Collaborative Multiagent Learning for Classification Tasks

Pragnesh Jay Modi and Wei-Min Shen
Information Sciences Institute and Department of Computer Science
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
{modi, shen}@isi.edu

ABSTRACT

Multiagent learning differs from standard machine learning in that most existing learning methods assume that all knowledge is available locally in a single agent. In multiagent systems, this assumption does not hold because relevant knowledge is distributed among the agents within the system. We describe two decentralized learning algorithms for *distributed classification tasks*, i.e. classification when the attributes are distributed among a set of agents and cannot be gathered into a central agent. Our main contribution is to introduce and formalize the distributed classification task, show that existing classification algorithms are not satisfactory for distributed classification tasks, and finally, to show that our collaborative learning algorithms perform well at distributed classification.

1. INTRODUCTION

Multiagent learning has been defined as "learning that relies on or even requires the interaction among several intelligent agents"[5]. It is a new and exciting field that has recently emerged as we attempt to build complex multiagent systems that operate in dynamic environments. Multiagent systems are extremely difficult to design robustly in advance and thus, we are naturally led to building systems that adapt and learn through experience. However, multiagent learning differs from standard machine learning in that most existing learning methods assume that all relevant knowledge is available locally in a single agent. In multiagent systems, this assumption does not hold because relevant knowledge, such as training experience and background information, is distributed among the agents within the system. Furthermore, domain constraints (such as privacy and cost) may prevent centralization of data.

We formalize our problem by introducing the *distributed classification task*. A distributed classification task is a classification task where the attributes of each instance are distributed among a set of agents. If all information can be gathered within a single agent, the problem can be trivially solved using existing classification methods. However, in many applications, centralization of data is not possible or feasible. For example, local data may be quickly changing, too complex to communicate, too massive to communicate or agents may not be willing to reveal private data even though they are collaborative. It is generally accepted that centralization of all data is undesireable in most multiagent systems. (See for instance [13] and [8]) In this paper, we will assume that centralization of data is not possible. So given a distributed situation, our goal is for the agents to learn in a decentralized manner and as accurately as possible.

We address the distributed classification task by focusing on a special case of multiagent learning: collaborative multiagent learning, i.e. the agents work together as a group to improve their accuracy at a given learning task. Since the agents are collaborative, they are willing to communicate with one another during the learning process. We will show how agents are able to learn accurately while never revealing their local attributes to a central agent. We will present two algorithms, DVS and DDT, for collaborative multiagent learning. DVS is an algorithm that performs a *distributed* specific-to-general search through a conjunctive hypothesis space. DDT is inspired by decision tree algorithms and can be applied to more general distributed classification tasks. We prove the correctness of DVS, while DDT is shown to perform well through empirical evaluation.

It is important to realize that collaborative multiagent learning is fundamentally different from ensemble learning methods such as bagging and boosting [2]. In ensemble learning, the methodology is to combine the predictions of N *independent* learners via voting so as to improve overall accuracy by taking advantage of the variance across learners. This approach will fail when the target concept is not expressible by an individual learner. In contrast, collaborative learning allows a group of agents to learn concepts that no agent can express individually, let alone learn. Collaborative learning for distributed classification tasks is also fundamentally different from previous work on parallel/distributed processing for classification [9]. The primary motivation there is to speed up learning when dealing with massive datasets by dividing the training set onto a set of processors and allowing them to learn concurrently. In contrast, collaborative learning assumes a distributed situation is given and must be dealt with by any learning algorithm. Although efficiency is certainly important, this is not our main goal.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
AGENTS'01, May 28-June 1, 2001, Montréal, Quebec, Canada.
Copyright 2001 ACM 1-58113-326-X/01/0005 ...$5.00.

2. PROBLEM DEFINITION

In an attempt to answer the general question of how agents can learn when relevant knowledge is distributed among them, first we must adopt some method of knowledge representation. One common method of knowledge representation in machine learning is the attribute-value formalism for classification tasks. The input to a classification task consists of a set of pre-classified training examples E, with each example described by (the value of) a vector of attributes $\mathcal{A}$. The goal is to construct a mapping from attribute values to classes. We adopt and modify this knowledge representation scheme for our work because it has two distinct advantages; First, it has been shown that the attribute-value formalism is an extremely general way of representing knowledge and can be applied to a variety of domains; Second, there are wide varieties of existing ML techniques and algorithms that apply to this type of representation. This allows us to take advantage of these existing techniques when devising algorithms for multiagent learning.

