Analysis and Synthesis of Agents that Learn from Distributed Dynamic Data Sources

Doina Caragea, Adrian Silvescu, and Vasant Honavar

Artificial Intelligence Research Laboratory, Department of Computer Science, Iowa State University, Ames, Iowa 50011-1040, USA,
honavar@cs.iastate.edu,
WWW home page: http://www.cs.iastate.edu/~honavar/aigroup.html

Abstract. We propose a theoretical framework for specification and analysis of a class of learning problems that arise in open-ended environments that contain multiple, distributed, dynamic data and knowledge sources. We introduce a family of learning operators for precise specification of some existing solutions and to facilitate the design and analysis of new algorithms for this class of problems. We state some properties of instance and hypothesis representations, and learning operators that make exact learning possible in some settings. We also explore some relationships between models of learning using different subsets of the proposed operators under certain assumptions.

1 Learning from Distributed Dynamic Data

Many practical knowledge discovery tasks (e.g., learning the behavior of complex computer systems from observations, computer-aided scientific discovery in bioinformatics) present several new challenges in machine learning. The data repositories in such applications tend to be very large, physically distributed, often autonomously managed, and constantly growing over time (as new data get added). Thus, there is a need for algorithms for learning from distributed data by analysing the distributed data sets where they reside instead of shipping large volumes of data across networks, in an incremental fashion, as the data becomes available over time, without having to reprocess the already processed data [5, 14].

Although some incremental and distributed learning algorithms have been proposed in the literature, most of them [9, 4, 15], do not guarantee generalization accuracies that are provably close to those obtainable in the batch or centralized learning scenario. Some notable exceptions include parallel and distributed versions [1, 13, 6, 12] and incremental versions [3] of batch algorithms that preserve the underlying nature of the centralized algorithm. At present, with the exception of some interesting results (e.g., *mistake bounds*) for the closely related problem of *online learning* [7], a characterization of hypothesis classes that admit efficient exact or approximate distributed or incremental learning is lacking. Yet from a practical standpoint, the design and implementation of such learning agents is clearly of interest. Against this background, there is a need to address incremental and distributed learning problems in their full generality.

This paper presents some tentative steps towards a framework for specification, analysis, and synthesis of incremental and distributed learning agents. We define some learning and information extraction operators to formally model some existing learning algorithms. We explore some properties of instance and hypothesis representations, and learning operators that guarantee the existence of incremental and distributed learning algorithms with provable performance guarantees relative to their batch or centralized counterparts. We offer some examples to illustrate the use of this theoretical framework in designing new incremental and distributed learning algorithms.

2 Incremental Learning and Distributed Learning

A generic incremental learning scenario is shown in Fig. 1. In an *incremental*

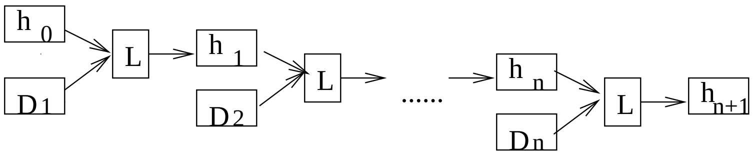

Fig. 1. Incremental Learning

learning scenario, data sets $D_1, D_2, \cdots, D_n$ are assumed to become available to the learner at discrete instants in time $t_1, t_2, \cdots, t_n$. The learner starts with a (possibly null) initial hypothesis h_0 which constitutes the prior knowledge of the domain. We assume that the learner is typically unable to store the data in its raw form. Thus, it can only maintain and update its hypothesis base as new data becomes available. Thus, h_0 gets updated to h_1 on the basis of D_1, and h_1 gets updated to h_2 on the basis of data D_2, and so on.

In a *distributed learning* scenario, the data set is assumed to be physically distributed across multiple, possibly autonomous, data repositories $D_1, \cdots, D_n$. The learner can visit the repositories to gather the information necessary for generating knowledge (e.g., in the form of pattern classification rules) by processing the data where it is stored. Alternatively, the data repositories may transmit the information to the learner. In either case, we prohibit transport of raw data among different sites. A distributed learning scenario is shown in Fig. 2.

