Agents that Learn from Distributed Dynamic Data Sources[*]

Doina Caragea
Artificial Intelligence Research Laboratory,
Department of Computer Science,
Iowa State University,
Ames, IA 50011 USA
dcaragea@cs.iastate.edu

Adrian Silvescu
Artificial Intelligence Research Laboratory,
Department of Computer Science,
Iowa State University,
Ames, IA 50011 USA
silvescu@cs.iastate.edu

Vasant Honavar
Artificial Intelligence Research Laboratory,
Department of Computer Science,
Iowa State University,
Ames, IA 50011 USA
honavar@cs.iastate.edu

ABSTRACT

Recent advances in high throughput data acquisition and data storage technologies have made it possible to gather and store large amounts of data at increasing rates. Much of this data is physically distributed and the data sets grow in size at a fairly fast rate. Translating the ability to gather data into fundamental gains in understanding of the respective domains (e.g., bioinformatics) calls for data-driven knowledge acquisition agents that can incrementally process large amounts of data that is distributed in space or time.

More generally, design of agents that learn from their interactions with open-ended, dynamic environments (including other agents) call for the design of learning agents with provable convergence properties in incremental and distributed settings.

We propose a theoretical framework for specification and analysis of a large class of learning problems that arise in open-ended, dynamic environments consisting of multiple, distributed data and knowledge sources. We state some properties of instance and hypothesis representations and learning operators that constitute necessary and sufficient conditions for incremental and distributed learning of pattern classifiers.

We demonstrate the use of the proposed framework to design learning agents based on variants and extensions of the Support Vector Machine (SVM) algorithm in distributed and incremental learning settings. We conclude with a brief discussion of some promising directions for further research.

[*]This research was supported in part by grants from the National Science Foundation (NSF ACI 9982341), the John Deere Foundation, and Pioneer Hi-Bred, Inc.

1. LEARNING IN OPEN-ENDED, DYNAMIC, DISTRIBUTED ENVIRONMENTS

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. 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 agent-based systems, 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, data repositories are autonomous and physically distributed. Thus it is desirable to perform as much analysis as possible where the data 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 the 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. A key problem in acquiring useful knowledge from large, dynamic, distributed data sources is that of devising *cumulative learning* algorithms that can incrementally incorporate new data as they become available ([12, 21]) over time (*incremental learning*) or across space (*dis-*

tributed learning) or both. More generally, design of agents that learn from their interactions with open-ended, dynamic environments consisting of several distributed, autonomous, data and knowledge sources (including other agents) call for learning algorithms with provable convergence properties in such settings.

In this paper, we focus on a framework for the specification, analysis, and design of learning agents for incremental and distributed learning of pattern classifiers. However, the proposed framework can be extended to other types of learning problems in open-ended, dynamic environments.

2. SUPPORT VECTOR MACHINES

Let $\mathcal{E} = \{(\mathbf{X}_1, y_1), (\mathbf{X}_2, y_2), \cdots, (\mathbf{X}_l, y_l)\}$, where $\mathbf{X}_i \in \mathcal{R}^N$ and $y_i \in \{-1, 1\}$ be a set of training examples for a 2-category classifier. Let

$$S^+ = \{\mathbf{X}_i | (\mathbf{X}_i, y_i) \in \mathcal{E} \;\&\; y_i = +1\}$$

and

$$S^- = \{\mathbf{X}_i | (\mathbf{X}_i, y_i) \in \mathcal{E} \;\&\; y_i = -1\}.$$

Suppose the training data is *linearly separable.* Then it is possible to find a hyperplane that partitions the N-dimensional pattern space into two half-spaces R^+ and R^- such that $S^+ \subset R^+$ and $S^- \subset R^-$.

