

This is the desired result.

Theorem 1 gives us the condition for propitious PAC decomposition in terms of VC dimension. We may use this relationship to obtain the condition for propitious decomposition in terms of the cardinalities of the concept spaces for the subconcepts and the target concept.

Theorem 2. *Let X be a finite set and let F be a class of concepts on X such that all members of F have length k. If $d = D_{VC}(C^k)$ is the VC dimension of F, then*

$$2^d \leq |F| \leq (2^k + 1)^d.$$

Proof. We know from [Nat91] that

$$2^d \leq |F| \leq (|X| + 1)^d.$$

Since all members of F have length k, X can have cardinality of at most 2^k. Substituting this value in the equation above gives the desired result.

Theorem 3, *Let a Boolean target concept $c_t \in C^k$ be learned within accuracy and confidence parameters ε_{c_t} and δ_{c_t}, using examples of size k, and by decomposition into n subconcepts of equal size, accuracy, and confidence parameters. Then the condition for propitious PAC decomposition is given by*

$$\frac{1}{\varepsilon_{c_t}}(k+1)\frac{\log_2(|C^k|)}{\log_2(2^k+1)} - \frac{1}{1-\sqrt[n]{(1-\varepsilon_{c_t})}}(\frac{k}{n}+1)\log_2(|C^{\frac{k}{n}}|) \geq$$

$$\frac{1}{\ln(2)}\left[(n-1)(k-1) + \frac{\ln(\delta_{c_t})}{\varepsilon_{c_t}} - \frac{\ln[1-\sqrt[n]{(1-\delta_{c_t})}]}{1-\sqrt[n]{(1-\varepsilon_{c_t})}}\right].$$

Proof. Taking $\log_2$ of the inequalities in theorem 2,

$$d \leq \log_2(|F|)$$

$$d \geq \frac{\log_2(|F|)}{\log_2(2^k+1)}$$

So

$$\frac{\log_2(|F|)}{\log_2(2^k+1)} \leq d \leq \log_2(|F|) \qquad (6)$$

Substituting the above values of d for the VC dimensions in theorem 1, we get

$$\frac{1}{\varepsilon_{c_t}}(k+1)\frac{\log_2(|C^k|)}{\log_2(2^k+1)} - \frac{1}{1-\sqrt[n]{(1-\varepsilon_{c_t})}}(\frac{k}{n}+1)\log_2(|C^{\frac{k}{n}}|) \geq$$

$$\frac{1}{\ln(2)}\left[(n-1)(k-1)+\frac{\ln(\delta_{c_t})}{\varepsilon_{c_t}}-\frac{\ln[1-\sqrt[n]{(1-\delta_{c_t})}]}{1-\sqrt[n]{(1-\varepsilon_{c_t})}}\right].$$

This is the desired result.

This theorem gives the strict bound which, if satisfied, gives the condition for propitious PAC decomposition in terms of cardinalities. However, it assumes that the learning machine used for learning the target concept is the weakest possible, while the one used for learning the subconcepts is the strongest possible. This does not reflect real world conditions. We must consider the relationship between the VC dimensions $D_{VC}(C^k)$ and $D_{VC}(C^{\frac{k}{n}})$. The machine used to learn the target concept must be at least as powerful as the machine used to learn the subconcepts. That is to say,

$$D_{VC}(C^{\frac{k}{n}}) \le D_{VC}(C^k). \tag{7}$$

This condition must be kept in mind while applying theorems 1 or 3. In fact, in most cases the machine used to learn the subconcepts would be significantly weaker than the machine used to learn the target concept.

There is one point of note about the results we obtained in this section. Despite the increased accuracy and confidence requirements of learning the subconcepts, these results tell us that in most cases it is highly desirable to decompose a subconcept. The reason for this attractiveness is the low cost of example decomposition, given by equation 1, for the class of learning tasks considered in this thesis. For other classes of

learning tasks, this cost may be too high justify decomposition. We discuss this cost further as the 'relevant set detection problem' in later chapters.

CHAPTER 4: FEATURE EXTRACTION

Concept learning by example decomposition as presented in this thesis is a meta-theory. It cannot directly be used on data; instead it can be used to design algorithms which in turn work with raw data. Another way this meta-theory may be used is to generate or explain existing theories.

In this chapter we use our work to develop a theory for an important field of research, feature extraction. Though successful empirically, there is no common theoretical background for feature extraction as a whole. Many important questions need answering. These include: What causes the emergence of features in data? How do these features interact? How may these features be detected in a general setting? And finally, what exactly is a feature? Feature extraction is primarily thought of as an image recognition field, but its scope extends to all learning problems. Popular algorithms like PCA, used in diverse fields, perform feature extraction. In this chapter, we translate our theoretical terminology and framework to reason about feature extraction. This framework is used to develop some new results, including some constructive theorems and an upper limit on the number of features possible in a given example set. We also use the framework to develop an algorithm for feature extraction from scratch.

Introduction

Feature extraction is the process of generating a set of characteristic attributes from a given dataset. Feature extraction is primarily an empirical science, with little theoretical background. Most of the theoretical work pertains to individual algorithms [HKCWL03],

[LLS02], [MM05] or low-level feature extraction [NA02], [Foe94]. This work aims to provide a unifying understanding of feature extraction as a pervasive learning process present across many disciplines. A feature is simply a regularity existing in a given data set, and as such may be applied to any process like various kinds of learning, theory building, psychology, or image recognition. We use the specific term 'feature extraction' because of the top-down pattern-recognition like nature of our theoretical setup (to be introduced in later sections). Generally speaking, feature extraction involves constructing a predictive hypothesis on any (possibly non-visual) data. This hypothesis is the extracted feature. For example, the divine proportion [Hun70] may be thought of as a commonly occurring feature in the natural sciences. Repeated geometric shapes may be considered features in image recognition. The concept of momentum is a feature on a pool table. Most science involves the extraction of abstract features by looking at raw data, and then finding interconnections among those features.

Feature extraction has not been studied sufficiently in a general theoretical setting. Even the definition of what exactly is a feature has not been satisfactorily answered. What characterizes a feature? How do features come into being? How can we identify features? Are there any general properties present in all features? What sort of relationships may exist among features, and how may they be discovered efficiently? These are all interesting questions with wide applications, but have not been studied adequately. One of the reasons why a broad theoretical treatment of feature extraction has been overlooked so far is the pervasiveness of the field. Wide usage and application of the term makes it difficult to find common ground. Another reason is that the field is

essentially new. Most feature extraction is computationally expensive, and efforts thus far have been focused on computational efficiency rather than theoretical thoroughness.

The research presented here takes initial steps towards a more complete theoretical understanding of feature extraction in this thesis. New terminology and theoretical framework for feature extraction is introduced. The approach in this thesis is fundamentally different from other work in that we try to understand features not by their properties, but by first principles. A feature is studied from the vantage points of Kolmogorov complexity and computational learning theory. Once a feature is clearly defined, we study the emergence of features, which yields a framework to understand the relationships among features. To seed our framework, we shall borrow some concepts from computational learning theory.

Layout of the Chapter

We use the framework developed in previous chapters to form a theory of feature extraction. This is achieved by understanding how features emerge in the training data, and what relationships among features look like. When these questions are answered in our terminology, a comprehensive picture begins to emerge. We complete the theory by describing the process of feature extraction in our terminology.

Since a framework for feature extraction is a meta-theory (being a theory about a kind of learning), the first thing it would produce are theoretical results. These theoretical results may then be used to obtain applied results. Thus, we start by developing some

constructive theorems using our framework. We also provide an upper limit on the number of features possible in a given dataset.

Finally, we analytically develop a learning algorithm using our framework.

A Framework for Feature Extraction

Consider examples $\{x_1, x_2 \ldots x_n\}$ from an instance space X presented to a feature extraction algorithm. Feature extraction works on the principle that there is localization of some property within each example, or across multiple examples. This localization of a computational property (or regularity) leads to the identification of a feature. These regularities in the instance space are caused by an underlying target function. To understand these regularities, let us consider the target concept causing them. The target function t is

$$t: Z \rightarrow X.$$

Z is the domain of t, and X is the instance space. Since t is a function, there exists a program $P(t)$ for it. $P(t)$ draws its input from Z and provides output to X. Let us call this program $P(t)$ the target program.

Definition 1: Let t be a target function

$$t: Z \rightarrow X.$$

The **target program** $P(t)$ is the smallest program for t.