A number of variations on these basic incremental and distributed learning scenarios can be envisioned under different assumptions concerning where and when data processing is performed, what information is made available to the learner, etc. More generally, we can consider incremental learning from

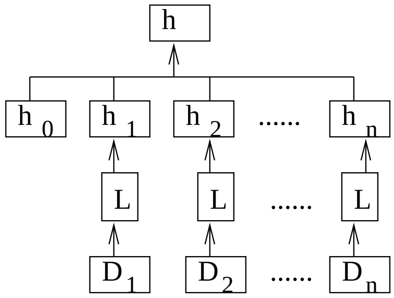

Fig. 2. Distributed Learning

distributed data sources. Space does not permit a detailed discussion of such scenarios.

3 Horizontal and Vertical Data Fragmentation

In many applications, the data set consists of a set of tuples where each tuple stores the values of relevant attributes. The distributed nature of such a data set can lead to at least two common types of data fragmentation: *horizontal fragmentation* wherein subsets of data tuples are stored at different sites; and *vertical fragmentation* wherein subtuples of data tuples are stored at different sites. Assume that a data set D is distributed among the sites $1, \cdots, n$ containing data set fragments $D_1, \cdots, D_n$. We assume that the individual data sets $D_1, \cdots, D_n$ collectively contain enough information to generate the complete dataset D. In many applications, 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. Even in cases where access to raw data may not be limited, the large size of the data sets makes it infeasible to assemble the complete data set D at a central location.

3.1 Horizontal Fragmentation

In the distributed setting, the data is fragmented in such a manner that each site contains a set of data tuples. The union of all these sets constitutes the complete dataset. If the individual data sets (horizontal fragments) are denoted by $D_1, D_2, \cdots, D_n$, and the corresponding complete data set by D, then *Horizontally Distributed Data* (HDD) has the following property: $D_1 \cup D_2 ... \cup D_n = D$, where $\cup$ denotes set union. Hence, in this case, a distributed learning algorithm L_d is exact with respect to the hypothesis inferred by a learning algorithm L if it is the case that:

$$L_d(D_1, D_2, \cdots, D_n) = L(D_1 \cup D_2 \cup \cdots \cup D_n). \quad (1)$$

The challenge is to achieve this guarantee without providing L_d with simultaneous access to $D_1, \cdots D_n$.

Similarly, we can envision horizontal fragmentation of data in the incremental setting.

3.2 Vertical Fragmentation

In the distributed setting, each data tuple is fragmented into several subtuples each of which shares a unique key or index. Thus, different sites store *vertical* fragments of the data set. Each vertical fragment corresponds to a subset of the attributes that describe the complete data set. It is possible for some attributes to be shared (duplicated) across more than one vertical fragments, leading to overlap between the corresponding fragments. Let $A_1, A_2, \cdots, A_n$ indicate the set of attributes whose values are stored at sites $1, \cdots, n$ respectively, and let A denote the set of attributes that are used to describe the data tuples of the complete data set. Then in the case of *Vertically Distributed Data* (VDD), we have: $A_1 \cup A_2 \cdots \cup A_n = A$. Let $D_1, D_2, \cdots, D_n$, denote the fragments of the dataset stored at sites $1, \cdots, n$ respectively, and let D denote the complete data set. Let the ith tuple in a data fragment D_j be denoted as $t^i_{D_j}$. Let $t^i_{D_j}$.index denote the *unique index* associated with tuple $t^i_{D_j}$ and let $\times$ denote the *join* operation. Then the following properties hold for VDD:

1. $D_1 \times D_2 \times \cdots \times D_n = D$, and
2. $\forall\ D_j, D_k,\ t^i_{D_j}.\text{index} = t^i_{D_k}.\text{index}$.

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 might be necessary to use combinations of attribute values to infer associations among tuples [1]. In what follows, we will assume the existence of unique indices in vertically fragmented distributed data sets.