The set of such hyperplanes (the solution space) is given by

$$f_{\mathbf{W},b} = \text{sign}(\mathbf{W} \cdot \mathbf{X} + b)$$

where each solution hyperplane can be specified by a pair $(\mathbf{W}, b)$ such that:

$$\mathbf{W} \cdot \mathbf{X}_i + b \geq 1 \;\; \forall \mathbf{X}_i \in S^+$$

and

$$\mathbf{W} \cdot \mathbf{X}_i + b \leq -1 \;\; \forall \mathbf{X}_i \in S^-.$$

A solution hyperplane which satisfies the additional constraint

$$\min_{i=1,\cdots,l} |\mathbf{W} \cdot \mathbf{X}_i + b| = 1$$

is called the canonical hyperplane and defines an one-to-one correspondence between the solution space and the set of pairs $(\mathbf{W}, b)$. The distance between a hyperplane defined by a pair $(\mathbf{W}, b)$ and the nearest points from the training set is given by:

$$d(\mathbf{X}, (\mathbf{W}, b)) = \frac{|\mathbf{W} \cdot \mathbf{X} + b|}{\|\mathbf{W}\|},$$

then the distance corresponding to the canonical hyperplane is equal to $\frac{1}{\|\mathbf{W}\|}$. SVM selects from among the hyperplanes that correctly classify the training set, one that minimizes $\|\mathbf{W}\|$. This involves solving the following quadratic programming problem:

$$\min_{\mathbf{W},b} \Phi(\mathbf{W}) = \frac{1}{2}\|\mathbf{W}\|^2$$

subject to

$$y_i(\mathbf{W} \cdot \mathbf{X}_i + b) \geq 1 \;\; \forall i = 1, \cdots, l.$$

The hyperplane which minimize $\|\mathbf{W}\|$ is the same as the hyperplane for which the *margin* of separation between the two classes is maximized.

If the goal of the classification problem is fo find a linear classifier for a non-separable training set, a new set of variables, called slack variables, can be introduced to define the following optimization problem:

$$\min_{\mathbf{W},b,\Xi} \Phi(\mathbf{W}, \Xi) = \frac{1}{2}\|\mathbf{W}\|^2 \;\; + C(\sum_{i=1}^{i=l} \xi_i)^k$$

subject to

$$y_i(\mathbf{W} \cdot \mathbf{X}_i + b) \geq 1 - \xi_i \;\; \forall i = 1, \cdots, l$$

$$\xi_i \geq 0 \;\; \forall i = 1, \cdots, l.$$

If the training examples are not linearly separable, an SVM works by mapping the training set into a higher dimensional *feature* space using an appropriate kernel function ψ. The kernel function is chosen to ensure that the data becomes linearly separable in the feature space. Therefore the problem can be solved using linear decision surfaces in the higher dimensional space. Any consistent training set can be made separable with an appropriate choice of a feature space of a sufficiently high dimensionality. However, in general, this can cause the learning algorithm to overfit the training data resulting in poor generalization. SVM avoids this problem by choosing the maximal margin hyperplane from the set of all separating hyperplanes ([22]).

The solution given by the SVM will be of the following form:

$$f(\mathbf{X}) = \text{sign}(\mathbf{W} \cdot \psi(\mathbf{X}) + b) = \text{sign}(\sum_{i=1}^{l} y_i \lambda_i \psi(\mathbf{X}) \cdot \psi(\mathbf{X}_i) + b)$$

where $(\mathbf{W}, b)$ defines the solution hyperplane and $\mathbf{W}$ is a weighted sum of the training instances in the feature space. Here, λ_i is the weight assigned to training instance $\mathbf{X}_i$. Thus, the maximum margin separating hyperplane in the feature space can be represented as a weighted sum of the training patterns. In this weighted sum, the training patterns that lie far from this hyperplane receive weights of zero and only those patterns that lie close to the decision boundary between the two classes have non-zero weights. The training patterns that have non-zero weights are called the *support vectors.* The number of support vectors is usually a small fraction of the size of the training set. This raises the possibility that SVM algorithm can perhaps be adapted in a relatively straightforward fashion to work in an incremental setting ([20]).

