

Learning Classification Trees From Distributed Horizontally and Vertically Fragmented Data Sets

Tarkeshwari Sharma, Adrian Silvescu, and Vasant Honavar
Artificial Intelligence Research Laboratory
Department of Computer Science
Iowa State University, Ames, IA 50011
{taru|silvescu|honavar}@cs.iastate.edu

Abstract

¹ Recent advances in data storage and acquisition technologies have made it possible to produce increasingly large data repositories. Most of these data sources are physically distributed and assembling them together at a central site is expensive in terms of network bandwidth and insecure. Hence there is a need for Learning Algorithms that are able to learn from distributed data without collecting it in a central location. We present provably exact algorithms for learning decision trees from distributed data sets. We prove that the results obtained in this case are the same as those obtained if the data were stored at a central location. We also give a time, space and communication cost analysis.

We conclude with a discussion of a general technique for adapting some of the existing learning algorithms to learn from distributed datasets.

¹This research was supported in part from grants from the National Science Foundation (NSF 9982341), Department of Defense, the John Deere Foundation, the Carver Foundation, and Pioneer Hi-Bred Inc.)

1 Introduction

Recent advances in sensor, high throughput data acquisition, and digital information storage technologies have made it possible to acquire, store, and process large volumes of data in digital form in a number of domains. For example, biologists are generating gigabytes of genome and protein sequence data at steadily increasing rates. Organizations have begun to capture and store a variety of data about various aspects of their operations (e.g., products, customers, and transactions). Complex distributed systems (e.g., computer systems, communication networks, power systems) are equipped with sensors and measurement devices that gather and store, a variety of data that is useful in monitoring, controlling, and improving the operation of such systems. Many application domains, (e.g., military command and control, law enforcement etc) require the use of multiple, geographically distributed, heterogeneous data and knowledge sources (Honavar, Miller & Wong, 1998; Yang, Havaldar, Honavar et al., 1998). Translating the recent advances in our ability to gather, process, and store data at increasing rates into fundamental gains in scientific understanding (e.g., characterization of macromolecular structure-function relationships in biology) and organizational decision making presents several challenges in computer and information sciences in general and machine learning, data mining, and knowledge discovery in particular.

Data repositories of interest in many applications are very large. Many of the existing mining algorithms do not scale up to extremely large data sets. One approach to this problem is to partition the data set into several subsets of manageable size, learn from each resulting dataset, and somehow combine the resulting hypotheses. In other applications (e.g., collaborative scientific discovery) in addition to being large, repositories are autonomous and physically distributed. Thus it is desirable to perform as much analysis as possible at the sites where the datasets are located (e.g., using mobile software agents that transport themselves to the data repositories, or stationary software agents that reside at the repositories), and return only results of analysis in order to conserve network bandwidth. The sheer volume and the rate of accumulation of the data, often prohibits the use of *batch learning* algorithms which would require processing the entire data set whenever new data is added to the data repository. It is also possible that organizations are not willing to provide access to raw data for various security and privacy

reasons but they allow access to some kind of summary or statistics of data, e.g., count of people having a particular disease as opposed to revealing list of names of people.

A key problem in acquiring useful knowledge from large, dynamic, distributed data sources is that of devising *distributed learning* algorithms that can incorporate data across locations.

In this paper we present algorithms for learning decision trees from distributed data sets. The resulting algorithms that yield the same hypothesis from a collection of distributed data sets as that obtained from the complete data set. We also provide upper bounds on the time, space, and communication complexity of the proposed algorithms. We conclude with a discussion of a general technique for adapting some of the existing algorithms to work in a distributed setting.

The rest of the paper is organized as follows: In section 2 we give a general idea of the problem and various ways of fragmenting the data. . In section 3 we illustrate our technique by giving a solution to this problem, for horizontally and vertically fragmented data, using decision trees. Section 4 concludes with a summary of the paper, a brief discussion of related work, and a brief outline of ongoing and future research.