In the case of vertically fragmented data, a distributed learning algorithm L_d is exact with respect to the hypothesis inferred by a learning algorithm L if it is the case that:

$$L_d(D_1, D_2, \cdots, D_n) = L(D_1 \times D_2 \times \cdots \times D_n). \tag{2}$$

The challenge is to guarantee this without providing L_d with simultaneous access to $D_1, \cdots D_n$.

Similarly, we can envision vertical fragmentation of data over time in the incremental setting. This is of special relevance in applications where data representation may be augmented over time by the addition of new attributes (e.g., measurements obtained using novel experiments in an ongoing scientific project). 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 combine the attribute values to infer associations among tuples. We can also envision data sets that are both horizontally and vertically fragmented in space, time, or both.

4 Learning Operators

Let $\mathcal{X}$ be a sample space where from the examples are drawn, and let $\mathcal{D}$ be the set of all possible subsets of the sample space $\mathcal{X}$. We can assume that the subsets in $\mathcal{D}$ are obtained from $\mathcal{X}$ by sampling according to different probability distributions. Let $\mathcal{C}$ be the space of all possible functions that we may want to learn or approximate, and $\mathcal{H}$ the space of the functions that a learning agent can draw on in order to construct approximations of the functions in $\mathcal{C}$. In a typical inductive learning scenario, $\mathcal{H}$ is a set of hypotheses. However, in the analysis that follows, it is useful to allow $\mathcal{H}$ to include not only the hypotheses but also other functions defined over $\mathcal{D}$. Examples of such functions include those that compute statistical summaries of a given data set, select subsets of a data set, or in general, extract useful information from a given data set. In what follows, we define some learning operators.

A *learning operator* is specified by $L : \mathcal{D} \to \mathcal{H}$, where L denotes any inductive learning algorithm or information extraction algorithm. It takes as input a dataset D and returns a function h that satisfies some specified criterion with respect to the data set. For example, if L is a consistent learner, it outputs a hypothesis that is consistent with the data. In other scenarios, L might compute relevant statistics from D.

The *"inverse" learning operator* is specified by $L^{-1} : \mathcal{H} \to \mathcal{D}$. As opposed to the learning operator, it takes as input a function h and returns a dataset D that satisfies some specified criterion with respect to L and the data set D. For example, L^{-1} might output when given h, a data set which when provided as input to L, results in the output h.

The *selection operator* is specified by $Sel : \mathcal{D} \to \mathcal{D}$. It generates a new dataset D' based on an existing dataset D by selecting examples according to a specified criterion (e.g., by sampling D according to some desired probability distribution).

The *union operator* $\cup : \mathcal{D} \times \mathcal{D} \to \mathcal{D}$ takes as arguments two datasets D_1 and D_2 and outputs a new dataset $D = D_1 \cup D_2$. It may represent the standard set union, multi-set union, or any suitably well-defined operation.

The *augmentation operator* $L_A : \mathcal{H} \times \mathcal{D} \to \mathcal{H}$ augments or refines a function h by incorporating new data D according to some specified criterion. For instance, it may minimally modify a hypothesis so as to be consistent with new data.

The *combination operator* $L_C^n : \mathcal{H}^n \to \mathcal{H}$ produces a new function h by exploiting the information provided by the given functions $h_1, h_2, \cdots, h_n$.

This set of definitions is meant to be merely illustrative (and not exhaustive) with respect to the types of operators that might be useful in incremental and distributed learning settings. Because the augmentation and combination operators are more general in some sense than the learning operator, it is desirable to enforce some consistency conditions among these operators. Let $h_\emptyset := L(\phi)$. Then we would like the following equalities to hold:

1. $L_A(h_\emptyset, D) = L(D)$,
2. $L_A(h, \phi) = h$, and

3. $L_C^2(h_\emptyset, h) = h$.

As a consequence, the following less general equalities should also hold:

4. $L_A(h_\emptyset, \phi) = L(\phi) = h_\emptyset$, and
5. $L_C^2(h_\emptyset, h_\emptyset) = L(\phi) = h_\emptyset$.