We formulate the *distributed classification task* as follows: Given a collection α of n agents, we assume the attribute vector $\mathcal{A}$ is divided into (not necessarily disjoint) subsets, $A_i \subseteq \mathcal{A}$ $(\cup A_i = \mathcal{A}), i \in 1...\text{n}$. Each $agent_i \in \alpha$ knows the classification of every training example, but has access only to A_i from each training example. We will use the notation E_i to denote the projection of training examples E onto A_i. In this way, each agent has only a local, partial view of the training experience. We can immediately see that if the target concept involves attributes that are not allocated to the same agent, no agent can individually learn the concept. We will assume that each training example has a unique id known to all. This is so agents can communicate about individual training examples by id only. This is a reasonable assumption for many domains including those we will describe in the Motivating Domains section. Furthermore, we assume that agents are not able to share the values of their local attributes with other agents. This restriction can arise for a variety of reasons: local data may be too massive to communicate or agents may not be willing to reveal private data even though they are collaborative. The goal of the agents is the same as in the centralized task: to construct a mapping from attribute values to classes.

3. MOTIVATING DOMAINS

Due to the exploratory nature of this work, it is important to show that the problem we have introduced arises in real-world domains. In this section, we describe two applications in which the distributed classification task arises. Furthermore, collaborative learning is useful in these applications because agents are willing to work together to improve their accuracy at the given learning task. However, decentralization of the learning process is necessary because agents are not willing or able to communicate all their local data to others.

3.1 Agentized Organizations

Within a human organization such as a research institution, agent-based technology can facilitate collaborative activities. Each person in the organization is represented by an agent proxy; the agent proxies team together to manage resources, monitor organizational goals and execute actions when goals are threatened [10]. The agents are able to automate routine tasks for the organization, allowing humans to devote their time to more important tasks. This type of system clearly operates in a complex and dynamic environment, one in which learning is a key asset. However, an agent proxy has access to highly personal information about a particular individual. Although this agent is willing to work with other agent proxies to further organizational goals, it is not willing to reveal this personal information.

For example, one type of organizational goal that the agents automate is meeting scheduling. The agent proxies consult their respective humans' schedule to find locally acceptable meeting times and communicate with one another to find globally acceptable meeting times. In distributed classification terms, we can see that each meeting to be scheduled is a training example consisting of attributes relevant to each attendee. All agents know about a particular meeting and can communicate about it via a unique id. The classification to be made is whether the meeting is successfully held or not. A learning algorithm could help the agents predict whether a particular meeting will be cancelled, allowing them to better anticipate the organization's needs. However, the relevant training data is distributed among the proxies and it is private and so cannot be communicated to a single agent. (Would you want your proxy to reveal you miss meetings at 1 pm because you tend to take 2 hr lunches?) A learning algorithm that improves the process of meeting scheduling must be distributed and cannot assume that the data about every human's preferences, work hours, etc. is available in one agent.

3.2 Distributed Sensor Network

Another motivation for decentralized collaborative learning algorithms comes from a distributed sensor domain [1]. This domain consists of stationary 360 degree sensors and targets moving through their sensing range. Each sensor consists of three sectors, each sector is capable of covering 120 degrees. Only one sector of a sensor can be active at a given time. Activation of a particular sector on a sensor at a particular time is under control of an agent. A global *configuration* simply states the sector each agent should activate. The goal for a set of agents, each controlling one sensor, is to choose a configuration so as to cooperatively track all targets. An RF transceiver is used by an agent to communicate with the other agents but bandwidth is very low. Therefore, it is infeasible for agents to communicate large amounts of training experience to a central agent who can then use it to learn.

A tracking episode corresponds to the presence of some set of targets. Each tracking episode is described by an attribute vector, where each attribute corresponds to some sensory data local to a sensor. Therefore, no agent knows the entire attribute vector but the correct configuration depends on the values of the attributes at each agent. Given the entire attribute vector, it is easy to see which global configuration the sensors should adopt. In this way, we label each tracking episode with the correct configuration (or in classification terms, the correct class). Now we can apply a collaborative learning algorithm for distributed classification tasks so that agents can learn to select sector heads in a more coordinated fashion. However since bandwith is low, it is necessary to devise a decentralized algorithm that allows agents to communicate small amounts of information while never having to transfer huge data sets of local experience.