The length of the target program should be as small as possible. This is in keeping with Occam's razor. Not only is excess code wastage, a smaller description also tends to be the right one [Cha75] [Lev74]. So we propose the following restriction on *P(t)*.

Constraint 1: The length of *P(t)* is minimal.

Next we study the target program to see how features arise in data.

The Causes of Features

In this section, we develop a framework to reason about features. Let us consider what a feature is. Through all the definitions and usages of the word feature, the common theme is that the elements belonging to a feature are somehow similar to each other, and different from the elements not belonging to that feature. So the primary quality of a feature is that there is something that distinguishes it from other entities. Stating this in computational terms allows us to define a feature.

Definition 2: A **feature** *f* in an instance space *X* is a localization of some computational property among multiple elements of *X*.

This definition of a feature provide rigor to our intuition. It is important to note that a feature is always spread over multiple elements. Even in cases like image recognition of a landscape, where the all the features, such as a regular shape leaves etc, may be present within a single example image, the feature is spread over multiple pixels. An example of a feature that exists among multiple members of *X* would be a cluster of two-dimensional

points. The instance space here consists of coordinates for each point, and the feature exists over a subset of such points as a cluster. So the localization of the feature-property may be in either a single member of X or among a subset of X, depending on how X is organized. But this does not affect the definition of a feature.

Since it is the target program that causes regularities in the dataset, studying it helps us understand the emergence of features. By constraint 1, the length of the target program is minimal. Assume there is some section of code that needs to be computed multiple times. How will this section of code be represented in the target program? We may not simply write out this section multiple times, because that would cause redundancy, and we may not allow a minimal program to be redundant. Any computation that has to be performed more than once in a minimal program must be expressed in the form of a subroutine. There is a lower limit on the size of a subroutine. The savings in size provided by creating a subroutine must be greater than the cost of naming and calling it, else the minimalism constraint is violated.

We formalize the idea of a subroutine with the help of the notion of a datapath. The datapath of a unit of data, b_0, in program p is the path that b_0 traces through the program. It is the sequence of statements that use b_0 in their input in direct or computed form.

Definition 3: The **datapath** of a unit of data b_1 in program p is the sequence of statements (s_1, s_2... s_n) such that

$b_1 \in$ input(s_1), and

output(s_i) $\in$ input(s_{i+1}), for all $1< i <n$.

Each datapath may be viewed as a string of statements. A subroutine would exist for repeated computation, i.e. if the datapaths of two separate inputs have a common substring. Consider a target program *P(t)* with input vector (b_1, b_2 ... b_k) with corresponding datapaths ((s_{11}, s_{21}... s_{n1}), (s_{12}, s_{22}... s_{n2}) …, (s_{1k}, s_{2k}... s_{nk})). Assume that the datapaths of any two inputs share a common substring (s_i, s_{i+1} ... s_{i+p}) of length p; and $p>c$, where c is the small constant cost of creating and calling a subroutine. Then (s_i, s_{i+1} ... s_{i+p}) must be written only once in $P(t)$ in the form of a subroutine.

These naturally emergent subroutines are very important, as it turns out that these subroutines are the causes of features in the instance space. Intuitively speaking, each subroutine is a small program performing a computational task. Thus a subroutine S will impart the computational characteristics of the task it performs to each input that passes through it. The set of outputs of S will share a common computational property, imparted by S. By definition, the set of outputs of S are now part of a feature. So subroutines correspond to features in the instance space.

Theorem 1: A given feature f in instance space X is associated with a corresponding subroutine S_f in the target program for X, and vice versa.

Proof: Proof follows from definitions. We first show that every subroutine causes the emergence of a feature. Then we show that every feature would have an associated subroutine.

a) A subroutine in the target program always causes the emergence of a feature:

A subroutine, by definition, is used multiple times. Thus, for a subroutine S_f in the target program for X, there would be multiple elements $\{x_{S1}, x_{S2} \ldots x_{Sk}\}$ in X that are influenced by S_f. And since all these elements have passed through the same computational procedure (the subroutine S_f), they would all share a computational property imparted by S_f. Thus $\{x_{S1}, x_{S2} \ldots x_{Sk}\}$ would be the elements of a feature, by definition of a feature.

b) A feature always has an underlying associated subroutine in the target program:

Let a feature f be present in the elements $\{x_{f1}, x_{f2} \ldots x_{fk}\}$ of X, where $k>1$. By definition of a feature, these elements share some computational property. Since a computational property may only be imparted by a computational procedure, the elements $\{x_{f1}, x_{f2} \ldots x_{fk}\}$ pass through the same computational steps $\{s_1, s_2 \ldots s_p\}$ in the target program for X. This series of steps $\{s_1, s_2 \ldots s_p\}$ is computed at least k times in the target program, once for each element. In order to avoid violating the minimalism constraint on the target program this series of steps must be expressed as a subroutine of length p that would impart the feature-property to $\{x_{f1}, x_{f2} \ldots x_{fk}\}$.

Q.E.D.

Since a subroutine in the minimal target program is always associated with a feature and vice versa, they may be viewed as integral parts of each other.

Definition 4: The **associated subroutine** S_f for a feature f is the subroutine that imparts the feature's computational property to the elements of the feature.

Definition 5: The **feature set** $\{f\}$ for a given subroutine S_f is the subset of instance space that is computed by S_f.

The feature set may alternatively be defined simply as the elements of a feature.

Relationships Among Features

In image recognition applications, an often-overlooked element is the relationships between the different features. Relationships between features become of prime importance if we are considering features generally across different fields. For example, in science, the first step is to collect data; then comes the process of theory building, which involves extracting features from the data, and finding out the relationships between them. It is these relationships that give science its inferential and predictive powers. As many scientists would attest, the processes of discerning features and relationships among them are interrelated, and one often aids the other. We shall be able to formalize this idea in later sections. Features in a given instance space may be related to each other, and influence each other. Before we study the nature of these relationships, our framework allows us to define 'influence' better.

Definition 6: A feature f **influences** another feature g if the output of the associated subroutine for f, S_f serves as input in some form for associated subroutine for g, S_g.

Note that the output of S_f need not directly serve as input for S_g. There may be intermediate computational steps, or the output of S_f may pass through another subroutine

before reaching S_g. The calling order of the subroutines forms a web of influence, the structure of the target program. This web is the structure of relationships between the features. The study of this web yields insights into how knowledge is organized.

In this web, some associated subroutines are closer to the instance space than others. This leads us to the idea of the order of a feature. The order of a feature is, informally, its distance from the instance space. Lower order features can be extracted relatively easily, while higher order features require more work.

Definition 7: the **order** of a feature *f* is given by the following

1. A feature whose associated subroutine provides output directly to the instance space is order 1.
2. A feature whose associated subroutine has output linked to a subset of the instance space and/or input of other subroutines has order n+1, where n is the highest order of all the features it influences.

The idea of the order of a feature corresponds intuitively to the idea of the complexity of learning increasing with logical depth.

Two features overlap if their feature sets have some common elements.

Definition 8: Two features *f* and *g* are said to **overlap** if, for their feature sets $\{f\}$ and $\{g\}$,

$$\{f\} \cap \{g\} \neq \varnothing$$

The target program is a network of associated subroutines. Since higher order features are more difficult to extract than lower order features, it is useful for learning purposes to stratify this web by putting all features of a given depth into one layer. Layers quantify the complexity of learning associated subroutines.

Definition 9: A **layer** *k* of features in an instance space *X* is the set of all features of the order *k*.

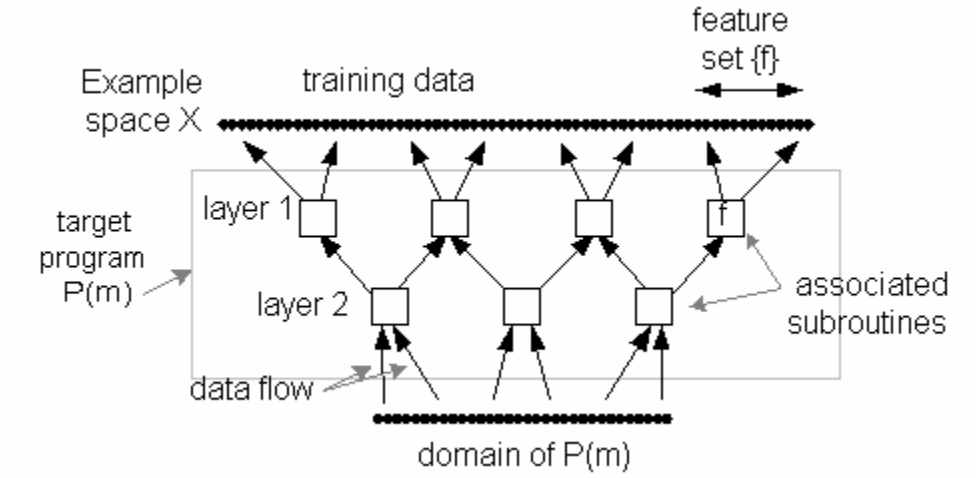

Figure 1: Target program structure

The terminology introduced in preceding sections defines concepts in feature extraction clearly and unambiguously. This allows us to talk about feature extraction in rigorous terms. The framework introduced above (i.e., the target program structure and associated definitions) does more than provide an understanding of how features emerge in data.