3. LEARNING IN OPEN-ENDED, DYNAMIC ENVIRONMENTS: INCREMENTAL LEARNING AND DISTRIBUTED LEARNING

The incremental learning problem can be formulated as follows: the learner incrementally refines a hypothesis (or a set of hypotheses) as new data become available. Because of the large volume of data involved, it may not be practical to store and access the entire data set during learning. Thus, the learner does not have access to previously analyzed data

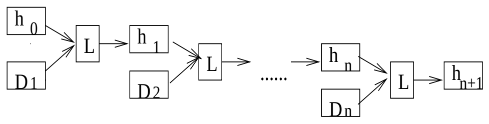

Figure 1: Incremental Learning

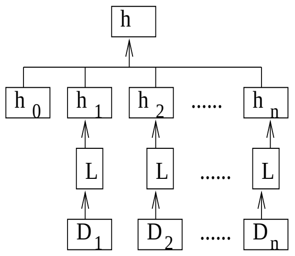

Figure 2: Agglomerative Distributed Learning

(with the possible exception of a relatively small subset of critical examples that is stored by the learner). A generic incremental learning scenario is shown in figure 1.

We assume that data collections D_1, D_2 and so on are made available at discrete instants in time t_1, t_2, etc. We start with initial hypothesis h_0 which constitutes prior knowledge of the domain. We assume that the system is not permitted 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.

One type of distributed learning problem (the *agglomerative distributed learning* task) can be formulated as follows: A hypothesis is learned independently by learners situated at each of the physically distributed data repositories and the resulting set of hypotheses are somehow combined to obtain the desired pattern classifier. An agglomerative distributed learning scenario is shown in figure 2.

An incremental or distributed learning algorithm is said to be exact with respect to some criterion of interest (e.g., the learned hypothesis, expected generalization accuracy) if it is guaranteed to yield the same result as that obtained in the batch learning scenario wherein the entire dataset is accessible to the learning algorithm during learning. 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.

4. DESIGN OF INCREMENTAL AND DISTRIBUTED LEARNING AGENTS FOR PATTERN CLASSIFICATION

We consider the design of learning agents for data-driven acquisition of pattern classification knowledge from distributed, dynamic data sources to explore the challenges of designing agents that learn from their interactions with dynamic, open-endedenvironments that consist of multiple, autonomous data and knowledge sources (including other agents). In particular, we focus on the design of variants of the Support vector machines (SVM) ([6, 22, 9, 16]) algorithm for provably exact incremental and distributed learning. This is motivated by the demonstrated success of SVM algorithms in several data-driven knowledge discovery applications including gene expression analysis using microarray data ([4]), text classification ([11]), among others.

4.1 Incremental and Distributed Learning Using SVM

In what follows, we will assume without loss of generality that the training patterns are represented (if necessary, using a suitable kernel function) in a feature space in which the data set is linearly separable.

Suppose that two data sets D_1 and D_2 become available to the learner at different instants in time (say t_1 and t_2). Our goal is to learn a binary classifier using D_1 and D_2 in an incremental setting. If the learning is exact, the resulting classifier should be the same as the one obtained in the batch learning setting using the data set $D = D_1 \cup D_2$.

A naive approach to incremental learning agents using SVM ([20]) works as follows:

1. Have an SVM based learning agent process D_1 and generate a set of support vectors SV_1
2. Add SV_1 to D_2 to get a data set D_2'
3. Have an SVM based learning agent process D_2' and generate a set of support vectors SV_2

One can envision a similar approach to agglomerative distributed learning in a setting wherein the data sets D_1 and D_2 are physically distributed:

1. Have an SVM based learning agent A_1 process D_1 and generate a set of support vectors SV_1
2. Have an SVM based learning agent A_2 process D_2 and generate a set of support vectors SV_2
3. Have an SVM based learning agent A process $SV_1 \cup SV_2$ to obtain the set of support vectors SV.

These two algorithms work reasonably well in practice if the two data sets D_1 and D_2 each individually are representative of the entire training set $D_1 \cup D_2$, so that the hyperplane determined by the support vectors derived from either one of them does not differ very much from that derived from