The previous conditions basically ensure that the two more general operators L_A and L_C^2 behave nicely when provided an empty dataset as one of the inputs.

5 Incremental and Distributed Learning Criteria

Batch learning algorithms have been the subject of extensive experimental and theoretical analysis. Consequently, it is desirable to develop a theoretical framework which allows us to gain useful insights into the performance of distributed and incremental algorithms by relating them their batch or centralized counterparts.

In what follows, we define *exactness* of a distributed or incremental learning algorithm. This definition is intended to illustrate the sorts of analysis that are facilitated by the theoretical framework that is sketched out in this paper. A similar analysis can be performed with respect to other performance criteria that are motivated by the needs of specific applications.

Definition 1. *Given two datasets D_1 and D_2, and a learning algorithm L for a function class $\mathcal{H}$, we say that $D_1 \cup D_2$ is exact incremental-learnable and exact distributed-learnable with respect to the algorithm L if the conditions*

$$L_A(L(D_1), D_2) = L(D_1 \cup D_2), \text{ and} \tag{3}$$

$$L_C^2(L(D_1), L(D_2)) = L(D_1 \cup D_2) \tag{4}$$

hold (respectively).

Note that the union operator used above can have different meanings in different scenarios. In some cases, it may denote standard set union; in some others, it may stand for an operation that combines subtuples of a data tuple from different data sets based on the unique index that helps associate the data subtuples (see the discussion of horizontally and vertically fragmented data in the previous section for details).

The conditions for exact incremental and exact distributed learning can be easily generalized for the case where the complete data set is distributed among n data sets. In many real world problems involving sufficiently expressive concept classes, exact incremental or distributed learning may not be possible even in principle. In other instances, although possible in principle, it may not be feasible in practice for computational reasons. At present, a characterization of hypothesis classes that lend themselves to exact or approximate distributed or incremental learning is lacking. From a practical standpoint, the design and implementation of data and hypotheses representations that can support computationally efficient and scalable distributed and incremental learning algorithms is clearly of interest.

6 Models of Distributed and Incremental Learning

In this section we will explore relationships among some of the learning operators introduced above and define them in term of the others. Then we will explore some properties of instance and hypothesis representations and operators which guarantee the existence of exact incremental and exact distributed learning under certain assumptions. Provable equivalences among certain learning operators (or combinations thereof) can help to transform the algorithms developed in one setting (e.g. distributed learning) to another setting (e.g., incremental learning) under certain well-defined conditions.
Consider the learning operators defined as follows:

1) $L_A(h, D) := L(L^{-1}(h) \cup D)$
2) $L_C^2(h_1, h_2) := L(L^{-1}(h_1) \cup L^{-1}(h_2))$
3) $L_C^2(h_1, h_2) := L_A(L_A(h_\emptyset, L^{-1}(h_1)), L^{-1}(h_2))$
4) $L_C^2(h_1, h_2) := L_A(h_1, L^{-1}(h_2))$
5) $L_A(h, D) := L_C^2(h, L(D))$

Lemma 1. *Assume that $L(L^{-1}(h)) = h$. Then definition* **3)** *is equivalent to definition* **4)**.

Proof. $L_A(L_A(h_\emptyset,\ L^{-1}(h_1)),\ L^{-1}(h_2)) = L_A(L(L^{-1}(h_1))$
$L^{-1}(h_2)) = L_A(h_1, L^{-1}(h_2))$.
The first equality follows from the consistency of L_A and the second one from the assumption made.

Theorem 1. *Assume that $L^{-1}(L(D)) = D$. Then the operators given by definition* **1)** *and definition* **2)** *satisfy the exact learning criteria. That is,*

(1) $L(D_1 \cup D_2) = L_A(L(D_1), D_2)\ \forall D_1, D_2 \in \mathcal{D}$ (in the incremental setting)
(2) $L(D_1 \cup D_2) = L_C^2(L(D_1), L(D_2))\ \forall D_1, D_2 \in \mathcal{D}$ (in the distributed setting)

Proof. The proof of (1) above follows from the observation that $L_A(L(D_1), D_2) = L(L^{-1}(L(D_1)) \cup D_2) = L(D_1 \cup D_2)$. The proof of (2) above follows from the observation $L_C^2(L(D_1),\ L(D_2)) = L(L^{-1}(L(D_1)) \cup L^{-1}(L(D_2))) = L(D_1 \cup D_2)$.