2 The Distributed Learning Problem

2.1 Distributed Learning

The *distributed learning* problem can be summarized as follows: The data is distributed among different sites and the learner's task is to discover some useful knowledge (e.g., a hypothesis in the form of a decision tree that classifies instances). This can be accomplished by a learning agent that visits the different sites to gather the information it needs to generate a suitable hypothesis from the data. This corresponds to the *serial distributed learning* scenario shown in figure 2. Alternatively, the different sites can transmit the necessary information to the learning agent situated at a central location as in the parallel distributed learning shown in figure 1. We assume that it is not feasible in the distributed learning setting to transmit raw data from different sites. Consequently, the learner has to rely on the information extracted from the sites. Thus, identification the information required

by different learning algorithms, and design of efficient means for providing such information to the learner are central questions that need to be addressed in devising distributed learning algorithms.

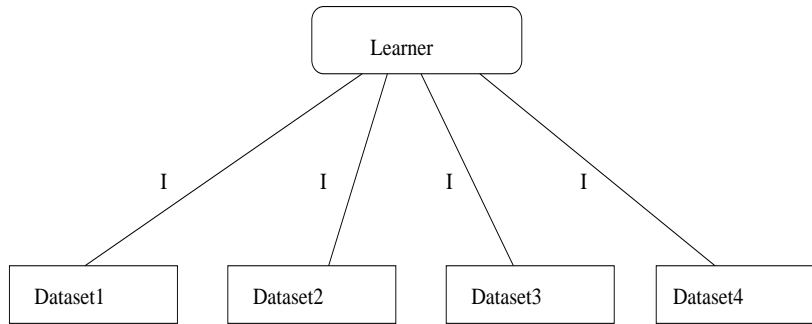


Figure 1: Parallel Distributed Learning

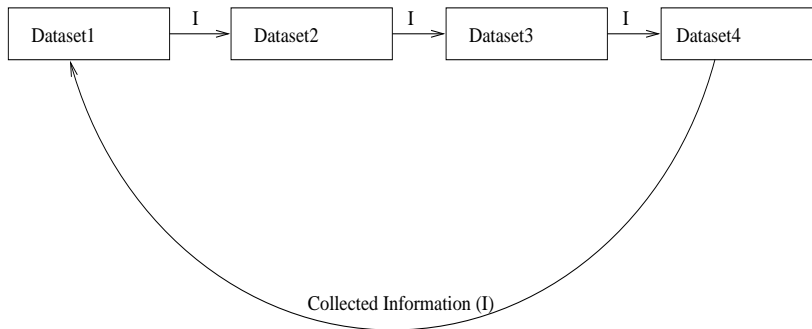


Figure 2: Serial Distributed Learning

A *Distributed Learning* algorithm $L_{Distributed}$ is said to be *exact* with respect to the hypothesis inferred by a learning algorithm L if the hypothesis produced by $L_{Distributed}$ using distributed data sets $D_1 \cdots D_n$ stored at sites $1 \cdots n$ (respectively), is the *same* as that obtained by L from the complete data set D . That is, $L_{Distributed}$ is an exact distributed learning algorithm with respect to the hypothesis inferred by a learning algorithm L if it is the case that:

$$L_{distributed}(D_1, D_2, \dots, D_n) = L(D_1 \bar{\cup} D_2 \bar{\cup} \dots \bar{\cup} D_n)$$

where $\bar{\cup}$ denotes multiset union. For example, the multiset union of the multisets $\{a b a\}$ and $\{b c a d\}$ is $\{a a a b b c d\}$.

Similarly, we can define exact distributed learning with respect to other criteria of interest (e.g., expected accuracy of the learned hypothesis). More generally, it might be useful to consider *approximate* distributed learning in similar settings. However, we restrict the discussion that follows to an approach to exact distributed learning using *information extraction*.

This approach to exact distributed learning involves extracting from distributed datasets, the information necessary for inferring the appropriate hypothesis. We introduce the information extraction operator $I(D_i)$ that extracts from each data set D_i , the information necessary for $L_{Distributed}$ to learn from $D_1 \cdots D_n$. If the information extracted from the distributed datasets is same as that used by L to infer a hypothesis from the complete dataset D (that is, $C[I(D_1), I(D_2), \dots, I(D_n)] = I(D)$), $L_{Distributed}$ will be exact with respect to L .