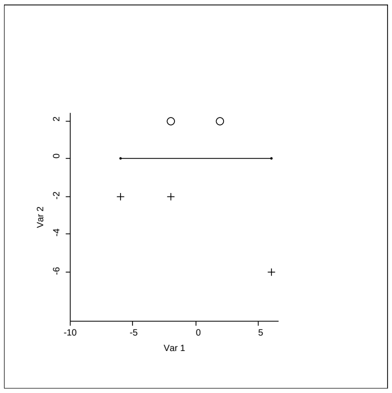

Figure 3: Counterexample to the naive approach: dataset D_1

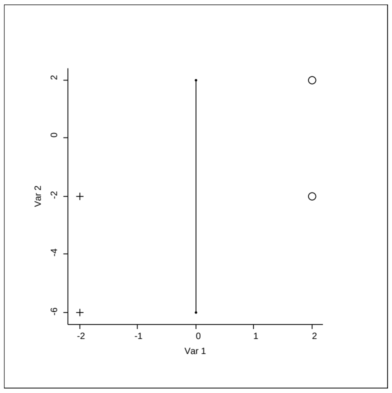

Figure 4: Counterexample to the naive approach: dataset D_2

the entire data set. However, if that is not the case, it can be shown that the resulting hyperplane can be an arbitrarily poor approximation of the target hypothesis ([5]). This can be seen by considering the scenario illustrated in figures 3, 4, and 5.

Suppose that
$D_1 = \{(-6,-2,+),(-2,-2,+),(6,-6,+),(-2,2,-),(2,2,-)\}$,
and
$D_2 = \{(-2,-2,+),(-2,-6,+),(2,2,-),(2,-2,-)\}$.

Thus, the set $D_1 \cup D_2$ is clearly linearly separable. We can run the following experiment using an SVM algorithm (e.g., SVM^{light} ([13]):

- Apply SVM to $D_1 \cup D_2$ to get the support vector set $SV(D_1 \cup D_2) = \{(-2,-2,+),(6,-6,+),(2,-2,-)\}$
- Apply SVM to D_1 to get the support vector set $SV_1 = \{(-6,-2,+),\ (-2,-2,+),(-2,2,-),(2,2,-)\}$
- Apply SVM to D_2 to get the support vector set $SV_2 = \{(-2,-2,+),(-2,-6,+),\ (2,2,-),(2,-2,-)\}$
- Apply SVM to $SV_1 \cup SV_2$ to get the support vector set $SV(SV_1 \cup SV_2) = \{(-2,-2,+),(-2,2,-),(2,-2,-)\}$

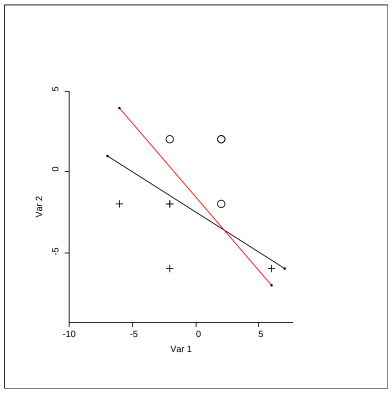

Figure 5: Counterexample to the naive approach: entire dataset $D = D_1 \cup D_2$

Note that $SV(D_1 \cup D_2) \neq SV(SV_1 \cup SV_2)$. Because the separating hyperplane depends on the support vectors, this implies that the solution found by the SVM in the incremental setting is different from the solution found by batch learning. Thus, the naive approach to incremental learning using SVM loses important boundary information (the data point $(6,-6,+)$ in the example above). Since depending on the underlying distribution over the pattern space, this can happen with an arbitrarily high probability, the resulting classifier can have an arbitrarily high error. Therefore a better approach to designing learning agents that are effective in an incremental setting is necessary.

4.2 Incremental and Distributed Learning Agents

Let L be an inductive learning algorithm for pattern classification. Suppose L works with an instance space $\mathcal{X}$ and a hypothesis space $\mathcal{H}$. Each h in $\mathcal{H}$ classifies instances in $\mathcal{X}$. For example, if the hypotheses are binary, each instance is classified into one of 2 classes. We fix the instance and hypothesis representations. Then L takes as input, a labeled finite training set $\mathcal{E} = \{(\mathbf{X}_i, y_i)\}$ and outputs a hypothesis $h \in \mathcal{H}$. That is, $L(\mathcal{E}) = h$. If L is a consistent learner, the hypothesis produced by L correctly classifies each of the training instances.

In what follows, we consider a special case wherein L and $\mathcal{X}$ and $\mathcal{H}$ have the property that the hypotheses have a direct encoding in terms of training instances. For example, some instance based learning algorithms and the resulting classifiers ([1]) and support vector machines satisfy this property. Under this assumption, we can think of the output of L as simply a set of instances (e.g., support vectors in the case of SVM) plus some additional information (e.g., the weights associated with the instances in the case of SVM). If we ignore the additional information for the time being, both the output as well as input to L have the same representation (namely that used to encode the training instances).