Theorem 2. *Assume that $L^{-1}(L(D)) = D$ and that L_A satisfies the Exact Learning criteria. Then the operator given by definition* **4)** *also satisfies the exact learning criteria in the distributed setting. That is,*
$L(D_1 \cup D_2) = L_C^2(L(D_1), L(D_2))\ \forall D_1, D_2 \in \mathcal{D}$

Proof. $L_C^2(L(D_1),\ L(D_2)) = L_A(L(D_1),\ L^{-1}(L(D_2))) = L_A(L(D_1),\ D_2) = L(D_1 \cup D_2)$

Theorem 3. *Assume that L_C satisfies the exact learning criteria. Then the operator given by definition* **5)** *also satisfies the exact learning criteria in the incremental setting. That is, $L(D_1 \cup D_2) = L_A(L(D_1), D_2)\ \forall D_1, D_2 \in \mathcal{D}$*

Proof. We have: $L_A(L(D_1), D_2) = L_C^2(L(D_1), (L(D_2))) = L(D_1 \cup D_2)$

The results of this section show how we can emulate some operators using other operators so as to guarantee exact learning under certain assumptions. The condition $L^{-1}(L(D)) = D$ is quite strong and is seldom met in practice. The next section explores the design of exact incremental and distributed learning algorithms under weaker assumptions. A more complete characterization of the necessary and sufficient conditions for exact or approximate distributed and incremental learning is a subject of our ongoing research.

7 Designing exact learning agents

One approach to devising distributed or incremental algorithms based on an existing batch or centralized learning algorithm is by identifying the information requirements of the learner and designing efficient means of providing the necessary information to it in the distributed or incremental setting. This decomposition of the learning task into *information extraction* and *hypothesis generation* phases offers a general approach to adapting some of the existing learning algorithms to work in the distributed setting. The *hypothesis generation* component of the algorithm can be thought of as the *control* part of the algorithm, which triggers the execution of the *information extraction* part as needed. The execution of the two parts is typically interleaved in time. In this model of distributed learning, only the *information extraction* component has to effectively cope with the distributed nature of the data.

We illustrate this approach to design some incremental and distributed algorithms based on existing batch algorithms (e.g., instance based learning, decision tree learning, and support vector machines induction).

7.1 Incremental and distributed k-nearest neighbor classifiers

The k-nearest neighbor algorithm is an example of an instance based learning algorithm that can be easily transformed into an exact algorithm for learning from horizontally fragmented data in both incremental as well as distributed settings. The learning phase of a k-NN algorithm consists simply in storing the data and the information extraction is done during the classification phase. Thus, in the k-NN case we have incremental/distributed classification as opposed to the most algorithms where we have incremental/distributed learning, but centralized classification. The classification phase in a k-NN algorithm can be separated in two phases L_{extr} for information extraction phase, which returns the k closest neighbors of a given example x, and L_{proc} for information processing phase which will take a majority vote among the k closest neighbors. The representation of the result of L_{extr} part will be the same as the representation of the instances. In this case we can easily define an inverse operator L_{extr}^{-1} as being the identity. Therefore $L_{extr}^{-1}(L_{extr}(D)) = D'$, where D' contains the k closest points to the point x which should be classified. We will get the following sufficient conditions for exact incremental/distributed information extraction:

Incremental case:

$$L_{extr}(D_1 \cup D_2) = L_A(L_{extr}(D_1), D_2) := L_{extr}(L_{extr}^{-1}(L_{extr}(D_1)) \cup D_2). \quad (5)$$

Distributed case:

$$L_{extr}(D_1 \cup D_2) = L_C^2(L_{extr}(D_1), L_{extr}(D_2)) :=$$
$$L_{extr}(L_{extr}^{-1}(L_{extr}(D_1)) \cup L_{extr}^{-1}(L_{extr}(D_2))). \quad (6)$$