2.2 Horizontally and Vertically Fragmented Distributed Data Sets

A data set D is distributed among sites $1 \cdots n$ containing data set fragments $D_1 \cdots D_n$. The fragments can correspond to *horizontal*, *vertical*, or both *horizontal* and *vertical* fragmentation of D . We assume that the individual data sets $D_1 \cdots D_n$ altogether contain enough information to generate the complete dataset D . However, it might be the case that the individual data sets are autonomously owned and maintained. Consequently, the access to the raw data may be limited and only summaries of the data (e.g., number of instances that match some criteria of interest) may be made available to the learner. Furthermore, given the sheer size of the data sets, it may not be feasible to explicitly construct the complete data set D at a central location.

- **Horizontal Fragmentation:** In this case, the data is distributed in such a manner that each site contains a multiset of tuples. The (multiset) union of all these multisets constitute the complete dataset. If the multisets of tuples of data are indicated by D_1, D_2, \dots, D_n , each site

contains one or more of these sets. Let D denote the complete data set, then *Horizontally Distributed Data* (HDD) has the property:

$$D_1 \cup D_2 \dots \cup D_n = D$$

A complete data set containing student data is shown in table1. Horizontal fragments of this dataset are shown in table2 and table3.

Student Id	Name	Department	GPA
1121	Taru Trivedi	Computer Science	3.5
1122	Adrian Silvescu	Animal Science	3.7
1123	John Silver	Animal Science	3.8
1124	Doina Caragea	Computer Science	3.6

Table 1: Student Dataset (Complete)

Student Id	Name	Department	GPA
1121	Taru Trivedi	Computer Science	3.5
1122	Adrian Silvescu	Animal Science	3.7

Table 2: Student Dataset (Horizontal Fragment I)

Student Id	Name	Department	GPA
1123	John Silver	Animal Science	3.8
1124	Doina Caragea	Computer Science	3.6

Table 3: Student Dataset (Horizontal Fragment II)

- **Vertical Fragmentation:** In this case, each data tuple is fragmented into several subtuples each of which shares a unique key or index. Thus, different sites store *vertical*, possibly overlapping, fragments of the data set. Each fragment corresponds to a subset of the attributes that describe the complete data set.

Let A_1, A_2, \dots, A_n indicate the set of attributes whose values are stored at sites $1 \dots n$ and A denote the set of attributes that are used to

describe the data tuples of the complete data set. Then *Vertically Distributed Data* (VDD) has the property:

$$A_1 \cup A_2 \dots \cup A_n = A$$

Let D_1, D_2, \dots, D_n respectively denote the fragments of the dataset stored at sites $1 \dots n$ and let D denote the complete data set. Let the i th tuple in a data fragment D_j be denoted as $t_{D_j}^i$. Let $t_{D_j}^i.\text{unique_index}$ denote the unique index associated with tuple $t_{D_j}^i$. The VDD has the property:

$$D_1 \times D_2 \times \dots \times D_n = D$$

$$\forall D_j, D_k, t_{D_j}^i.\text{unique_index} = t_{D_k}^i.\text{unique_index}$$

Thus, the subtuples from the vertical data fragments stored at different sites can be put together using their unique index to form the corresponding data tuples of the complete dataset. It is possible to envision scenarios in which a vertically fragmented data set might lack unique indices. In such a case, it will be necessary to combinations of attribute values to infer associations among tuples. In what follows, we will assume the existence of unique indices in vertically fragmented distributed data sets.

Vertical fragments of the student dataset shown in table1 are shown in table4 and table5. In this case, the student id is used as the unique index associated with subtuples.

Student Id	Name	Department
1121	Taru Trivedi	Computer Science
1122	Adrian Silvescu	Animal Science
1123	John Silver	Animal Science
1124	Doina Caragea	Computer Science

Table 4: Student Dataset (Vertical Fragment I)

Similarly, we can envision distributed data sets that are both horizontally and vertically fragmented.