In this case, given a learning algorithm L and data sets $D_1, D_2, \cdots, D_N$, a sufficient condition for exact learning, i.e.

- $L(...L(L(D_1) \cup D_2)... \cup D_N) = L(D_1 \cup ... \cup D_N)$ (incremental case)

- $L(L(D_1) \cup ... \cup L(D_N)) = L(D_1 \cup ... \cup D_N)$ (distributed case)

is the following (*u-closure*) property:

- $L(L(D) \cup D') = L(D \cup D')$(*incremental case*)
- $L(L(D) \cup L(D')) = L(D \cup D')$(*distributed case*)

for any arbitrary sets D and D'.

Support Vectors do not satisfy this property but the convex hulls of the instances that belong to the two classes do ([10]).

The convex hull of a set of points S, denoted $conv(S)$ is the smallest convex set containing S. That is,

$$conv(S) = \{\mathbf{X} \in R^N | \mathbf{X} = \sum_{\mathbf{X}_i \in S} \lambda_i \mathbf{X}_i, \sum \lambda_i = 1, \lambda_i \geq 0\}.$$

Thus, $conv(S)$ is the set of all non-negative affine combinations of points from S. If the set S is finite, the convex hull is a convex polyhedron given by the intersection of a finite number of closed halfspaces. We are interested in the vertices of this polyhedron because they uniquely define the convex hull. The *u-closure* property is a very well known thorem for convex sets:

Theorem: Let A_1 and A_2 be two convex sets. Then:

1. $conv(conv(A_1) \cup A_2) = conv(A_1 \cup A_2)$
2. $conv(conv(A_2) \cup conv(A_2)) = conv(A_1 \cup A_2)$

Since the *u-closure* property is satisfied for convex sets it follows that by using convex hulls we can devise exact incremental and distributed learning agents in the general case (N-datasets). Let $VConv(A)$ denote the vertices that define the convex hull of a convex set A. It can be easily shown that:

1. $VConv(VConv(A_1) \cup A_2) = VConv(A_1 \cup A_2)$
2. $VConv(VConv(A_2) \cup VConv(A_2)) = VConv(A_1 \cup A_2)$

Suppose that we have two datasets D_1 and D_2 such that $D_1 \cup D_2$ is linearly separable. Let $D_i(+)$ and $D_i(-)$ denote the positive and negative instances in the data set D_i. Similarly, let $D(+)$ and $D(-)$ denote the positive and negative instances in the data set D. Let $SVM(D)$ denote the result of applying the SVM algorithm to the data set D. The incremental SVM learning agents can be designed as follows:

1. Have the learning agent compute $VConv(D_1(+))$ and $VConv(D_1(-))$
2. Add $VConv(D_1(+))$ to $D_2(+)$ to obtain $D_2'(+)$
3. Add $VConv(D_1(-))$ to $D_2(-)$ to obtain $D_2'(-)$
3. Have the learning agent compute $VConv(D_2'(+))$ and $VConv(D_2'(-))$.
4. Generate a training
   $D_{12} = VConv(D_2'(+)) \cup VConv(D_2'(-))$.
   Compute $SVM(D_{12})$.

Similarly, agglomerative distributed learning agents can be constructed as follows:

1. Have a learning agent A_1 compute $VConv(D_1(+))$ and $VConv(D_1(-))$.
2. Have a learning agent A_2 compute $VConv(D_2(+))$ and $VConv(D_2(-))$.
3. Have a learning agent A_3 compute
   $D_{12}(+) = VConv(conv(D_1(+)) \cup VConv(D_2(+))$ and
   $D_{12}(-) = VConv(conv(D_1(-)) \cup VConv(D_2(-))$.
4. Generate a training set $D_{12} = D_{12}(+) \cup D_{12}(-)$. Compute $SVM(D_{12})$.

The above approach can be easily generalized to work with an arbitrary number of datasets yielding agents for incremental learning in distrubuted and incremental settings. The proposed algorithms are exact in the sense that the solution found in the incremental and distributed settings are guaranteed to be identical to that obtained in the batch setting for any given training set.