We call these properties *u-closure* properties. Suppose we have an algorithm $k\_nn(D, x)$ which compute the k closest neighbors of a new instance x in the set D (in the information extraction phase). The solution given by the k-NN algorithm for the instance x can be written as follows:

$$h(x) = \arg\max_{c \in \mathcal{C}} |\{y | y \in k\_nn(x),\ h(y) = c\}|,$$

where $\mathcal{C}$ is the set of possible classes. Given n datasets $D_1, \cdots, D_N$ and an instance x to be classified, the information extraction algorithm works as follows: $k\_nn(k\_nn(D_1, x) \cup \cdots \cup k\_nn(D_n, x), x)$. Similarly, incremental information extraction (in the case of two data sets D_1 and D_2) works as follows: $k\_nn(k\_nn(D_1, x) \cup D_2, x)$.

Theorem 4. *The following two equalities hold for arbitrary data sets $D_1, \cdots, D_n$, and a new instance x:*

$$k\_nn(k\_nn(D_1, x) \cup \cdots \cup k\_nn(D_n, x), x) = k\_nn(D_1 \cup \cdots \cup D_n, x).$$

$$k\_nn(k\_nn(D_1, x) \cup D_2, x) = k\_nn(D_1 \cup D_2, x)$$

These two equalities guarantee the *u-closure* properties, and hence the resulting algorithms are exact.

7.2 Incremental and Distributed Induction of Support Vector Machines

Support Vectors Machines (SVM) [16] have proved to be a successful technique for batch learning. SVM summarizes the data in a very compact form by identifying the set of instances (the so-called *support vectors*) that specify the maximal margin hyperplane separating the two classes. We can devise an algorithm for exact learning of support vector machines from horizontally fragmented data in both incremental as well as distributed settings by decomposing the learning task into two phases: first extract some information about datasets using an algorithm L_{extr} for information extraction, and then process the information to generate a hypothesis using L_{proc} as needed. The information extracted by L_{extr} will have the same representation as the instances in the training set. More specifically, it will select a subset of the training set that is sufficient for exact learning. Taking again L_{extr}^{-1} to be the identity operator ($L_{extr}^{-1}(L_{extr}(D)) = L_{extr}(D)$), we can get similar conditions for incremental and distributed learning. Specifically, we can guarantee exact incremental and distributed learning if the following *u-closure* properties hold:

$$L_{extr}(D_1 \cup D_2) = L_A(L_{extr}(D_1), D_2) = L_{extr}(L_{extr}^{-1}(L_{extr}(D_1)) \cup D_2) \tag{7}$$

in the incremental setting, and

$$\begin{aligned} L_{extr}(D_1 \cup D_2) = L_C^2(L_{extr}(D_1), L_{extr}(D_2)) = \\ L_{extr}(L_{extr}^{-1}(L_{extr}(D_1)) \cup L_{extr}^{-1}(L_{extr}(D_2))) \end{aligned} \tag{8}$$

in the distributed setting.

It can be easily seen that support vectors do not satisfy this property but the convex hulls of the instances that belong to the two classes do. This observation leads to exact incremental and distributed learning algorithms [3].

7.3 Distributed decision tree algorithms

An approach to exact learning of decision trees from horizontally and vertically fragmented data is proposed in [12]. 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. These counts are used by a decision tree construction algorithm [11] to find the best attribute among the set of examples being considered.

The formal description of the algorithm in the case of HDD is the following:

$$L_{HDD}(D_1, \cdots, D_n) := I_P(I_E(D_1), \cdots, I_E(D_n)). \tag{9}$$

where I_E is an operator which collects statistics about attributes at different sites and $L_C^n \equiv I_P$ combines the information gained and constructs the decision tree.

In vertically distributed datasets, each example has a unique index associated with it. Subtuples of an example are distributed among different datasets. However, they can be related to each other using their shared index. While constructing a branch of decision tree, the count of examples that satisfy the constraints on attribute values along the branch is found using unique indices to relate the subtuples of an example. To find the best attribute, a pass is made through all data sites to compute the count of examples.