4. ALGORITHMS

We describe two decentralized learning algorithms for the distributed classification task. The first method, DVS, performs a distributed specific-to-general search through a conjunctive hypothesis space to learn boolean concepts. We will prove it to be correct. The second, DDT, is inspired by decision tree algorithms and can be applied to more general classification tasks. In the next section, we will show that it performs well through empirical evaluation.

4.1 DVS

We first simplify the problem by assuming a limited hypothesis space representation and a boolean classification (concept learning). In particular, we consider simple conjunctions of constraints on attribute values. If all the data is centralized, one can use the FIND-S algorithm [6]; starting with the most specific hypothesis, generalize it as needed to cover the positive examples but none of the negative ones. This will determine the most specific hypothesis (MSH) that is consistent with the training data. We will call this hypothesis the *global* MSH. DVS is an algorithm that allows a group of agents to learn boolean concepts from distributed examples using a similar, but decentralized, approach.

The DVS algorithm proceeds in two stages: each agent first learns a *local* MSH; then agents communicate to ensure that the group prediction is consistent with the training examples. We define group prediction as follows: If all the agents' local MSH agree that a new example is a member of the target concept, then the group predicts positive; Otherwise, they predict negative. We will call this the *joint hypothesis*. The reason for defining the joint hypothesis in this way depends crucially on the global hypothesis space representation, which is a simple conjunction of attributes. This has the advantage that it is easily decomposable into local hypotheses. In fact, we will see that each agents' local MSH is the projection of the global MSH onto the set of attributes available to that agent.

In the first learning stage of DVS, each agent performs a local search through its own version space. It begins with the most specific hypothesis (the null hypothesis) and generalizes as necessary to find the (local) MSH consistent with the positive examples. Since an agent does not have access to the entire dataset (remember, it only can see the values for a subset of the attributes), it most likely will not be able to perfectly distinguish between the positive and negative examples. In such cases, the agent is allowed to generalize its local hypothesis to cover the negative example. In this way, the agent searchs for an MSH that covers all the positive examples and as few of the negative examples as possible. During this process, the ids of the false positives are kept in a list. Each agent is guaranteed to form some local MSH since in the worst case, the most general hypothesis ("always predict yes") will suffice.

In the second stage, the agents must communicate to determine that their local MSHs, when taken together, yield a correct joint hypothesis. The joint hypothesis will be incorrect only when all agents predict positive for a training example that is actually negative. To avoid this, they must collectively rule out the false positives. This can be done in an asynchronous, decentralized manner as follows: Let $\mathcal{L} = \{l_1, l_2 ... l_n\}$ be the set of sets containing the example ids of the false positives of each agent's local MSH. It is the task of the agents to determine that no example id exists in every set ($\bigcap_{i=1}^{n} l_i = \oslash$). In other words, they need to determine that the following statement ϕ holds

$$\phi : \forall l_i \in \mathcal{L}, \forall e \in l_i, \exists l_j \in \mathcal{L} \; s.t. \; e \notin l_j$$

Each $agent_i$ broadcasts its l_i to others. A receiving $agent_j$ compares l_i with l_j and replies to $agent_i$ with the list of example ids that are in l_i but not in l_j. $agent_i$ removes those example ids from l_i. All agents perform this procedure asynchronously. Every agent terminates with empty l_i if and only if ϕ is true. If ϕ is false, then the agents are able to say that the target concept is not learnable given a conjunctive hypothesis representation because some example is predicted to be positive by all agents even though it is not a member of the target concept. In such cases, the centralized learner will also fail due to a version space "collapse".

More intuitively, we can see that the agents are performing a specific-to-general distributed search through a joint hypothesis space. Figure 1 shows the local MSH of four agents as hyperplanes through (a 2-d projection of) a m-dimensional feature space. Even though no single agent is able to individually learn the concept, the joint hypothesis, denoted by the dark area in Figure 1, is able to perfectly distinguish the positive and negative training examples. Although a voting mechanism is used to define the joint hypothesis, it is important to realize that the collaboration during learning was more sophisticated than just voting. In particular, the agents have to collaborate to ensure that all (and only) the positive examples are covered by the joint hypothesis. This was done by determining the truth value of statement ϕ. We now show that this algorithm is equivalent to a centralized algorithm that finds the global MSH.