This structure helps us understand properties of features. One property that we have already touched upon is that lower order features are easier to discover than higher order features. In later sections we shall study some more such properties. Every little thing that we know about the target function helps us in learning it by acting as bias [Utg86]. So we hypothesize that these insights will ultimately help in the design of new feature extraction algorithms, and more comprehensive understanding of current ones.

Feature Extraction as a Process

In this section, we talk about the process of feature extraction as a whole. We shall use the terminology we have developed so far to sharpen our understanding of what goes on during feature extraction. The process of feature extraction is one of reverse engineering the target program from the training data. We look at the given data, identifying a pattern in some subset of the data, and then come up with a computational model to explain or generate that pattern.

Definition 10: **Feature extraction** for a feature f is the process of modeling the associated subroutine S_f as closely as possible by first discerning the feature set $\{f\}$ from the instance space X, and then by passing $\{f\}$ as input to some learning method L.

But there may be multiple related features in a given dataset. In this case we also have to discover these relationships to provide a complete understanding of the data. We may broadly divide the task of feature extraction into the following three steps.

1. Feature set detection
2. Individual feature extraction from a feature set

3. After multiple features have been extracted, learning the relationships between the features

Learning an individual feature involves modeling its associated subroutine by looking at the feature set it generates. Of course, we have to discover the feature set first. Given below is an algorithm for extracting a single feature from *X*.

ALGORITHM: EXTRACT SINGLE FEATURE

Given: Instance space *X*, distance metric *D*, transform *T*, Learning algorithm *L*

Output: A feature *f* in *X* demarked by distance metric *D*

Given the instance space *X*, use transform *T* to transform *X* into an appropriate form, X^T

1. Use a clustering algorithm *C* using distance metric *D* to discover feature set $\{f\}$ in X^T
2. Use learning algorithm *L1* to discover the computational structure, S_f, of feature *f* in $\{f\}$

END ALGORITHM: EXTRACT SINGLE FEATURE

L1 may be any learning algorithm. This algorithm assumes the existing knowledge of a distance metric *D* and a transform *T*. However, in actual algorithms they have to be specified by a human expert or discovered using a learning algorithm. Both *D* and *T* are related to S_f, and thus in some cases may be thought of as being discovered along with it. Because of the fine distinction between *D*, *T*, and S_f, they are commonly understood to be

the same thing, and are usually lumped together into the hypothesis for the feature. But appreciating the differences between them helps us better understand feature extraction.

Relationship Between *D* and S_f: The associated subroutine S_f imparts some property to all members of $\{f\}$ making them computationally similar in some way. This property may be used as a distance metric D for a clustering algorithm C; in fact it would be an ideal problem-specific choice if we could find it. Thus if we find a highly successful distance metric, it may be a clue to the computational property that caused the cluster, giving us a clue for S_f. This means that we may use learning method L to discover D and S_f as the same thing. Theorems 2, 3, 4, and 5 give us some problem independent distance metrics.

Relationship between *T* and S_f: The transform T is used to bring the input data set into a form more conducive for feature extraction. This may be viewed as 'decoding' the input data before learning. Since both T and S_f are computational procedures performed on $\{f\}$, in some cases they can be appended together and thought of as one. They are, however, different. An example would be where T is the Fourier transform. Here S_f may be completely independent of T.

Relationship between *T* and *D*: The purpose of T is to prime the input data for the next step, clustering. So the choice of T should be such that clusters would readily and correctly form when distance metric D is used.

To extract all the features in a given instance space, we may repeatedly call the 'extract single feature' method.

ALGORITHM: EXTRACT ALL FEATURES

Given: Instance space X

Output: The features in X and their interrelationships

1. While there are features left to discover
    a. Call EXTRACT SINGLE FEATURE to discover feature f_i
2. For $i = 1$ to n
    a. For $j = i+1$ to $n-1$
        i. Use learning method *L2* to discover relationship between features f_i and f_j

END ALGORITHM: EXTRACT ALL FEATURES

The 'extract all features' algorithm serves to illustrate the different steps required to extract features. Most algorithms perform these tasks in one form or the other, but not necessarily in the order or exact form suggested above.

The first step in this algorithm is to extract all features in the data. The problem here is that we do not know the number of features in the dataset in advance. Usually, the algorithm would run until it could not discover any new features in the data. This does not mean there are no features left, it means that there may be features but they are too complex to be elicited by our algorithm. The number of features in the dataset needs to be

estimated by existing algorithms like SIFT [Low99], and there are no theoretical guidelines as to what this limit should be. A human expert usually sets it. Later in this thesis, we come up with an upper limit to the number of features possible in a given dataset.

The algorithm 'extract all features' is a bit naïve. We may write cleverer algorithms that take advantage of multiple features in an embedded environment. We present one such algorithm in later sections when we discuss feature extraction using mutual information.

Results and Applications

In this section we provide some constructive theorems, which may be used to create or enhance feature extraction algorithms. Also, we derive an upper bound on the number of features possible in a given dataset. We discuss the learning of features in light of our framework. Finally, we analytically develop a new method for feature extraction using our framework.

How to Extract a Feature

In the framework developed in the previous sections, we looked at the idea of a feature set sharing some property. We looked at the program/subroutine generating the set. Now we consider what it would look like to a feature extraction algorithm.

It is interesting to note that if the regular parts of the instance space were represented as a string by appending its elements together, then the target program would be the Kolmogorov complexity of that string [LV97].

Theorem 2: All elements of a feature set have similar Kolmogorov complexity, unless the elements overlap with another set.

Proof: Since the target program *P(t)* is minimal, all subroutines $S_{f1}, S_{f2}, \ldots S_{fn}$ in *P(t)* are minimal for the tasks they accomplish. Consider a feature set $\{f\}$ consisting of elements $\{x_{f1}, x_{f2} \ldots x_{fk}\}$ of X, where $k>1$. The Kolmogorov complexity K_i for $\{f\}$ is its datapath, or the length of the subset of target program *P(t)* it passes through. For each element x_i of $\{f\}$ that does not belong to any other feature,

$$K_f = |S_f| + c_f$$

Where c_f is some constant and $|S_f|$ is the length of S_f.

Q.E.D.

Theorem 2 goes towards providing a theoretical understanding of the 'sameness' of feature set elements. Theorem 2 would be constructive if we used some method sensitive to Kolmogorov complexity, for example Kolmogorov-Levin complexity [Sch97]. However instead of an abstract approach like Kolmogorov-Levin complexity, we discuss some more commonly used, well-understood properties that arise due to the sameness of feature set elements.

In the previous section we saw some general algorithms to extract features. But how do we translate these algorithms into practice? We need practical ways of determining feature sets. We need usable values for transform *T* and distance metric *D*. Of course, we cannot have the best distance metric, S_f. If we knew this property we would actually have

the associated subroutine for the feature and the process of learning would be complete. However, using our framework we can come up with some indirect properties of features, such as kurtosis, that may be used to discover them. Kurtosis of a sample set measures how the values in that set are concentrated around the center of the distribution. Thus it measures the 'peakedness' of a sample set. The kurtosis of all elements in a feature set would tend to be similar, since those elements have passed through the same datapath.

The structure of the target program forms a directed graph. The nodes of this graph are the associated subroutines, and the edges represent the calling order of the subroutines (or, equivalently, the data flow). The directions of the edges are given by the direction of data flow. There would always be a root node representing the input to the first subroutine in the target program. The output of this initial subroutine would flow (possibly through various other subroutines) to the instance space, generating features in the training data. This flow would form a sub-graph in the minimal program. Certain properties impose a partial order on this structure. Examples of such partial orders form the core of the next three theorems.