Student Id	Department	GPA
1121	Computer Science	3.5
1122	Animal Science	3.7
1123	Animal Science	3.8
1124	Computer Science	3.6

Table 5: Student Dataset (Vertical Fragment II)

3 Distributed Learning using Decision Trees

This section discusses the decision tree learning algorithm (Quinlan, 1986) and shows how it can be adapted to work with horizontally and vertically distributed data sets.

3.1 Decision Tree Learning

Decision tree learning is a method for approximating discrete valued target functions from labeled examples, where the learned function is represented by a decision tree. The ID3 (Iterative Dichotomizer 3) algorithm proposed by Quinlan (Quinlan, 1986) and its variants offer a simple, and practically rather effective approach to inferring decision trees from labeled examples.

Consider a set of instances S which is partitioned into M disjoint subsets (classes) C_1, C_2, \dots, C_M such that

- $S = \bigcup_{i=1}^M C_i$
- $C_i \cap C_j = \emptyset \ \forall i \neq j$

The probability of a randomly chosen instance $s \in S$ belonging to class C_j is $\frac{|C_j|}{|S|}$, where $|X|$ denotes the cardinality of the set X . So, the *information content* of the knowledge of membership of a randomly chosen instance in class C_j is $-\log_2 \left(\frac{|C_j|}{|S|} \right)$ bits. The expected information content of knowledge of class membership of a random instance $s \in S$ is

$$-\sum_j \frac{|C_j|}{|S|} \cdot \log_2 \left(\frac{|C_j|}{|S|} \right)$$

This quantity, which is also known as the *entropy* of set S and measures the expected information needed to identify the class membership of instances in S . The decision tree learning algorithm searches in a greedy fashion, for attributes that yield the maximum amount of information for determining the class membership of instances in a training given training set S of labeled instances. The result is a decision tree that correctly assigns each instances in S to its respective class. The construction of the decision tree is accomplished by recursively partitioning S into subsets based on values of the chosen attribute until each resulting subset has instances that belong to exactly one of the M classes. The selection of attribute at each stage of construction of the decision tree maximizes the estimated expected information gained from knowing the value of the attribute in question.

The *information gain* for an attribute a , relative to a collection of instances S is defined as:

$$Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where $Values(A)$ is the set of all possible values for attribute A , and S_v is the subset of S for which attribute a has value v .

The information gain associated with an attribute can be calculated using the count of examples from different classes that have specific values for the attribute in question.

Suppose we want to partition the set of instances at a particular node in a partially constructed decision tree. Let a_j denote the attribute at the j th node (starting from the root node) leading up to the node in question. Let $v(a_j)$ denote the value of the attribute a_j attribute along the path leading up to the node in question.

For adding a node below any branch of tree, the set of examples being considered satisfy the constraints on values of attributes specified by:

$$L_l = [a_1 = v(a_1), a_2 = v(a_2), \dots, a_l = v(a_l)]$$

where l is the depth of the node in question.

We need to obtain the relevant counts from the set of examples that satisfy this constraint. This calculation has to be performed once for each node that is added to the tree starting with the root node.

Using the following two theorems, we prove that count of examples collected from distributed data sets is same as that which would be collected from the complete data set. This suffices to prove that the decision tree constructed from a given data set in the two scenarios is exactly same.

When data is horizontally distributed, examples for a particular value of a particular attribute are scattered at different locations. For finding the count of examples for a particular node in tree, all the sites are visited and count is accumulated. The learner uses this count to find the best attribute to further partition the set of examples being considered. We use f_l to denote the count of examples that satisfy the constraints specified by L_l and f_l^i denotes the count of examples that satisfy the constraints given by L_l at site i . Algorithm for computing the counts in case of HDD is shown in fig 3:

Algorithm ComputeCountHDD

Input: L_l (attributes and their values along the path leading upto the node under which a subtree is being added)

Output: count of examples across all sites that satisfy the constraints specified by L_l .

```

    f := 0
    for i = 1 to m //there are m sites
        f := f + f_l^i
    return f
end

```

Figure 3: Algorithm Compute Count for HDD.