5. EXPERIMENTAL RESULTS

In order to demonstrate the effectiveness of the proposed algorithm we conducted a few experiments on carefully constructed artificial datasets. The purpose of these experiments is to compare the working of the SVM algorithm in the batch setting with that of naive and sophisticated approaches to incremental learning with SVM. The experiments presented used 2-dimensional data to facilitate visualization although the algorithms can work for any finite number of dimensions. In each experiment, we generated two datasets D_1 (figure 6) and D_2 (figure 7) so that the positives examples in D_1 contain boundary information which is important for determining the maximal margin separating hyperplane in the batch setting. This information is lost when a naive approach (suggested in [20]) to incremental or distributed learning is used. The convex-hull based SVM algorithm preserves the necessary boundary information ensuring that the decision boundary found in the distributed and incremental settings is identical to that obtained in the batch setting. The points labeled "+" and "o" denote positive and negative examples respectively. The hyperplane with a slope of -1 in figure 8 corresponds to an incorrect solution and the other hyperplane corresponds to the correct solution (i.e., that found by the SVM algorithm in the batch setting). The data points that lie between the two hyperplanes (lower right corner in figure 8) represent the critical points. By varying the fraction of the number of critical points in the training set, we can demonstrate the degradation of the performance of the naive approach relative to the convex-hull based approach to incremental learning. In the first experiment the fraction of critical points was approximately half that in the second experiment. In each case,

Table 1: Comparison of batch (BSVM), naive distributed (NDSVM), naive incremental (NISVM), convex hull-based incremental (CISVM), and convex-hull based distributed (CDSVM) Learning. $D_1 \to D_2$ in the incremental case denotes the fact that D_1 is learned first followed by D_2. Similarly $D_2 \to D_1$ denotes that D_2 is learned first followed by D_1.

Experiment 1		
Alg.	Train Acc.	Test Acc.
BSVM	100.00	100.00
NDSVM	85.06	75.00
NISVM $1 \to 2$	85.06	75.00
NISVM $2 \to 1$	100.00	100.00
CISVM $1 \to 2$	100.00	100.00
CISVM $2 \to 1$	100.00	100.00
CDSVM	100.00	100.00
Experiment 2		
Alg.	Train Acc.	Test Acc.
BSVM	100.00	100.00
NDSVM	72.32	61.00
NISVM $1 \to 2$	75.32	61.00
NISVM $2 \to 1$	100.00	100.00
CISVM $1 \to 2$	100.00	100.00
CISVM $2 \to 1$	100.00	100.00
CDSVM	100.00	100.00

the test samples were generated from the same distribution as that corresponding to $D_1 \cup D_2$. We used SVM^{light} [13] as an implementation of SVM algorithm and qhull [2] for computing the convex hull.

The results of the experiments are shown in table 1. Note that the SVM algorithm finds the separating hyperplane with 100% training accuracy. It turns out that this accuracy is matched on the test set as well. This is not too surprising because the data set $D_1 \cup D_2$ is linearly separable and the training and test data are drawn from identical distributions. In all experiments, the convex-hull based incremental (CISVM) and distributed (CDSVM) algorithms produced a set of support vectors that was exactly identical to that produced by the SVM in the batch setting (BSVM) thereby verifying the theoretical results presented in section 4. It is worth noting that the naive approach to incremental learning using SVM (NISVM) performs well when the data set D_2 is presented first followed by D_1. This is not surprising since D_1 contains the critical samples. For the same reason, NISVM performs poorly when D_1 is presented first followed by D_2. Furthermore, the performance of NISVM worsens (in terms of both training and test accuracy) as the data distribution is changed so as to increase the relative density of critical points. In conclusion, the experimental results support the claims made in section 4, showing that the convex hull based SVM algorithms yield exact incremental and exact distributed learning.

6. SUMMARY AND DISCUSSION

Many applications of agent-based systems call for the design of learning agents and inter-agent interaction mechanisms that support learning from interaction with open-ended, dynamic environments that include multiple autonomous data and knowledge sources (including other agents). Examples of such domains include distributed information networks for selective information retrieval, information extraction, information fusion, and data-driven collaborative knowledge discovery e.g., in bioinformatics, complex systems monitoring and control applications [12].