Theorem 3: Higher order features have greater influence than lower order features in a sub-graph.

Proof: Consider the associated subroutine of a feature f with order $k>1$. Since a subroutine is used multiple times, it influences at least two features, forming at least two sub-graphs. Every feature in either one of these sub-graphs will be influenced by f. Thus f will have more influence than any lower order feature in its sub-graphs.

Q.E.D.

The next theorem deals with the rate of change of feature set elements in a temporal system. First, we offer a straightforward lemma.

Lemma 1: The elements of a feature set have similar rates of change.

Proof: Follows from theorem 2.

The less obvious property arises when we look beyond individual features, and consider a web of features embedded in a system.

Theorem 4: The rate of change of feature set elements increases monotonically with decreasing order of the feature in a sub-graph unless

- **a.** The target program directly modifies the rate of change.
- **b.** The changes negate each other.

Proof: Feature set elements are the output of the features' associated subroutine. For a given feature *f* in layer *n*, let *f* be influenced by features $g_1, g_2 \ldots, g_k$. Assuming that the changes do not negate each other and that the target program does not modify the rate of change (e.g., by setting its input to zero), then the rate of change for *f*, $R(f)$, is in the range

$$MAX(R(g_1), R(g_2) \ldots, R(g_k)) \leq R(f) \leq R(g_1) + R(g_2) + \ldots R(g_k)$$

Since the rate of change is additive

$$R(f) = a_0 * (\text{influence on } f),$$

where a_0 is some constant

Then theorem 2 gives us

$R(f) = a_1$ / (order of f),

where a_1 is some constant.

Q.E.D.

We may impose a similar property on kurtosis of a feature set.

Theorem 5: Kurtosis of a feature set increases monotonically with decreasing order of the feature unless directly modified by the target program.

Proof: Consider a feature f with kurtosis $Kurt(f)$. Let f be influenced by features $g_1, g_2 \ldots, g_k$. $Kurt(f)$ is in the range

$$MIN(Kurt(g_1), Kurt(g_2) \ldots, Kurt(g_k)) \leq Kurt(f) \leq (1/k^2) * \Sigma_i\, Kurt(g_i)$$

Assuming $g_1, g_2, \ldots g_k$ have similar variances.

From the above relation

$Kurt(f) = a_0$ * (influence on f),

where a_0 is some constant.

Then theorem 2 gives us

$Kurt(f) = a_1$ / (order of f),

where a_1 is some constant.

Thus every instance of multiple influences tends to increase the kurtosis of the target, unless the target program directly manipulates the kurtosis.

Q.E.D.

The theorems given above are constructive in nature as they provide bias for feature extraction algorithms. They may be used by themselves or in conjunction with other biases to extract features.

A problem we discussed earlier was estimating the number of features present in a given dataset. A theoretical upper bound does not exist in current literature. However such a value is required in many learning algorithms such as SIFT [Low99]. A simple combinatorial count where every combination could be a feature yields the number of features possible in a given instance space X of cardinality p to be 2^p-1, which is the number of non-empty subsets possible in X. However, it is intuitively clear that all such combinations could not represent useful features. Our intuition turns out to be right; the number of useful features possible in a given dataset is much smaller. We derive such an upper bound below.

For convenience, let us assume that the size of each element of X is uniform, given by s.

Lemma 2: The size of a minimal program cannot be greater than the size of the output it produces within some small additive constant.

Proof: This is a basic result from Kolmogorov complexity. Let some output O be produced by a minimal program P. If we view O as a string, then the size of P would be the Kolmogorov complexity of O. The Kolmogorov complexity of a string cannot be greater than the length of the string itself, within a small additive constant. Thus,

$$|P| \leq |O| + c$$

Where c is a small constant.

Q.E.D.

Theorem 6: The upper bound on the number of features, n, possible in a given instance space X, where s is the size of each element, is given by

$$n*log(n) < |X|*s$$

Proof: The maximum number of subroutines present in a target program would be give by

(Maximum target program size) / (minimum subroutine size) (1)

The size of the instance space is $|X|*s$, where $|X|$ is the cardinality of X and s is the size of a single element of X. By lemma 2 this is also the maximum size of the target program. Ignoring the constant, the upper limit of target program size is,

Maximum target program size = $|X|*s$ (2)

Now we calculate the minimum size of a subroutine. Since a subroutine is defined by its usefulness in saving space, the size of a subroutine should be at least more than the cost incurred in calling it. If there are at most n subroutines, then we need $log(n)$ bits to uniquely name them. So, for n subroutines

Minimum subroutine size > $log(n)$ (3)

From (1), (2), and (3)

$$n < |X|*s/log(n)$$

$$\Rightarrow n*log(n) < |X|*s$$

Q.E.D.

This upper bound includes the case where each example in the instance space would have different features to offer. Let us consider an important special case where the features present in each example would be exactly the same. For instance, in face recognition, the features such as eyes, nose, etc. may exist in every sample of the instance space. In this case, any one example could have all the features that are present in the entire instance space. Here, the target program may be considered to be the program that generates one example, and we would still have all the required associated subroutines. The upper bound on the number of features is greatly reduced in this case. Here, all the features possible in X can be present within a single (possibly idealized) element of X.

Theorem 7: The upper bound on the number of features, n, possible in a given instance space X is given by

$$n*log(n) < s$$

Where s is the size of an element x_i of X such that, for any feature f,

$$\text{if } f \in X, \text{ then } f \in x_i$$

Proof: The proof is similar to the proof of theorem 6 except that the maximum target program size is limited by the size s of sample x_i.

Q.E.D.

The framework introduced in this thesis eliminates chimerical or useless features, which are otherwise included in a simple combinatorial count. This is more reflective of the real world where an unnaturally large number of useful features do not exist. In fact,

extracting even a small number of useful features is a painful business. It is also important to note that the size, and not the cardinality alone, of the instance space are considered to derive this upper bound. This allows for the size and granularity of the elements of the instance space to be taken into account.

Feature extraction using mutual information

It would be a demonstration of the usefulness of our work if we could come up with novel methods of feature extraction from first principles using our framework. In this section, we develop a method to detect higher-order features in a much shorter time than the usual unsupervised blind search. The most interesting feature of this algorithm is that it is synthesized analytically using our framework.

If we have to learn one feature, there is little else we can do except picking the property setting it apart from other features. However, since the target program is a network of associated subroutines, we usually find many interrelated features in a dataset. Using this property we can come up with some additional schemes for extracting features. In the following section we use first-order features that have already been discovered to find higher-order features.

Consider an associated subroutine f_C with depth 2. Let the feature set of f_C on the training string be $\{f\}_C$. Assume that f_{c1} and f_{c2} are two associated subroutines with depth 1 that are completely influenced by f_C. Let their feature sets be $\{f\}_{c1}$ and $\{f\}_{c2}$ respectively. Then $\{f\}_{c1}$ and $\{f\}_{c2}$ will be subsets of $\{f\}_C$.

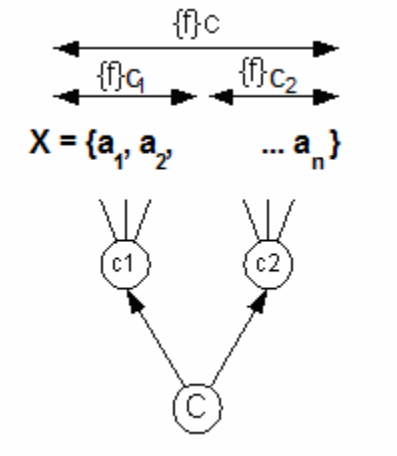

Figure 2: Common cause

Because f_{c1} and f_{c2} share a common factor, f_C, $\{f\}_{c1}$ and $\{f\}_{c2}$ will have some mutual information. This mutual information is actually information about f_C, since it stems from f_C. Once we have discovered f_{c1} and f_{c2}, we may extract this mutual information to learn f_C. This turns the process of unsupervised blind searching of f_C to one of gradient descent. Thus the discovery of higher-order associated subroutines may be facilitated by this mutual information.

ALGORITHM: EXTRACT HIGHER ORDER FEATURE

Given: Two discovered features f_{c1} and f_{c2}

Preconditions: f_{c1} and f_{c2} share some higher order feature f_C, all information about f_C is contained in f_{c1} and f_{c2}

Output: f_C

1. Let $\{f\}_{c1}$ be the input for a supervised learning algorithm *SL1*. Let $\{f\}_{c2}$ be the training signal for *SL1*.