Theorem 1 For every input $L_l = [a_1 = v(a_1), a_2 = v(a_2), \dots, a_l = v(a_l)]$, $ComputeCountHDD(l)$ computes f_l .

Proof:

$$f = \sum_{i=1}^m f_l^i$$

= number of examples that satisfy constraints given by L_l in $(D_1 \bar{\cup} D_2 \bar{\cup} \dots \bar{\cup} D_m)$

// by definition of HDD

$$= f_l$$

□

In this algorithm, for each node in tree, f_l is calculated for a set of attributes and their values.

Time Complexity: The time required to calculate f_l^i at each site i , is proportional to $|D_i|$. Hence, the time required for calculating f_l is therefore proportional to $|D_1 \cup D_2 \cup \dots \cup D_m| = |D|$. Let A denote set of all attributes and V denote the largest set of values that an attribute can take. For each node, the maximum time required to calculate f_l is bounded by $|A| |V| |D|$. Therefore, the run time of the distributed decision tree learning algorithm for horizontally distributed data sets is proportional to the product of the time required per node and the number of nodes in the tree. Let $treesize(D)$ denote the number of nodes in the decision tree constructed from the dataset D . Then the run time of the algorithm is bounded by nodes and is $|A| |V| |D| treesize(D)$.

Space Complexity: The information gathered to facilitate learning is simply the counts of examples for each class along each outgoing branch of the node in question. A loose upper bound for this is given by $M |A| |V|$ where M is the number of classes.

Communication cost: The upper bound for information carried from one location to another is $M |A| |V|$ for one node in the tree. Hence, the communication cost involved in constructing the complete tree is $M |A| |V| treesize(D)$.

In vertically distributed datasets, we assume that each example has a unique indices associated with it. Subtuples of an example are distributed across different sites. However, they can be related to each other using their unique index. In order to select the attribute to partition the instances at a node in a partially constructed tree, the relevant counts are gathered using the unique indices. To find the best attribute, a pass is made through all the data sites to compute the count of examples. As before, let $L_l = [a_1 = v(a_1), a_2 = v(a_2), \dots, a_l = v(a_l)]$ denote constraints on the values of attributes satisfied by the instances at the tip of the node in question. Let $I_{L_{l-1}}$ denote the set of indices for tuples satisfying L_{l-1} . We use f_{L_l} to denote the count of examples that satisfy the constraints given by L_l among the set of tuples with indices in $I_{L_{l-1}}$.

Algorithm for computing the counts in case of VDD is shown in fig 4.

<p>Algorithm ComputeCountVDD</p> <p>Inputs: L_l (attributes and their values along the path leading upto the node under which a subtree is being added), $I_{L_{l-1}}$ (the set of indices of instances that satisfy the constraint L_{l-1}).</p> <p>Output: count of examples whose indices are in $I_{L_{l-1}}$ and satisfy the constraints specified by L_l.</p> <p>begin</p> <p style="padding-left: 2em;">visit the site that has subtuples that has values for the attribute a_l</p> <p style="padding-left: 2em;">Compute I_{L_l} by selecting tuples which satisfy the constraint L_l and whose indices appear in $I_{L_{l-1}}$</p> <p style="padding-left: 2em;">return I_{L_l}</p> <p>end</p>

Figure 4: Algorithm Compute Count for VDD.

Theorem 2 *For every input specified by L_l and $I_{L_{l-1}}$, $ComputeCountVDD(L_l, I_{L_{l-1}})$ computes f_{L_l} .*

Proof:

Provided that I_L is correctly computed, then $| I_{L_l} | = f_{L_l}$

We prove this by induction. Base case:

I_{L_0} = the set of all indices.