The formal description of the algorithm in the case of VDD is as follows:

$$L_{VDD}(D_1, \cdots, D_n) := I_P(I_E(D_1), \cdots, I_E(D_n)). \tag{10}$$

In this case, I_E collects statistics and finds the best attribute at each location, and $L_C^n \equiv I_P$ uses this information to construct the tree. Specifically the statistics which are collected consist in counts for combinations of attribute values. As shown in [12], this approach yields exact and efficient algorithms for learning decision trees from horizontally, vertically, and both horizontally and vertically fragmented data sets in the distributed setting.

8 Additional Algorithms

This section explores the specification of some additional algorithms that have been proposed in the literature for distributed and incremental learning within the proposed theoretical framework.

8.1 Bagging, Boosting and Stacked generalization

Bagging [2] and Stacked Generalization [17] are hypotheses combination techniques used typically to improve the accuracy of a learning algorithm. Bagging works as follows:

$$L_{Bagging}(D) := Majority\_Vote(L_1(Sel(D)), \cdots, L_n(Sel(D))) \tag{11}$$

where $L_1, \cdots, L_n$ are arbitrary learning algorithms, and the combination is done using a majority vote. It is possible to develop similar characterizations of stacked generalization, boosting, and related techniques. This opens up the possibility of approximate constructing incremental and distributed learning algorithms based on such techniques. The interested reader is refered to [8] for an example.

8.2 Meta-Learning

Meta-learning [9] is a technique which combines independent classifiers generated from distributed databases into a single global classifier. The classifiers generated from individual databases are integrated so as to improve the overall predictive accuracy of the combined classifier. More specifically, the individual classifiers are used to generate predictions on a separate validation test. The predictions and the validation test are put together to form a new data set. The final classifier is obtained by a new learning process applied to this dataset. This process can be captured as follows:

$$L_{Meta}(D_1, \cdots, D_n, D_0) := L(L^{-1}(L_1(D_1)|D_0) \cup \cdots \cup L^{-1}(L_n(D_n)|D_0)) \tag{12}$$

where $D_1 \cup \cdots D_n = D$ and D_0 is the validation set and L is a learning operator.

8.3 DAGGER

Another strategy to combine multiple learned models can be found in [4]. The goal of DAGGER is to learn a single comprehensive model from distributed data sets. The key idea is to learn the hypotheses $h_1, \cdots, h_n$ from the datasets $D_1, \cdots, D_n$ and use them to guide sampling of a new set of informative examples from these datasets. Then the learning operator L generates a final model from the union of the informative examples as follows:

$$L_{DAGGER}(D_1, \cdots, D_n) := L(L^{-1}(L_1(D_1)) \cup \cdots \cup L^{-1}(L_n(D_n))), \tag{13}$$

where the operators used are similar to those in meta-learning, and each hypothesis $h_i = L_i(D_i)$.

8.4 Incremental tree induction

Incremental tree induction (ITI) [15] is an algorithm that sequentially restructures a hypothesis in the form of a decision tree as new examples are encountered on the basis of some statistics that are maintained. Examples are processed one at a time. Hence, each new set contains only one example. Let n be the number of datasets (in this case, examples). We can describe this algorithm as follows:

$$L_{ITI}(D_1, \cdots D_n) := L_A(L_A(\cdots(L_A(h_\emptyset, D_1)\cdots, D_{n-1}), D_n), \tag{14}$$

where L_A is the algorithm which updates the statistics and $h_\emptyset = L(\emptyset)$. This algorithm does not guarantee exact learning in the incremental setting.