   Let $\{f\}_{c2}$ be the input for a supervised learning algorithm *SL2*. Let $\{f\}_{c1}$ be the training signal for *SL2*.

2. Cease learning when outputs of *SL1* and *SL2* are sufficiently similar.
3. Present output of *SL1* as f_C

END ALGORITHM: EXTRACT HIGHER ORDER FEATURE

Since the training signal for $\{f\}_{c1}$ is $\{f\}_{c2}$, the only part of $\{f\}_{c1}$ that would be able to model itself after $\{f\}_{c2}$ is the mutual information between $\{f\}_{c1}$ and $\{f\}_{c2}$. This mutual information will be information about f_C. *SL1* and *SL2* may be separate instances of the same algorithm. The second precondition is not necessary; all information about f_C need not be contained in f_{c1} and f_{c2}. Even partial discovery of f_C is helpful. The method could be extended to more than two variables in case f_C is distributed sparsely over many features. Of course, in order to use this method, we still have to know which features share a common cause so that we can pass this subset to the supervised learning algorithm. We may know this by keeping a correlation matrix for all the features. Variations of this idea are used in some existing learning methods [Bec92], [JKF03].

Such mutual information depends on the richness of interaction among the associated subroutines. We contend that such richness is widely present in nature, making this a lucrative factor to model into algorithms dealing with real-world problems.
The most important thing about this algorithm, and its most alluring feature, is the fact that this algorithm was designed by analytical synthesis. If algorithms can be designed by deduction instead of induction, then it opens up a whole new avenue for designing learning algorithms. For instance, it may be possible to combine a deductive system (or a theorem prover) with this framework to design a learning algorithm generator.

Discussion on Feature Extraction

It is noteworthy that subroutines are defined by their being used multiple times. This extends to the defining property of a feature being that it is present in multiple elements. Such a definition based on utility has the advantage of eliminating chimerical features that may be included in a simple combinatorial count.

The question arises; will a target concept have one objective smallest program? Though it is unlikely that vastly different programs of the same Kolmogorov complexity would represent a concept equally well, it is intuitively obvious that in some cases we may move around the subroutines within a program to form a different program of the same size. Alternatively, a small section of a program may be rewritten in a different but equivalent way, forming a different program. In any case, if there are multiple programs to choose from, we may pick any one. All of the programs would be subject to the principles developed in this thesis. It would make an interesting direction for future research to study if it is possible for one such program to have more subroutines than another one; or if a certain amount of shared computation is inherent to a given task.

There are some deeper philosophical implications of this work. The upper limit on the number of features in a dataset presents an upper limit on the amount of knowledge extractable from the data. The presented structure for the target program also has implications for epistemology.

Rich Prevalence of Associated Subroutines in Nature

What does a target program typically look like in nature? Natural factors like redundancy, common causes, fanning out of causal chains, sharing of material, etc, all lead to shared computation. Thus we contend that associated subroutines are highly prevalent in reasonably complex natural systems. Most systems in nature consist of a large number of associated subroutines in a rich web of interconnections. This agrees with observations in the real world. For an example, consider any accepted physical, chemical, or biological theory. All such natural theories have a large number of commonly used concepts that influence other concepts or explain observations. These concepts are like associated subroutines and some of them were discovered in a manner similar to feature extraction. In fact, the only systems that appear to not exhibit such structure are either trivially simple systems, or artificial man-made problems like cryptography.

An Example - Face Recognition

We discuss face recognition by feature detection as an example of learning by example decomposition. In this case, X is the database of face images. x would be a single face. Face recognition by feature detection works by recognizing the salient features on a human face and then using them to classify the faces. So a subconcept would be a recognized feature on a face.

We describe a technique used in [BP93]. Each face is normalized and then represented by a database entry whose fields are a digital image of the face's frontal view and a set of four masks representing eyes, nose, mouth, and face (see figure 1). The location of the four masks relative to the normalized eye position is the same for the whole database. For

recognition, the unclassified image is compared with the images in the database, returning a vector of one matching score per feature. The unknown face is then classified as the one giving the highest cumulative score.

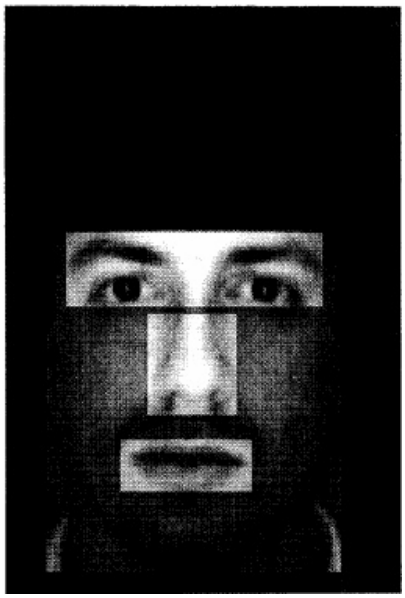

Figure 3: Different regions used for template matching (from [BP93])

In this example the four masks are the four *subconcepts*. The fact that the four masks are in the same location corresponds to the *assumption of consistent relevance*. Since these masks exist for each example, the *assumption of total relevance* holds true in this case. The *assumption of contiguity* also holds. However, the *assumption of non-overlap* does not hold as one of the masks is the face (face being the region below the eyebrows). This example takes advantage of many of the ideas discussed in previous sections. Classification of a mask is an *easier individual learning task* than classification of the whole image because of the reduced size. The *spurious parts of an example* (the region external to the masks) are ignored. Decomposing the image into masks allows the classification process to focus on much smaller areas. We know that the complexity of

this process increases exponentially with size. Thus the *hypothesis space is exponentially reduced* improving the classification success rate (by allowing better learning in a smaller mask) and simplifying the learning process. It is important to note that the computational complexity of learning a feature may not change, but the sample complexity does. Though it is not explicitly used in [BP93], these masks are embedded in an *environment of related tasks*. This fact may be used implicitly by employing the same learning algorithm for all masks with a high success rate. Knowledge of these subconcepts or, specifically, masks allows for *a better understanding* of what is important to face recognition. This may help in future research to further refine the masks.

CHAPTER 5: ADVANTAGES

We developed the condition for propitious PAC decomposition in the previous section. It turned out that this condition was rather easy to satisfy for the class of learning problems under discussion. In this section we analyze the reason why this is so, and also discuss some less obvious benefits of learning by example decomposition.

The savings arise from multiple sources. We discuss the following in this section.

- Exponentially reduced hypothesis space
- Learning in parallel
- Simpler individual learning tasks
- Subconcepts embedded in an environment of related tasks
- Exploitable relationships between subconcepts
- Detection and ignoring of spurious parts of examples
- Reuse of subconcepts
- A better understanding (e.g., credit assignment)

Each one of these ideas is practically exploitable in learning algorithms. This can be done by extracting *bias* [Utg86], or clues to build into learning algorithms. In fact, using bias is the only way we can statistically improve the performance of learning algorithms over blind search. The following discussion sheds light on some ways we may extract bias from example decomposition.

Exponentially Reduced Hypothesis Space

The most obvious way in which example decomposition provides bias is by splitting the hypothesis space among the subconcepts. The maximum cardinality for a concept space C^k for a concept c_t defined on examples of size k is 2^k. Whereas the maximum cardinality for a subconcepts' concept space is $2^{\frac{k}{n}}$, assuming n equal length subconcepts. This is an exponential reduction in hypothesis space.

Decomposing a concept into smaller concepts provides exponential savings in terms of the size of the hypothesis space. This translates to reduced sample complexity.

Learning in Parallel

With the assumption of total relevance in example decomposition, all the subconcepts are learned in parallel. This condition is reflective of a large class of natural problems where each example is descriptive of the entire system. Examples include face recognition, all biometrics (thumbprint recognition, cornea recognition, etc), temporal data from physical systems where each state reflects the whole system, etc. The total sample complexity will be that of the 'weakest link', the subconcept that requires the most examples. The other subconcepts would be learned before it. This also leads to a huge reduction in sample complexity.

Simpler Individual Learning Tasks

Since example decomposition splits one large concept into many smaller ones, it breaks down the *process* of learning into smaller chunks. Also to be considered is the fact that a linear decomposition in target concept size leads to exponential decomposition of the

hypothesis space. We can thus use less computationally intensive learning algorithms and weaker machines to learn the subconcepts one by one. In complex learning problems that call for inordinate computational resources, learning without decomposition may not be possible at all.