Induction step:

I_{L_l} = The set of indices in $I_{L_{l-1}}$ that satisfy $a_l = v(a_l)$ □

Suppose we denote the indices for the examples at the n^{th} node (assuming left to right numbering of nodes) at depth k in the tree is denoted by I_{kn} . Let A denote set of all attributes, V denote the largest set of values that an attribute can hold and let h denote the depth of the tree. Since each node in the tree corresponds to a unique combination of attribute values, indices of examples at one node cannot appear in another node at the same depth in the tree. That is,

$$\forall k (\forall i \neq j I_{ki} \cap I_{kj} = \phi)$$

Time Complexity: The time required for partitioning the examples at the n th node at depth k is

$$|A| |V| |I_{kn}|$$

Therefore, the time required for partitioning examples at all of the nodes at depth k in the tree bounded by

$$\begin{aligned} & \sum_{n \in \text{Level } k} |A| |V| |I_{kn}| \\ &= |A| |V| \sum_n |I_{kn}| \\ &\leq |A| |V| |D| \end{aligned}$$

for all levels in tree,

$$\begin{aligned} &\leq \sum_{k \in \text{levels}} |A| |V| |D| \\ &\leq |A| |V| |D| h \end{aligned}$$

Space Complexity: Information gathered to facilitate learning is simply the counts of examples for each class along each outgoing branch of the node in question along with the relevant indices. Let *indexsize* denote the space required to store an index. A loose upper bound for this is given by $M |A| |V| + |D| \textit{indexsize}$ where M is the number of classes and $|D| \textit{indexsize}$ provides an upper bound on the number of indices.

Communication Complexity: The upper bound for the information carried from one site to another is $M |A| |V| + |D| \textit{indexsize}$ for one node in the tree. Hence, the communication cost involved in constructing the complete tree is $(M |A| |V| + |D| \textit{indexsize}) \textit{treesize}$.

The bounds given above are loose worst case upper bounds. They are based on naive implementation of the proposed algorithms to make the key ideas and the complexity analysis transparent to the reader. In practice, the time, space, and communication requirements can be less than those indicated by the bounds. For example, while implementing the HDD algorithm, we collect the counts for all attributes and their values at each site.

This avoids the need to visit a site once separately for each attribute and its value. Hence, the overall time required for calculating f for each node is bounded by $|D|$. Therefore, the runtime of the algorithm is bounded by $|D| \text{treesize}(D)$. Similarly, in the case of vertically distributed data, we find the local best attribute at a site and transmit that information. This avoids the need to visit a site for counts corresponding to every attribute and hence, the run time is bounded by $|D| h$. Since the information carried is only the information gain of the local best attribute, the space required reduces to $|D| \text{indexsize}$. The communication cost also reduces to $(|D| \text{indexsize}) \text{treesize}$.

4 Summary and Discussion

The design and analysis of efficient algorithms for learning from distributed data sets is one of the key challenges which needs to be addressed in order for us to be able to translate recent advances in our ability to gather and store large volumes of data into deeper understanding of the respective domains.

Most distributed and incremental learning algorithms that have been proposed in the literature (Davies, & Edwards, 1998; Domingos, 1997; Prodromidis & Chan, 1999; Provost, & Hennessy, 1996) have the disadvantage that the learning is not exact. Furthermore, most of them do not guarantee generalization accuracies that are provably close to those obtainable in the centralized setting. At present, with the exception of some interesting results (e.g., *mistake bounds*) for the closely related problem of *online learning* (Littlestone, 1988; Littlestone, 1994; Vovk, 1990; Blum, 1996; Bottou, 1998), a characterization of hypothesis classes that admit efficient and scalable algorithms for exact or approximate distributed learning is lacking. Yet from a practical standpoint, the design and implementation of such algorithms is clearly of interest.

An approach to adapting decision tree learning algorithms to work with distributed databases was explored in (Bhatnagar & Srinivasan, 1997). The scenario that they address can be viewed as a variant of the vertical fragmentation of data discussed in this paper. However, since their approach is motivated by somewhat different considerations, it is focused on the problem of obtaining counts from *implicit tuples*. In particular, they do not assume the existence of a unique index for each tuple in the complete data set that

can be used to associate the subtuples of the tuple. The resulting algorithm simulates the effect of *join* operation on the sites without enumerating the tuples.