Although some incremental and distributed learning have been proposed in the literature, many of them ([7, 17, 18]) 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 batch learning scenario. At present, with the exception of some interesting results (e.g., *mistake bounds*) for the closely related problem of *online learning* ([14, 15, 23, 3]), a characterization of hypothesis classes that admit efficient and scalable designs for learning agents for exact or approximate distributed or incremental learning is lacking. Yet from a practical standpoint, the design and implementation of such agents is clearly of interest.

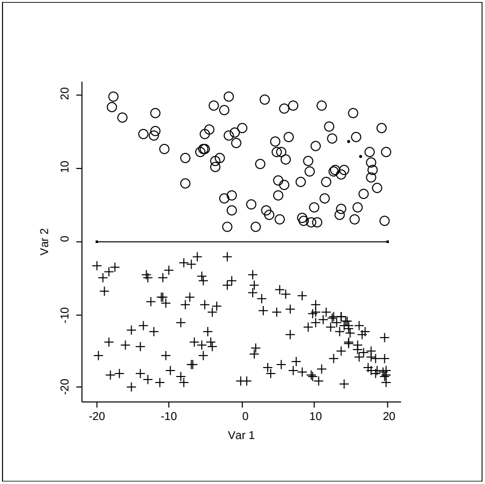

Figure 6: Dataset D_1 in the first experiment

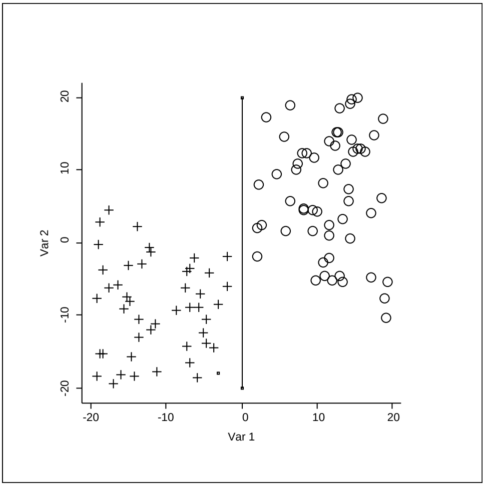

Figure 7: Dataset D_2 in the first experiment

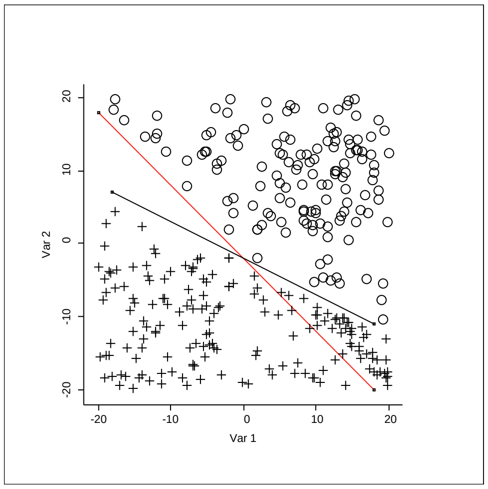

Figure 8: $D = D_1 \cup D_2$ in the first experiment

Against this background, we have explored distributed and incremental learning agents that are variants and extensions of the support vector machine (SVM) family of learning algorithms. In particular, we have shown that a naive SVM-based incremental learning agents (based on an approach first proposed in [20] can produce classifiers that are arbitrarily worse (in terms of generalization) relative to a classifier that is learned by an SVM-based learning agent on the same data set in the batch learning setting. We showed that the instances that constitute the vertices of the convex hulls (respectively) of the positive examples and the negative examples in the data sets are sufficient for exact incremental and exact agglomerative distributed learning.

Our experiments using carefully constructed artificial data sets verify the soundness of this approach. However, since complexity of convex hull computation has a linear dependence on the number of facets of the convex hull (and the number of facets can be exponential in dimension of the space), this approach is likely to be practical only when the convex hulls are simple (i.e., have relatively few facets) [8, 19] A closer examination of the underlying computational task reveals that it is possible to characterize a minimal subset of the training set that satifies the *u-closure* property that provides a sufficient condition for incremental and distributed learning. In fact, such a subset is the minimal set of training instances that determine the space of separating hyperplanes that cannot be ruled out as possible candidates on the basis of examples processed at any stage in the learning process. We call these instances *extended support vectors* (ESV). SV form a subset of ESV and ESV are included in the convex hulls of the samples that belong to the two classes. Efficient algorithms for either exact or approximate identification and incremental update of the set of ESV is a topic of ongoing research.