A learning machine would need access to a maximum hypothesis space of 2^k to learn a concept c_t defined on examples of size k. However, if the concept is decomposed into n equal sized subconcepts, then the learning machine only needs a hypothesis space of

$$\max\left(2^{\frac{k}{n}}, (n-1)(k-1)\right).$$

In physical terms, a machine would need exponentially reduced state space to learn by decomposition.

Example 2 Suppose a learning problem has $k = 100$, and $n = 10$ equal sized subconcepts. A learning machine for this problem without decomposition would need 2^{100} possible states. The worst case for the amount of storage required is 2^{101}. The computation would be made even more inefficient given that 100 bits would be required to represent a state, which is greater than the width of most modern processor registers. However, with example decomposition, the worst case for storage space for a single subconcept is 2^{11}. Given 10 subconcepts, even if all of them have to be stored simultaneously, the storage requirement is 2^{12}. Each state would require 10 bits for representation, which is subject to fewer hardware constraints.

Example 3 Humans can hold 7 ± 2 objects in their short term memory. Given this limitation, complex or voluminous learning would be impossible for humans without focusing on independent subproblems, abstraction, or chunking. All of these can be expressed as learning by decomposition of one form or another.

Subconcepts Embedded in an Environment of Related Tasks

Baxter [Bax00] showed that bias can be automatically learned for learning tasks from the same environment. This meta-learning helps the learning of later tasks. The subconcepts exist in space of related problems. As such, the framework for meta-learning developed by Baxter is applicable to them. This may be used to refine the upper limit on the sample complexity of learning the subconcepts.

We have a set of probability distributions $P_1, P_2 \ldots, P_n$ on $X \times Y$ for each subconcept $c_1, c_2 \ldots, c_n$. This may be viewed as learning multiple related tasks embedded in an environment. Suppose we sample m times for each one of the n subconcepts, then we generate an $(n,m) - sample$. Refer to [Bax00] for details of definitions and terminology.

Theorem 1. Suppose X and Y are separable metric spaces and Q is any distribution on P. Suppose z is an $(n,m) - sample$ generated by sampling n times from P according to Q to give $P_1, \ldots, P_n$, and then sampling m times from each P_i to generate $z_i = \{(x_{i1}, y_{i1}), \ldots, (x_{im}, y_{im})\}, i = 1, \ldots, n$. Let H be any permissible hypothesis space family. If the number of tasks n satisfies

$$n \geq \max\left\{\frac{256}{\varepsilon^2}\log\frac{8C\left(\frac{\varepsilon}{32},\mathsf{H}^*\right)}{\delta},\frac{64}{\varepsilon^2}\right\}$$

and the number of examples m for each task satisfies

$$m \geq \max\left\{\frac{256}{n\varepsilon^2}\log\frac{8C\left(\frac{\varepsilon}{32},\mathsf{H}_l^n\right)}{\delta},\frac{64}{\varepsilon^2}\right\},$$

then with probability of at least $1-\delta$, all $h \in H$, will satisfy

$$er_D(H) \leq \hat{e}r_z(H) + \varepsilon$$

Proof. The proof for this theorem may be found in [Bax00] and will not be reproduced here.

We may simply restate the above theorem for subconcepts.

Theorem 2. Suppose X and Y are separable metric spaces and D is any distribution on $X \times Y$. Suppose z is an $m-sample$ generated by sampling $X \times Y$ m times according to D. Let H be a hypothesis space used to learn a target concept $c_t \in C$. If the number of decomposed subconcepts n satisfies

$$n \geq \max\left\{\frac{256}{\varepsilon^2}\log\frac{8C\left(\frac{\varepsilon}{32},\mathsf{H}^*\right)}{\delta},\frac{64}{\varepsilon^2}\right\}$$

and the number of examples m for each task satisfies

$$m \geq \max\left\{\frac{256}{n\varepsilon^2}\log\frac{8C\left(\frac{\varepsilon}{32},\mathsf{H}_l^n\right)}{\delta},\frac{64}{\varepsilon^2}\right\},$$

then with probability of at least $1-\delta$, all $h \in H$, will satisfy

$$er_D(H) \leq \hat{er}_z(H) + \varepsilon$$

Proof. Follows from previous theorem.

Exploitable Relationships Between Subconcepts

Having developed an understanding of the internal structure of a decomposed target concept, we may now use this knowledge to develop bias in a variety of ways. We may now exploit the web of influence between subconcepts to discover them. The following examples list a couple of ways.

The next example provides a less obvious case. It takes advantage of relationships between subconcepts to extract higher order subconcepts.

Example 4 Consider the case where we have three subconcepts, $c_1, c_2,$ and c_3; and c_3 influences c_1, and c_2. In this case c_1 and c_2 carry some mutual information about c_3. If c_1 and c_2 are learned, we can extract this information to obtain bias for learning c_3. This idea has been used in [Bec92]. One way to do this, for example, is to provide c_1 and c_2 as inputs to a neural network and use c_3 as a training signal. In this case, we may also perform efficient unsupervised learning by training c_2 and c_3 against each other, i.e.,

with c_3 as the training signal and c_2 as the input, or vice versa. This will cause the learner to settle on the mutual information between them, which is information about c_1.

A subconcept influences another if its state somehow influences the other. So, the state of its relevant set tells us something about the state of the other's relevant set. This idea may be used to extract bias for learning the influence variable. But to do so, we have to do away with the assumption of non-overlap. We discuss the cost of doing so in Chapter 6.

Definition 5 (Influence.) *A subconcept* c_i ***influences*** *subconcept* c_j *if*

$$R_i \cap R_j \neq \phi.$$

Additionally, the degree of overlap between the relevant sets would decide the degree of influence, but we save this idea for development in later work.

Example 5 *The most straightforward idea arises from the fact that, if two subconcepts influence each other, each influences the states the other may assume. So if a subconcept* c_1 *influences* c_2*, and we have already learned* c_1*, then the states that* c_1 *assumes influence the states that* c_2 *may assume as well. Thus knowledge of* c_1 *simplifies the learning of* c_2*.*

Detection and Ignoring of Spurious Parts of Examples

It is possible that not the entire example would be relevant to learning. So, a subset of bits in the example may be enough to classify it perfectly. In this case, the rest of the bits

are spurious, and increase the sample complexity and time complexity of learning. The sample complexity is increased because the hypothesis space is proportional to the size of the examples. The hypothesis space on examples of size k is 2^k. Also, larger examples take longer to read and process, increasing the time complexity.

Learning by example decomposition may prevent spurious parts of the example from being considered, as we determine the relevant set of each subconcept. If these relevant sets are optimal, then we end up considering exactly the useful part of an example.

Reuse of Subconcepts

It is possible for the same subconcept to be relevant for two separate substrings in the example. In this case we have to learn it only once. This sort of reuse is not easy in learning without decomposition.

Example 6 *Consider a function* $f:(a,b,c)\rightarrow z$ *being learned. Let* f *be*

$$z = \sin(a) * \tan(b) / \sin(c).$$

If we decomposed the example into sets $\{a\},\{b\},$ *and* $\{c\}$*, then it would be possible to learn the subconcept* sin() *once for* $\{a\}$ *and then reuse it for* $\{c\}$*. It only adds trivial complexity to check if available subconcepts classify other parts of the example as well. If we were learning the function without decomposition, this reuse would not have been simple without heavy bias.*

The success of this approach depends on the extent of reuse of subconcepts. The question is, how often can we expect to find repeated computation in learning problems? Natural

factors like redundancy, common causes, fanning out of causal chains, sharing of material, etc., all lead to shared computation in natural systems. Thus it seems likely that, at least within a natural system, we might encounter reuse of subconcepts.

A Better Understanding

Knowledge of the subconcepts gives us an insight into the internal structure of the target concept. This knowledge may be used in numerous ways to provide bias. For example, having decomposed the example for subconcepts, we can see which subconcept provides the best classification. Thus we can perform credit assignment and determine what part of the example is more important than others. This is one example; as mentioned before, this detailed knowledge can yield bias in many problem specific and independent ways. In fact, some of the previous ideas, such as ignoring spurious example parts, may be seen as an outcome of this knowledge.

CHAPTER 6: DISCUSSION

Relevant Set Detection

In this thesis, we partitioned examples to decompose a target concept. However, partitioning the examples may not be as straightforward as we considered, or perhaps we may need to partition the hypothesis space some other way. For example, we may use domain decomposition where a subset of presented examples may be relevant for a subconcept. We generalize the decomposition of the hypothesis space as the relevant set detection problem. This problem was hinted upon by Valiant in [Val84].