The distributed decision tree learning algorithms discussed in this paper are designed to deal with horizontally fragmented or vertically fragmented distributed data sets. They operate by decomposing the learning task into an information extraction phase and a hypothesis generation phase. This provides a general approach to designing *provably exact* distributed learning algorithms. The approach to learning from vertically fragmented distributed data sets assumes the existence of a unique index for each tuple in the complete data set. This assumption significantly simplifies the theoretical analysis of the resulting algorithm. Furthermore, in many distributed learning scenarios that arise in practice, it is reasonable to make such an assumption. For example, in scientific datasets generated by a number of collaborating laboratories investigating molecular structure-function relationships, each sample is identified by a unique identifier. However, it is possible to relax this assumption to modify the algorithm to deal with distributed learning scenarios wherein such an assumption may not hold.

The algorithms for distributed learning discussed in this paper assume either horizontal fragmentation or vertical fragmentation of data, but not both. However, it is relatively straightforward to devise distributed learning algorithms for data sets that exhibit both horizontal and vertical fragmentation.

In this paper, we presented adaptations of decision tree learning algorithms that can deal with horizontally and vertically distributed datasets without collecting all of the data at a central location. The approach used to devise distributed versions of the algorithm consists of two key components: identification of the information requirements of the learner; design of efficient means of providing the necessary information to the learner in the distributed setting. This decomposition of the learning task into *information gathering* and *hypothesis generation* phases offers a general approach to adapting some of the existing learning algorithms to work in the distributed setting (see figure 5). The *hypothesis generation* component of the of the algorithm can be thought of as the *control* part of the algorithm, which triggers the execution of the *information gathering* part as needed. The execution of the two parts is typically interleaved in time. In this model of distributed learning, only the *information gathering* component has to effectively cope

with the distributed nature of the data.

In the case of the decision tree learning algorithm, the information required by the learner is in the form of counts of examples that belonging to each class and satisfying certain constraints with respect to the values that they have for different attributes. This information is sufficient for the *hypothesis generation* component of the algorithm to produce the appropriate hypothesis in the form of a decision tree that correctly classifies all the examples in the distributed data set.

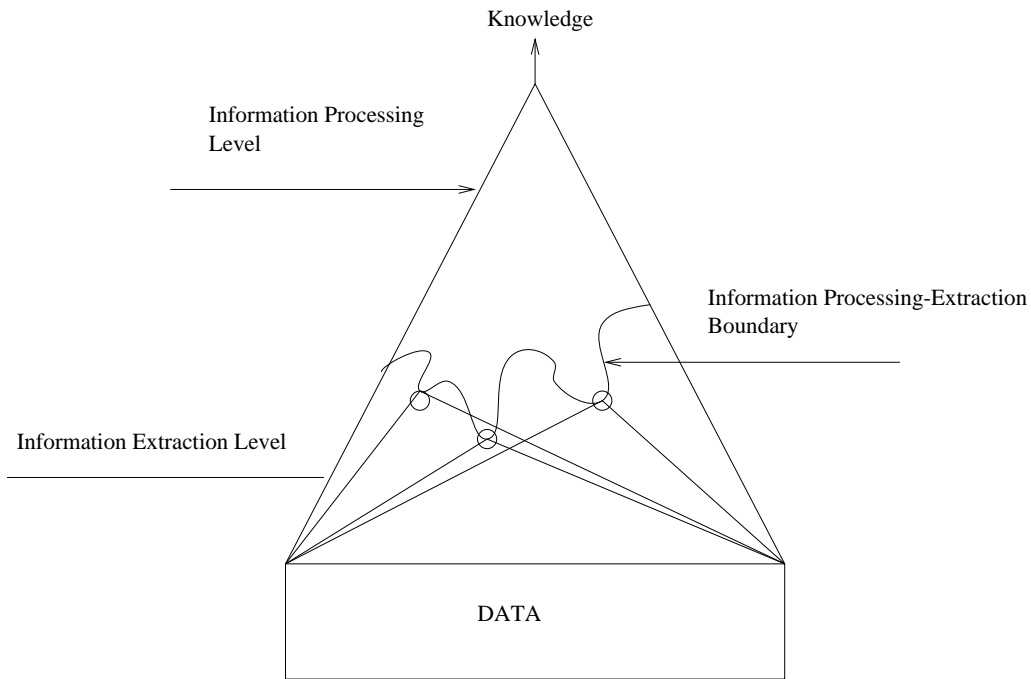


Figure 5: Decomposition of the Learning task into information extraction and learning components. The hypothesis generation component calls on the information extraction component as needed.