Theorem I: If the target concept is learnable, the joint hypothesis learned by DVS is equivalent to the global MSH.

proof: Let E be a set of preclassified examples with attribute vector $\mathcal{A}$, h_{cent} be the global MSH learned by an agent with access to the entire set E, h_{dvs} be the joint hypothesis learned by a group of agents using DVS, where E is distributed among them in the manner described, and e is a unclassified example that is to be labeled.

Assume h_{dvs} predicts negative for e. We show that h_{cent} must also predict negative. Since h_{dvs} predicts negative, there is some $agent_i$ whose local hypothesis predicts negative. Let $A_i \subseteq \mathcal{A}$ be the set of attributes local to $agent_i$. As $agent_i$ predicted negative, the values in e for the attributes A_i cannot have appeared in any positive example from E. Furthermore, since h_{cent} was also learned from E and is the MSH, it follows that it will also predict negative for e.

Conversely, assume h_{cent} predicts negative for e. Let $a_1 \in \mathcal{A}$ be a attribute constrained by h_{cent} and responsible for the negative classification. (remember, we assumed a conjunctive hypothesis space representation.) Let v_1 be the value of a_1 in e. Then, it must be the case that a_1 never has the value v_1 in any positive example from E. Assuming $a_1 \in A_i$ (which must be true for some i), $agent_i$ must also predict negative for (its partial view of) e. It follows that h_{dvs} will also predict negative. □

The DVS algorithm operates on a easily decomposable, conjunctive hypothesis space and can be used for learning boolean concepts. What happens when the hypothesis space is infinite and not so decomposable? Is decentralized, collaborative learning still possible? We now explore learning for general classification tasks in which the hypothesis space can represent any function of the input attributes. The following algorithm is based on decision tree learners.

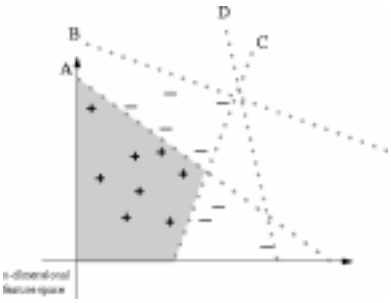

Figure 1: *A m-dimensional feature space partitioned by each agent's local MSH. No individual agent is able to classify the examples alone, but agent A and C together, are able to perfectly classify the training examples.*

4.2 DDT

Decision tree learning is one of the most practical methods for approximating discrete-valued functions. It is robust to noisy data and is capable of learning disjunctive expressions. We give a brief introduction to decision tree learning and refer the reader to [11] for a more detailed explanation. Decision tree learners perform a search through the space of decision trees by recursively choosing an attribute on which to partition the training examples. The "best" attribute on which to partition the examples is judged by the one that provides maximum information gain. One way to define information gain is the reduction in entropy of a set of examples, E, when split on a given attribute a. Entropy is given by

$$Entropy(E) = \sum_{i=1}^{c} -p_i log_2 p_i$$

where p_i is the proportion of E belonging to class i. Information gain, the reduction in entropy, is then given by

$$Gain(E, a) = Entropy(E) - \sum_{v \in values(a)} (|E_v|/|E|) Entropy(E_v)$$

By recursively choosing the attribute with maximum information gain at each stage, the learner performs a greedy search for the best decision tree. We desire a group of agents to perform this search in a distributed, asynchronous manner without having to share their entire local dataset with one another.

The key idea behind the DDT algorithm is to realize that information gain of a particular attribute can be computed by the agent who has access to the values of that attribute; the values of the rest of the attributes are not needed. Furthermore, the information gain measure is a highly compact summarization of each agent's local view of the training set. Agents can use this measure to communicate about their local data in an indirect way and thus perform a distributed search for the best decision tree.