Relevant Set Detection Problem

Instance: A sample set $z \in X$ and a concept c.

Question: For each $z_i \in z$, how can we determine if $z_i \in c$?

This definition covers example decomposition for Boolean learning if we consider $z = x$. Then z_i becomes a bit $b_i \in x$. The relevant set detection problem is fundamental to decomposition as after this point, learning the subconcepts is usually a straightforward process. Problem specific bias would play an important role here. In the worst case scenario, we would have to solve this problem in an unsupervised manner.

Representation plays an important role in detecting relevant sets. We can say that different representations of the learning problem correspond to different hypothesis spaces. So now, the choice of hypothesis space is an important consideration in

decomposability. Some hypothesis spaces would lend inherently to decomposability, while others would serve to obfuscate the relevant sets. In natural problems, some sort of transform may be required before the lines along which to decompose become clear.

Notes on Assumptions Made in This Thesis

We introduced four special assumptions while discussing sample complexity. Some of these assumptions are reflective of real world problems, while others simply facilitate analysis and do not cause a loss of generality. An important point to note is that these assumptions spell out the class of learning problems that we have considered in this thesis.

The assumption of consistent relevance implies that a certain feature will always be found in a particular location. The assumption of total relevance facilitates analysis. If the influence is consistent, then it is allowable for some example $x \in X$ to not serve as either a positive or negative example for some subconcept. However, in natural problems complete influence would usually go with consistent influence. Usually, if there is a placeholder for an example, it would hold something meaningful.

The assumption of contiguity is a natural one. For example, in image recognition, all the pixels belonging to a feature (at least in low level features) are localized. If the elements of a subconcept are not contiguous in a real problem, then there exist either transforms, or some other hints (for example, all the elements change together) that allow for relatively easy grouping of the elements. If this is not the case, then learning becomes much harder.

The assumption of non-overlap may be done away with by a small increase in sample complexity. If overlap was allowed, then we'd have to insert two markers in the example for each subconcept. In equation 1, instead of inserting $n-1$ markers in an example, we would have to insert $2n$ markers. Then the sample complexity of sample decomposition would be

$$p_{ex}(k, \frac{1}{\varepsilon_{ex}}, \frac{1}{\delta_{ex}}) = 2n(k-1) \tag{8}$$

CHAPTER 7: CONCLUSION

This thesis presented novel research on learning by decomposition. By using results from many diverse disciplines in computer science, such as computational learning theory, Kolmogorov complexity, and statistical learning theory, we developed a general theoretical framework for learning of concepts by decomposition of examples. We studied the different steps involved in the process: decomposition, learning subconcepts, and amalgamation; and analyzed the sample complexity of each of these processes, as well as of decomposition as a whole. We developed the conditions under which decomposition is advantageous.

We translated our work in example decomposition to develop a theory of feature extraction. No general theory of feature extraction has been attempted before this. In fact, there had been little theoretical understanding of exactly what a feature is. We defined many basic concepts in feature extraction and provided a theoretical framework for the process. Doing so afforded novel insights from which we generated an algorithm for feature extraction. We also provided some constructive theorems that may provide avenues for future research.

Finally, we discussed some reasons why learning by example decomposition works as well as it does, and discussed some problems and open questions in the field.

Future Research in Learning by Example Decomposition

The work presented in this thesis is of a fundamental, theoretical nature. Being so, it offers many avenues for future research. We organize these directions under three broad categories.

- Theoretical research
- Meta theoretical research
- Applied research

Theoretical research involves a straightforward extension of this work, expanding and generalizing the theory presented in this thesis further. Meta theoretical research refers to the use of this work as a meta theory. Applied research encompasses the direct and indirect practical uses of this theory. Finally, we further our case study of feature extraction by discussion some research leads specific to that field.

Theoretical Research

We introduced some basic terminology and concepts, and used them to develop our results. However, three important questions remain immediately open, each providing a lucrative direction of research. These are:

- Understanding where to partition examples.
- Developing the relationships between concepts.
- Decomposing the domain.

Perhaps the most important question and what would be the most significant contribution of this work in the foreseeable future is understanding where to partition the examples for decomposition. An exhaustive search raises the cost of decomposition, in many cases

compromising its propitiousness. Since, in practice, example decomposition is a ubiquitous approach, we instinctively feel that natural fault lines exist for decomposition and that these 'cracks' are discoverable, perhaps cheaply. The question is, does there exist some general, theoretical way to specify how to discover these fault lines, as opposed to domain specific heuristics? The answer is, yes. We already have one way of doing so, which is introduced in chapter 4 – theorem 2. That theorem states that all elements of a feature set have similar Kolmogorov complexity, unless the elements overlap with another set. So one answer to this query is that the fault lines for decomposition tend to divide the example into chunks of differing Kolmogorov complexities. An immediate extension would then be to incorporate Kolmogorov complexity more in the main theory, and to develop this result in a general setting, independent of feature extraction. However, since Kolmogorov complexity is incomputable, we may prefer to develop the same result with some other complexity measure to provide a more usable result.

Another extension of this theory that promises yields in applied research is developing the relationships between subconcepts. In chapters 4 and 5, we provided some examples of this by defining influence, and developing the idea of mutual information between concepts in light of our theory. Future directions of research here may involve quantifying influence, and developing a theory of how it works and the different ways in which it manifests itself.

A straightforward addition to this body of research would be to study domain decomposition instead of example decomposition. Many of the results would remain the same, but doing so would make this theory easier to correspond with other, existing theories and learning algorithms, enhancing the use of this work as a meta theory.

Meta Theoretical Research

The work presented in this thesis may be considered a meta theory. The directions of research that avail themselves under this rubric are: -

- Generate new theories
- Use these theories to describe current learning algorithms and theories

In this thesis, we interpreted the theory of learning the concept decomposition to generate a theory of feature extraction. This process involves recasting the terminology and axioms in a domain specific context, and then porting over the results. Usually some additional terminology (e.g. feature sets) needs to be defined, which leads to domain specific results. This process may be repeated for other fields that lack adequate theoretical structure (as in the case of feature extraction), or do not have results that may be provided by the theory of concept decomposition.

Describing existing theories and ideas in machine learning, such as concepts embedded in an environment [Bax00], in our framework affords a deeper understanding of both those ideas and our framework. In many cases this would produce new results.

Applied Research

The work presented in this thesis leads to many venues of applied resaerch. We organize them under the following categories:

- Direct (stemming from the theory of example decomposition)
- Indirect (stemming from generated theories)

A future direction for direct applied research would be to develop an algorithm that groups subconcepts by either their approximate Levin complexity or some other measure, and then learns them keeping in mind the different factors we considered in chapter 5. As the theory develops further, more and more avenues for developing learning algorithms would open up.

An example of indirect applied research is algorithms for feature extraction. Using the theory of example decomposition, we generated the theory of feature extraction. This theory, in turn, was used to generate new results and explain existing ideas in feature extraction. Theorems 3, 4 and 5 in chapter 4 may be used to make new feature extraction algorithms. Results such as an upper limit on the number of features in a given dataset allow us to modify existing algorithms that use that limit.

Future Research in Feature Extraction

The purpose of this part of the thesis was to interpret problem decomposition for a specific domain. The structure of the target program can provide us with many more clues for feature extraction. We have developed a few in this thesis but there is much more to be done. For example, the set of relationships of an associated subroutine with

other subroutines is the context for that associated subroutine. The context of a feature may be helpful in extracting it. Existing ideas like multitask learning [Car93], and automatically learning bias in a related environment [Bax00] can be expressed in and furthered using our framework.

Another line of future research we intend to pursue is to develop the framework further, using it to analyze existing methods and develop new ones, including novel feature extraction algorithms based on the results of theorems 3, 4 and 5 in chapter 4.

GLOSSARY

The following table provides a glossary of the mathematical terms used in the thesis.