In this model of distributed learning, the *boundary* that defines the division of labor between the information gathering and hypothesis generation components can be set at various levels. At one extreme, if no information gathering is performed, the hypothesis generation component needs to access the raw data. An example of this scenario is provided by distributed instance based learning of k nearest neighbor classifiers in the case of hori-

zonally fragmented data set. Here, the data set fragments are simply stored at the different sites. Classification of a new instance is performed by the hypothesis generation component which computes the k nearest neighbors of the instance to be classified (based on some specified distance metric) by visiting the different sites. The classification assigned to the instance is the same as the majority class among the k nearest neighbors of the instance.

At the other extreme, if the boundary between information gathering and hypothesis generation is set at the highest level, the task of the hypothesis generation component is trivial since the information gathered by the information gathering component directly yields the desired hypothesis. It is easy to show that this approach yields effective algorithms for learning purely conjunctive hypotheses in the distributed setting.

Given the relatively large number of learning algorithms that have been developed over the past decades, and the large body of theoretical and empirical results associated with existing algorithms, the proposed approach to devising distributed learning algorithms merits further investigation.

Work in progress is aimed at the elucidation of the necessary and sufficient conditions that guarantee the existence of exact or approximate cumulative learning algorithms in general and different types of incremental and distributed learning algorithms in particular in terms of the properties of data and hypothesis representations and information extraction and learning operators; characterization of information requirements for distributed and incremental learning under various assumptions; investigation of optimum division of labor between the information gathering and hypothesis generation components of the algorithm under different assumptions. design of theoretically well-founded algorithms for incremental and distributed learning; and application of such algorithms to large-scale data-driven knowledge discovery tasks in applications such as bioinformatics and computational biology.

References

Honavar, V., Miller, L., & Wong, J. (1998). *Distributed Knowledge Networks*, In: Proceedings of the IEEE Conference on Information Technology, Syracuse, NY.

Yang, J., Havaldar, R., Honavar, V., Miller, L. & Wong, J. (1998). *Coordina-*

tion of Distributed Knowledge Networks Using Contract Net Protocol, IEEE Information Technology Conference Syracuse, NY.

Quinlan, J. R. (1986). *Induction of Decision Trees*, Machine Learning, vol 1, pp 81-106.

Domingos, P. (1997). *Knowledge Acquisition from Examples Via Multiple Models*, In: Proceedings of the Fourteenth International Conference on Machine Learning, Nashville, TN.

Prodromidis, A.L., and Chan, P.K. (1999). *Meta-learning in distributed data mining systems: Issues and Approaches*, Book on Advances of Distributed Data Mining, editors Hillol Kargupta and Philip Chan, AAAI press (under review).

Provost, F., and Hennessey, D. (1996). *Scaling Up: Distributed Machine Learning with Cooperation*. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence.

Littlestone, N. (1988). *Learning when irrelevant attributes abound*, Machine Learning, 2:285-318.

Littlestone, N. (1994). *The weighted majority algorithm*, Information and Computation, 108:212-261.

Vovk, V. (1990). *Aggregating Strategies*. In: Proceedings of the Third Annual Workshop on Computational Learning Theory.

Blum, A. (1996). *On-line Algorithms in Machine Learning (a survey)*. In: Dagstuhl Workshop on On-line Algorithms, Dagstuhl, Germany.

Davies, W., & Edwards, P. (1998). *DAGGER: A New Approach to Combining Multiple Models Learned from Disjoint Subsets*, ML99

Bottou, L. (1998). *Online Learning and Stochastic Approximations*, Online Learning and Neural Networks, David Saad editor, Cambridge University Press, Cambridge, UK.

Bhatnagar, R. & Srinivasan S. (1997). *Pattern Discovery in Distributed Databases*, AAAi97, pp 503-508.