In this paper, in establishing the sufficient conditions for incremental and distributed learning, we have made the assumption that the output of the learning algorithm is represented essentially in the form of a subset of the training examples. It is possible to relax this assumption by introducing additional classes of learning operators (e.g., hypothesis refinement, hypothesis composition, etc.) that can be used to realize learning agents. 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 analysis of multi-agent learning systems. Long term goals of this research include: design of such 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.

7. REFERENCES

[1] D.W. Aha, D. Kilber, and M.K. Albert. Instance-based learning algorithms. In: *Machine Learning*, 6:37-66, 1991.

[2] C.B. Barber, D.P. Dobkin, and H. Huhdanpaa. The Quickhull Algorithm for Convex Hulls. *ACM Transactions on Mathematical Software*, Vol.22, No.4, 1996.

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

[4] M. Brown, W. Grundy, D. Lin, N. Christianini, C. Sugnet, T. Furey, M. Ares Jr., and D. Haussler. Knowledge Based Analysis of Microarray Gene Expression Data Using Support Vector Machines, Tech. Rep. UCSC CRL-99-09. In: *Computing Research Laboratory*, University of California at Santa Cruz, Santa Cruz, CA, 1999.

[5] D. Caragea, A. Silvescu, and V. Honavar. Distributed and Incremental Learning Using SupportVector Machines. *Tech. Rep. ISU-CS-TR 2000-04. Department of Computer Science, Iowa State University,* Ames, Iowa. March 2000.

[6] C. Cortes, and V. Vapnik. Support Vector Networks. *Machine Learning* 20, 273-297, 1995.

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

[8] H. Edelsbrunner, and E.P. Mucke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13:43-72, 1994.

[9] Y. Freund, and R. Schapire. Large Margin Classification using the Perceptron Algorithm. *Machine Learning*, To appear.

[10] P.M. Gruber, and J.M. Wills. Handbook of Convex Geometry. *Elsevier Science Publishers B.V.*, 1993.

[11] M.A. Hearst, B. Schlkopf, S. Dumais, E. Osuna, and J. Platt. Trends and Controversies - Support Vector Machines. *IEEE Intelligent Systems*, 13(4):18-28, 1998.

[12] V. Honavar, L. Miller, and J. Wong. Distributed Knowledge Networks. In: *Proceedings of the IEEE Conference on Information Technology*, Syracuse, NY, 1998.

[13] T. Joachims. Making Large-Scale SVM Learning Practical. In: *Advances in Kernel Methods-Support Vector Learning*, MIT Press, 1997.

[14] N. Littlestone. Learning when irrelevant attributes abound. *Machine Learning*, 2:285-318, 1988.

[15] N. Littlestone. The weighted majority algorithm. *Information and Computation*, 108:212-261, 1994.

[16] E. Osuna, R. Freund, and F. Girosi. Support Vectors Machines: Training and Applications. *Advances in Kernel Methods-Support Vector Learning*, MIT Press, 1997.

[17] A.L. Prodromidis, and P.K. Chan. 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), 1999.

[18] F. Provost, and D. Hennessy. Scaling Up: Distributed Machine Learning with Cooperation. In: *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 1996.

[19] S.S. Skiena. The Algorithm Design Manual. *Springer-Verlag New York, Inc.*, 1997.

[20] N.A. Syed, H. Liu, and K.K. Sung. Incremental Learning with Support Vector Machines. In: *Proceedings of the KDD Conference*, San Diego, CA, 1999.

[21] S. Thrun, C. Faloutsos, M. Mitchell, and L. Wasserman. Automated Learning and Discovery: State-of-the-art and research topics in a rapidly growing field. *AI Magazine*, 1999.

[22] V. Vapnik. Statistical Learning Theory. *Springer-Verlag, New York*, 1998.

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