Table 1: Mathematical Terms

Symbol	Description
c_t	Target concept
X	Instance/input space
Y	Output space
C	Concept space
x	Element of X
y	Element of Y
z	Data set
D	Distribution on $Xc_t \times Y$
h	Hypothesis
$f(.)$	Function combining subconcepts
k	Size of an example
C^k	Concept class over examples of size k
C	Concept class over examples of $k > 1$
H^k	Concept class over examples of size k
H	Hypothesis class over examples of $k > 1$
L	A learning algorithm
$p(\cdot,\cdot,\cdot)$	A polynomial function over three inputs
m	Number of samples

$L(m)$	Hypothesis returned by L with input of m samples
δ	Confidence parameter
ε	Accuracy parameter
c_i	Subconcept in c_t
S_c	Smallest set of examples to learn c
n	Number of subconcepts
R_i	Relevant set
b_i	Bit in subconcept i
x_{c_i}	Substring of x influenced by c_i
D_{VC}	VC dimension for a class
$C^{[k]}$	Concept class over examples of size at most k
F	A concept class
d	VC dimension value

LIST OF REFERENCES

[Ale99] Alexander, C. "The Origins of Pattern Theory: The Future of the Theory, and the Generation of a Living World." *IEEE Software*, 16(5):71-82, 1999.

[Bax00] Baxter, J. "A Model of Inductive Bias Learning." *Journal of Artificial Intelligence Research*, 12:149-198, 2000.

[Bec92] Becker, S. "*An Information-theoretic Unsupervised Learning Algorithm for Neural Networks*." Ph.D.-thesis, Graduate Department of Computer Science, University of Toronto, 1992.

[Bel57] Bellman, R. *Dynamic Programming*. Princeton University Press, 1957.

[BK90] Brazdil, P.B., Konolige, K. *Machine Learning, Meta-Reasoning and Logics.* Kluwer Academic Publishers, Boston, 1990.

[BP93] Brunelli, R. and Poggio, T. "Face Recognition: Features Versus Templates." *IEEE Transactions. Pattern Analysis and Machine Intelligence*, 15(10):1042-1052, 1993.

[Car93] Caruana, R. A. "Multitask Learning: A Knowledge-Based Source of Inductive Bias." In *Proceedings of the 10th International Conference on Machine Learning*, 41-48. 1993.

[Cha75] Chaitin, G.J. "On the Length of Programs for Computing Finite Binary Sequences: Statistical Considerations." *Journal of the ACM*, 22: 329-340, 1975.

[Chan87] Chan, T.F. "Analysis of Preconditions for Domain Decomposition." *SIAM Journal of Numerical Analysis*, 24(2):382--390, 1987.

[CM94] Chan, T.F. and Mathew, T. P. "Domain Decomposition Algorithms." *Acta Numerica,* pages 61--143. 1994.

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

[Die00] Dietterich, T.G. "The Divide-and-Conquer Manifesto." *ALT: International Workshop on Algorithmic Learning Theory*, 2000.

[Fis87] Fisher, D.H. "Knowledge Acquisition Via Incremental Conceptual Clustering." *Machine Learning*, 2(2):139-172, 1987.

[Foe94] Foerstner, W. "A Framework for Low Level Feature Extraction." *Lecture Notes in Computer Science*, 800:383--396, 1994.

[Fur99] Fürnkranz, J. "Separate-and-Conquer Rule Learning." *Artificial Intelligence Reviews*, 13(1):3-54, 1999.

[Gol95] Goldman, J. "Pattern Theoretic Knowledge Discovery." In *6th IEEE International Conference on Tools with Artificial Intelligence*, 1995.

[Gre07] Grenander, U., and Miller, M. *Pattern Theory: From Representation to Inference.* Oxford University Press, 2007.

[GE06] Guyon, I., and Elisseeff, A. *An Introduction to Feature Extraction.* Springer Verlag, 2006.

[Hau92] Haussler, D. "Decision Theoretic Generalizations of the PAC Model for Neural Net and Other Learning Applications." *Information and Computation*, 100:78-150, 1992.

[Hau90] Haussler, D. "Probably Approximately Correct Learning." *National Conference on Artificial Intelligence*, 1101-1108, 1990.

[HS03] Hlavatý, T., and Skala, V. "A Survey of Methods for 3D Model Feature Extraction." *Bulletin of IV. Seminar 'Geometry and Graphics in Teaching Contemporary Engineer' Szczyrk, Poland*, No: 13/03, 5-8, 2003.

[HKCWL03] Hong-Dun, L., Kang-Ping, L., Chung, B.T., Wu, L.C., and Liu, R.S. "Thermal Theory Based Feature Extraction Method for High Noise PET Images." *Nuclear Science Symposium Conference Record*, 5:3110-3114, 2003.

[Hun70] Huntley,C.E. *The Divine Proportion: A Study in Mathematical Beauty.* Dover publications. 1970.

[JKF03] Joshi, S., Kursun, O., and Favorov, O. V. "Exploiting the Structure of Order: An Application to Natural Images." *7th World Multiconference on Systemics, Cybernetics and Informatics (SCI)* . 2003.

[Lev74] Levin,L.A. "Laws of Information (Nongrowth) and Aspects of the Foundation of Probability Theory." *Problems of Information Transmission*, 10 (3): 206-210. 1974.

[LV97] Li,M. and Vitanyi,P. *An Introduction to Kolmogorov Complexity and its Applications*. Springer, New York, 1997.

[LLS02] Lian-Yin,Z., Li-Pheng,K., and Sai-Cheong,F. "Feature Extraction Using Rough Set Theory and Genetic Algorithms--an Application for the Simplification of Product Quality Evaluation." Computers & Industrial Engineering, 43 (4): 661-676, 2002.

[Low99] Lowe, D. G. "Object Recognition from Local Scale-invariant Features." In *Proceedings of International Conference on Computer Vision* , 1150-1157. 1999.

[Mau05] Maurer, A. “Algorithmic Stability and Meta-Learning.” *Journal of Machine Learning Research,* 6:967-994, 2005.

[Mic69] Michalski, R. S. “On the Quasi-minimal Solution of the General Covering Problem.” In *Proceedings Fifth International Symposium on Information Processing, FCIP 69, Bled, Yugoslavia*, 125-128, 1969.

[MS84] Michalski, R. S., and Stepp, R. E. “Learning from Observation: Conceptual Clustering.” *Machine Learning: An Artificial Intelligence Approach*, 331-363. Springer, Berlin, Heidelberg, 1984.

[MM05] Mierswa, I. and Morik, K. “Automatic Feature Extraction for Classifying Audio Data.” *Machine Learning*, 58(2-3):127-149, 2005.

[Mum96] Mumford, D. “Pattern Theory: a Unifying Perspective.” In *Perception as Bayesian Inference*, 25-62, 1996.

[Nat91] Natarajan, B.K. *Machine Learning: A Theoretical Approach.* Morgan Kauffman, San Mateo, CA, 1991.

[NA02] Nixon, M.S., and Aguado, A.S. *Feature Extraction and Image Processing*. Newnes, Oxford, 2002.

[Now77] Nowak, A. “On a General Dynamic Programming Problem.” *Colloquium Mathematicum.*, 37(1):131--138, 1977.

[Qua92] Quarteroni, A. “Domain Decomposition Methods,” In *AMS. Proceedings of the Sixth International Symposium on Domain Decomposition Methods. Como, Italy*, 1992.

[Rom04] Romesburg, C. *Cluster Analysis for Researchers*. Krieger Pub. Co, Malabar, Fla, 2004.

[RNGG94] Ross, T., Noviskey, M., Gadd, D., and Goldman, J. "Pattern Theoretic Feature Extraction and Constructive Induction." in *Proceedings. ML-COLT '94 Workshop on Constructive Induction and Change of Representation,* 1994.

[Sch97] Schmidhuber,J. "Discovering Solutions with Low Kolmogorov Complexity and High Generalization Capability." *Neural Networks*, 10 (5): 857-873, 1997.

[SM86] Stepp, R. E., and Michalski, R. S. *Conceptual Clustering: Inventing Goal Oriented Classifications of Structured Objects*. Morgan Kaufmann, Los Altos, CA, 1986.

[TB01] Talavera, L., and Bjar, J. "Generality-Based Conceptual Clustering with Probabilistic Concepts." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(2):196--206, 2001.

[Utg86] Utgoff, P. E. *Machine Learning of Inductive Bias*. Kluwer Academic, 1986.

[Val84] Valiant, L. G. "A Theory of the Learnable." *Comm. ACM*, 27:1134–-1142, 1984.

[Vap82] Vapnik, V. N. *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, New York, 1982.

[Vap98] Vapnik, V.N. *Statistical Learning Theory*. John Wiley & Sons, New York, 1998.

[ZKF02] Zhai, L.Y., Khoo, L.P. and Fok, S.C. "Feature Extraction Using Rough Set Theory and Genetic Algorithms--an Application for the Simplification of Product Quality Evaluation." *Computers & Industrial Engineering*, 43(4):661-676, 2002.