9 Summary and Future Research

The results presented in this paper constitute some useful (albeit tentative) first steps toward the development of a theoretical framework for the specification and analysis of a class of learning problems that arise in open-ended, dynamic environments. Such learning tasks arise in many real-world applications involving knowledge acquisition from multiple, distributed, possibly autonomous data and knowledge sources. We have introduced a family of learning operators and illustrated their use to formally describe some existing approaches and to design new distributed and incremental learning algorithms with provable performance guarantees. We have identified some properties of instance and hypothesis representations and learning operators that make exact learning possible in some open-ended, dynamic environments under certain assumptions. Work in progress is aimed at the elucidation of the necessary and sufficient conditions that guarantee the existence of exact or approximate cumulative multi-agent learning systems in general, and different types of incremental and distributed learning agents in particular, in terms of the properties of instance and hypothesis representations and learning operators, communication operators, and knowledge requirements of agents. Also of interest are PAC-style and mistake-bound analysis of incremental and distributed learning in multi-agent learning systems under different assumptions. Long term goals of this research include: design theoretically well-founded multi-agent systems for learning from interaction with open-ended dynamic environments that include multiple data and knowledge sources (including other agents) and application of such multi-agent learning systems to large-scale data-driven knowledge discovery tasks in applications such as bioinformatics.

Acknowledgements

This research was funded in part by grants from the National Science Foundation (9982341, 9972653, 0087152), the John Deere Foundation, and Pioneer Hi-Bred, Inc. to Vasant Honavar.

References

1. Bhatnagar, R., Srinivasan, S.: Pattern Discovery in Distributed Databases. Proceedings of AAAI, AAAI Press, 1997.
2. Breiman, L.: Arcing Classifiers. Annals of Statistics, Volume 26, 1998.
3. Caragea, D., Silvescu, A., Honavar, V.: Agents that Learn from Distributed Dynamic Data Sources. Proceedings of the Workshop on Learning Agents, Agents-00/ECML-00. Barcelona, Spain. June 2000.
4. Davies, W., Edwards, P.: DAGGER: A New Approach to Combining Multiple Models Learned from Disjoint Subsets. Machine Learning 2000.
5. Honavar, V., Miller, L., Wong, J.: Distributed Knowledge Networks. Proceedings of the IEEE Conference on Information Technology, Syracuse, NY, 1998.
6. Kargupta, H., Park, B., Hershberger, D., Johnson, E.: Collective Data Mining: A New Perspective Toward Distributed Data Mining. Advances in Distributed and Parallel Knowledge Discovery, Eds: Hillol Kargupta and Philip Chan. AAAI Press. 2000.
7. Littlestone, N.: The weighted majority algorithm. Information and Computation, 108:212-261, 1994.
8. Polikar, R., Udpa, L., Udpa, S., Honavar, V.: Learn++: An Incremental Learning Algorithm for Multilayer Perceptron Networks. Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP) 2000. Istanbul, Turkey. In press.
9. Prodromidis, A.L., Chan, P.K.: 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, 2000.
10. Provost, F., Hennessy, D.: Scaling Up: Distributed Machine Learning with Cooperation. Proceedings of the Fourteenth National Conference on Artificial Intelligence, 1996.
11. Quinlan, J.R.: Induction of Decision Trees. Machine Learning, vol. 1, pp 81-106, 1986.
12. Sharma, T., Silvescu, A., Andorf, C., Caragea, D., Honavar, V.: Algorithms for Learning from Distributed Data Sets. Tech. Rep. ISU-CS-TR 2000-10. Department of Computer Science, Iowa State University, Ames, IA, May 2000.
13. Srivastava, A., Han, E.H., Kumar, V., Singh V.: Parallel Formulations of Decision-Tree Classification Algorithms. Data Mining and Knowledge Discovery: An International Journal, vol. 3, no. 3, pp 237-261, September 1999.

14. Thrun, S., Faloutsos, C., Mitchell, M., Wasserman, L.: Automated Learning and Discovery: State-of-the-art and research topics in a rapidly growing field. AI Magazine, 1999.
15. Utgoff, P.E., Berkman, N.C., Clouse, J.A.: Decision Tree Induction Based on Efficient Tree Restructuring. Machine Learning, 1997.
16. Vapnik, V.: Statistical Learning Theory. Springer-Verlag, New York, 1998.
17. Wolpert, D.H.: Stacked Generalization. Neural Networks, 5:241-259, 1992.