DDT is a lazy learning algorithm in that each agent stores its training data for prediction. There is no training stage and no explicit hypothesis is learned. At prediction time, the agents collectively and asynchronously determine a path through an implicit decision tree. The path is determined using the training data and the attribute values of the test example. The leaf of this path is used to classify the test example. The benefits of lazy decision tree learning are described in [4]. The DDT algorithm is depicted in Figure 2 and is described next.

```
Given:
  - E_i, training examples with attribute vector A_i
  - e, unclassified example to be labeled
  - Q, an empty data structure for holding a sorted
    list of TreePaths

initialize
  a ← attribute with max info gain over E_i
  gain ← info gain of a over E_i
  v ← value of a in e
  [ids] ← list of examples in E_i with value v for a
  insert_in_order(Q, TreePath:([gain], [ids])

when received (TreePath T))
  insert_in_order(Q, T))

procedure make_prediction()
  TreePath:([infogains], [ids]) ← pop(Q)
  a ← attribute with max info gain over [ids]
  gain ← info gain of a over [ids]
  v ← value of a in e
  if gain == 0
    return the most common class in [ids]
  else
    [new_ids] ← list of examples in
                  [ids] with value v for a
    broadcast TreePath:([infogains,gain], [new_ids])
                  to all agents
    make_prediction()
```

Figure 2: *DDT algorithm*

Let a set of training examples E and an unclassified test example e, be given. Each $agent_i \in \alpha$ begins by choosing the local attribute with maximum information gain over E_i. From its local point of view, this is the best choice for the root of the tree. It then partitions E_i on this attribute and then creates a tuple we call a *TreePath*. A TreePath has two fields, a list of real numbers and a set of example ids. Intuitively, it is called a TreePath because it holds the information corresponding to a particular path through some decision tree. Each real number in the first field corresponds to the information gain of the attribute that was used to partition the examples at a particular node along the path. The second field holds the set of example ids at the leaf of this path. For example, suppose $agent_i$ chooses the attribute $a_1 \in A_i$ because it has maximum information gain over E_i, equal to say, 0.24 (see Figure 3). Suppose a_1 takes on the value v_1 in e, and $agent_i$ finds that the examples 1,2,8,9 and 11 in E_i are the ones that have value v_1 for a_1. It then creates the following TreePath: ((0.24) (1,2,8,9,11)). This is a path of depth one, since the examples were separated on only one attribute. Figure 3 shows that as additional attributes are used to further partition the examples, the list of information gains becomes longer and the set of example ids becomes smaller.

Every agent goes through the above process and broadcasts their TreePath to the rest of the agents. As agents receive TreePaths from others, they order them according to a greedy "best info gain first" heuristic. This ordering is shown in Figure 4. The best TreePath received by each agent is then deepened by one level in the same manner as

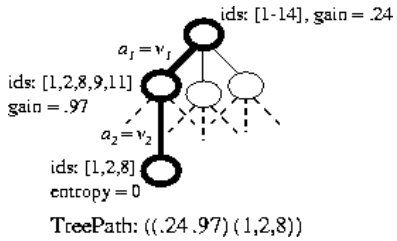

Figure 3: *A path through a decision tree.*

above, except instead of computing information gain over all the examples in E_i, it is computed over the list of example ids in the TreePath. This new TreePath is then broadcast to all agents and the process repeats. The process terminates whenever the best TreePath available to an agent cannot be extended because no attribute available to it provides positive information gain. Since the set of example ids for a TreePath is always getting smaller as the TreePath is extended and no TreePath is ever made shorter, it follows that each agent will eventually terminate. Notice that the termination criterion is specific to an agent's local view. This means that all the agents may not always terminate with the same prediction. Therefore, the agents employ a simple voting procedure to decide the group's prediction. This voting procedure requires some synchronization between the agents. In situations where this synchronization is undesireable, our evaluation shows that simply choosing some agent in advance as the designated predictor has comparable results.

Intuitively, DDT is performing a parallelized, breadth-first search for the best path through some decision tree. This is in contrast to the depth-first greedy strategy of most centralized decision tree learners. However, the inductive bias of DDT is similar in that it prefers shorter paths with higher information gain attributes at the top. We now highlight some of the important properties of this algorithm.

- Agents never share their local attribute values. In fact, agents never even reveal the identities of their local attributes. This satisfies our requirement of decentralization. In addition, this property provides a great deal of flexibility to an agent and the learning system as a whole. For example, an agent can locally, autonomously decide what attributes are relevant for a given learning task.
- DDT is asynchronous. This means that if any agent were to crash or become unavailable or any messages are lost, the group prediction still continues. In other words, the prediction accuracy of the group will degrade gracefully as agents or messages are lost. This is important when agents are operating in complex, dynamic environments where perfect communication cannot be assured. However, agents do need to synchronize at the start of a prediction episode and during voting.

5. EMPIRICAL EVALUATION

We now aim to show that existing classification algorithms are not satisfactory for distributed classification tasks, and

```
//This procedure compares two TreePaths.
//Returns true if T1 is better than T2.
procedure best_info_gain_first(T1,T2)
  gains1 ← list of information gains in T1
  gains2 ← list of information gains in T2
  for i in 1 to min(length of gains1, length of gains2)
    if gains1[i] > gains2[i]
      return TRUE
    else if gains2[i] > gains1[i]
      return FALSE
  if length of gains1 < length of gains2
    return TRUE
  else if length of gains2 < length of gains1
    return FALSE
  else return EQUAL
```

Figure 4: *Heuristic for ordering TreePaths.*

that the DDT algorithm performs well at distributed classification.

Table 1 shows the results of experiments on two UCI domains, **contact-lenses** and **soybean**. We were limited in our choice of domains because our DDT implementation does not yet deal with real-valued attributes, although this is a straightforward extension. **contact-lenses** has four attributes, 24 instances and three classes and **soybean** has 35 attributes, 683 instances and 19 classes. To create a distributed classification task, the attribute vector was evenly divided into four disjoint groups and distributed among four agents. Four fifths of the instances were used for training. Test results on the remaining one fifth are reported. We average the results of three runs.

We compare the accuracy of non-collaborative learning, collaboration via simple voting, collaboration using DDT, and collaboration using DDT plus voting. In non-collaborative learning, the agents used ID3 [11] to learn a decision tree using only their local attributes, A_i. This indicates each agent's accuracy when learning individually. We can see that they perform quite poorly. The column labelled "Vote" shows the group accuracy when the agents learn individually, but are able to collaborate using voting only. Although accuracy is somewhat improved, it is still poor. The section labelled DDT shows the accuracy of the agents when they collaborate. We see that they perform quite well and the performance is quite improved over learning individually. Finally, the column labelled "Vote" shows the group accuracy when the agents learn collaboratively and also collaborate via voting. As shown, the accuracy is 100% and 90% on the **contact-lenses** and **soybean** domain, respectively. In summary, the results justify our theoretical argument that when the target hypothesis involves attributes local to different agents, individual learning will perform poorly. Futhermore, DDT performs well even though agents never communicate their local attributes.

Table 2 shows the average number of messages transmitted during one prediction episode is shown. A message "sent" refers to a particular point-to-point data transfer and every message is broadcast. It is interesting to notice that some agents tend to receive more messages than they send, and in the **soybean** domain, it is correlated with poorer accuracy. We conjecture that these agents tended to terminate

Table 1: Comparison of collaborative vs. non-collaborative learning

*D*ata set	Ag1	Ag2	Ag3	Ag4	Vote
contact					
ID3 %correct	25	75	75	100	
ID3+Vote %correct	–	–	–	–	**75**
DDT %correct	100	100	100	100	
DDT+Vote %correct	–	–	–	–	**100**
soybean					
ID3 %correct	43	64	69	49	
ID3+Vote %correct	–	–	–	–	**76**
DDT %correct	89	91	84	86	
DDT+Vote %correct	–	–	–	–	**90**

Table 2: Amount of messages and data communicated in DDT

*D*ata set	Ag1	Ag2	Ag3	Ag4
contact				
num msgs sent/rcvd	5.75/4.5	5.75/4.0	4.75/4.0	3.8/4.0
bytes sent/rcvd	456/403	537/358	446/361	354/361
soybean				
num msgs sent/rcvd	18/18	18/18	14/18	16/18
Kbytes sent/rcvd	5.5/10.5	12.6/10.5	11/10.5	13/10.5

their search earlier than the others, which would result in their continuing to receive messages, but stop sending them. This would indicate that they had access to the less important attributes of the domain, causing their termination criterion to be satisfied prematurely, and hence resulting in poorer accuracy.

Lastly, we investigate the amount of raw data being communicated. How much data is communicated between agents compared to the size of the entire data set? Table 2 also shows the average amount of data communicated by each agent per prediction over the test set. In the **soybean** domain the amount of data communicated during one prediction is reduced by a factor of 5 compared to the size of the entire data set, which is 200 Kb. In contrast, the size of the entire **contact-lenses** data set is only 1132 bytes. This is so small that the amount of data transmitted between agents is not reduced. However, we also mention that no effort was made to efficiently represent the TreePath data structure. Most of the data is accounted for by long lists of example ids which can be easily compressed by representing consecutive ids as ranges. In summary, we can see that if the number of predictions to be made are few and the size of the data set is large, DDT provides a savings in the amount of raw data communicated by agents.

6. RELATED WORK

Much of the existing work on multiagent learning has focused on reinforcement learning schemes. Most relevant to our work is the subset of these approaches that allow the agents to collaborate during the learning process or view other agents' actions. In these approaches, each agent has a local view of the state. This local perspective problem is solved by either communicating local information (via bidding as in [12]) or learning models of other agents's behavior (as in [3]). However, the work presented here differs in that we assume a supervised learning task whereas these reinforcement learning approaches expect some measure of reward from the environment.

We mention one notable cooperative multiagent learning work that does not use reinforcement learning. [8] discuss the problem of cooperative learning in a multiagent system for parametric design. A set of agents must cooperatively search through a composite search space, where each agent has local access to constraining information. The goal is to find globally acceptable solutions. Unlike the work presented here, they address the local perspective problem by allowing agents to simply reveal their local data to others as necessary when conflicts arise. This non-local information is stored in a case-base by each agent for use during future search episodes. This approach has drawbacks if local information is changing over time or is difficult to communicate to others.

All of the previous work on distributed classification has as its primary motivation to speed up learning when dealing with massive datasets by dividing the training set onto a set of processors and allowing them to learn concurrently. One such example is [9] who investigate distributed learning for classification tasks where the training examples are divided among processors horizontally. Although it has not been our primary motivation, whether our method for dividing the training set in a vertical manner may also be used for dealing with massive datasets is unclear. However, it is certainly true that the processing required by an agent is decreased as the number of attributes available to it is reduced.

Finally, [7] address learning for classification tasks with redundant views. A *view* of a given classification problem consists of an attribute vector from which the target concept can be learned. Two views are *redundant* if they are disjoint and either of them can be used to learn the target concept. They show that when two redundant views of a problem are available, two learners, each operating on one view, can cooperate to improve learning. Our work differs in that we do not assume the agents' views are redundant and we are able to learn concepts that span the two views.

7. CONCLUSION

This paper introduced the distributed classification task for learning in multiagent systems. In this problem, a set of agents must learn to classify training examples when each agent has access to only some of the features. This is an extremely general way of formulating multiagent learning. We describe two algorithms for solving this problem in a decentralized, collaborative manner by a set of agents who never share their local features. We showed that collaborative learning is a useful technique because it allows agents to learn more accurately as a group than any one of them could learn individually.

Acknowledgements

This research is sponsored in part by DARPA/ITO under contract number F30602-99-2-0507, and in part by AFOSR under grant number F49620-01-1-0020.

8. REFERENCES

[1] BAE-Systems. http://www.sanders.com/ants/ecm.htm. In *ECM Challenge Problem*, 2000.

[2] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting and variants. *Machine Learning*, 36:105–142, 1999.

[3] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative mulitagent systems. In *Proceedings of the National Conference on Artificial Intelligence*, 1998.

[4] Jerome H. Friedman, Ron Kohavi, and Yeogirl Yun. Lazy decision trees. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[5] Michael Huhns and Gerhard Weiss. Special issue on multiagent learning. *Machine Learning*, 33:123–128, 1998.

[6] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[7] Ion Muslea, Steve Minton, and Craig A. Knoblock. Selective sampling with redundant views. In *Proceedings of the National Conference on Artificial Intelligence*, 2000.

[8] M V Nagendra Prasad, Susan E. Lander, and Victor R. Lesser. Cooperative learning over composite search spaces: Experiences with a multi-agent design system. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[9] Foster John Provost and Daniel N. Hennessy. Scaling up: Distributed machine learning with cooperation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[10] D.V Pynadath, M. Tambe, Y. Arens, H. Chalupsky, Y. Gil, C. Knoblock, H. Lee, K. Lerman, J. Oh, S. Ramachandran, P. S. Rosenbloom, and T. Russ. Electric elves: Immersing an agent organization in a human organization. In *Proceedings of the AAAI Fall Symposium on Socially Intelligent Agents — the human in the loop*, 2000.

[11] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[12] Gerhard Weiss. Learning to coordinate actions in multi-agent systems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1993.

[13] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10:673–685, 1998.