Classification of Examples by Multiple Agents with Private Features

Pragnesh Jay Modi, Peter Woo Tae Kim
Computer Science Department
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA 15213
pmodi@cs.cmu.edu, pk@andrew.cmu.edu

Abstract

We consider classification tasks where relevant features are distributed among a set of agents and cannot be centralized, for example due to privacy restrictions. We are motivated by a key classification task that arises in a calendar management domain where software assistants classify new meetings as likely to be difficult to schedule. Accurate prediction of the output class is difficult for an isolated single agent because the target concept may involve features to which the agent does not have access, for example each attendee's willingness to attend the meeting. To increase prediction accuracy, novel learning algorithms are required in which agents collaborate to classify new examples while maintaining the privacy of features. We introduce a novel distributed asynchronous decision-tree inspired algorithm for such tasks named DDT. DDT differs from previous approaches in that it applies to vertically partitioned data with categorical multi-valued features, it requires no explicit hypothesis generation, and there is no a priori restriction on number of agents. We present empirical results in our meeting scheduling domain and show that DDT outperforms a single agent learner and performs as well as a centralized learner with hypothetical access to all the features.

1. Introduction

In many multiagent domains where classification tasks arise, agents have private features they are not willing to reveal to other agents or humans. In this paper, we are motivated by one such domain – distributed meeting scheduling by personal assistant agents who act on the behalf of their human users. We assume that an agent who assists a human user in scheduling meetings (or more generally, managing his or her calendar) has access to significant information about the user's personal preferences such as which meetings are important to the user or which other people are important to meet with. In order for such an agent to be an effective assistant, these preferences must be kept private.

In a conventional classification task, accurate prediction of the class of a new unlabeled example requires access to all features that are relevant to its classification. This assumption is inappropriate for classification tasks in distributed multiagent domains where privacy is a concern. A key classification task we consider in the automated calendar management domain is predicting whether a given meeting will be successfully scheduled by a set of agents. The outcome of the distributed scheduling process is either a "success" in which a start time for the meeting was agreed to by all agents, or "failure" in which no start time could be agreed to after some finite amount of effort. The outcome of the process depends on the values of private features of each meeting attendee, such as the attendee's current calendar density, their willingness to bump prior scheduled meetings in favor of the new meeting, and the attendee's personal importance for participating in the meeting. These features are private to each attendee and the correct target concept often involves features belonging to different attendees. No single assistant agent may be able to learn the target concept accurately by itself.

How can a set of agents collaborate to accurately classify a new meeting as likely to be difficult to schedule when the set of input features is distributed among them and must be kept private? This problem can be viewed in a broader context as *privacy-preserving data mining* [1] and has been investigated by several researchers. Existing work can be dichotomized as addressing either *horizontal* or *vertical distribution* of data. In horizontal distribution, an agent knows all the features of a given example, but no agent knows all the examples. In vertical distribution, an agent knows some features of each example, but no agent knows all the features of a given example. We will assume a vertical distribution in this paper. There also exists significant related work in distributed data mining where the primary motivation is efficient learning from a massive training set that is divided among a set of processors or agents. See [7] for an introduction. In this paper, we are mainly concerned with privacy, and while privacy and efficiency may be inter-dependant, they are fundamentally different motivations.

Despite existing research on vertically partioned datasets and where privacy is the key motivation for distribution, accurate classification in this setting remains an open and challenging problem. In particular, previous approaches have several drawbacks that make them seemingly insufficent for the class of problem of concern in this paper. Vaidya and Clifton[9] present an approach that uses computation of scalar product to preserve privacy, but this approach is limited to only two agent interactions. Kargupta et. al. [5] discuss an approach which leverages the Fourier representation of a decision tree, but this approach is limited to binary valued features. Agrawal and Srikant [1] discuss an approach in which numerical values are perturbed to hide information and then later reconstructed to build an accurate classifier, but this approach seemingly does not apply straightforwardly to categorical features. Caragea, Slivescu and Honavar [2] and Vaidya [10] present approaches in which abstract information or counts about private distributed data are used to build an explicit decision tree. While these approaches preserve privacy of values, they require explicit hypothesis generation (e.g., creation of a decision tree) which inherently reveals some information and so is not as private as we desire. The approach we present in this paper attempts to overcome these previous limitations. Our approach applies to vertically partitioned data with categorical multi-valued features, has no a priori restriction on the number of agents, and no requirement of an explicit hypothesis generation.

We introduce a novel distributed classification learning algorithm called DDT. DDT preserves a significant degree of privacy because agents do not reveal their feature values or even reveal the features they use to make predictions. This is accomplished via two main ideas. First, DDT leverages a common theme from previous privacy preserving data mining research, which is to allow agents to communicate high level statistics about their private data rather than communicating the raw data itself [2, 10]. DDT is inspired by centralized decision tree algorithms, specifically the ID3 algorithm[8], in which computing the information gain of a feature is a core operation used to build a decision tree. DDT exploits the fact that information gain of a particular feature can be computed by the agent who has access to the values of that feature; the values of the rest of the features are not needed. Similar to ID3, DDT applies to multi-valued categorical features.

Second, DDT preserves privacy because it is a lazy learning algorithm in which there is no hypothesis generation phase and no explicit decision tree is built [4]. The alternative of generating an explicit global decision tree seemingly implies that an agent who knows the tree must also know something about its features, which is problematic when some features are desired to be kept private. DDT eliminates explicit hypothesis generation and instead all agents engage in prediction at runtime.

DDT is a convenient algorithm for distributed multiagent domains because agents can communicate asynchronously. DDT performs a distributed concurrent search through the space of decision trees by having each agent maintain a list of possible paths through some tree. This list is *dynamically ordered* according to a given inductive bias, such as a preference for shorter trees and higher information gain at the top. Dynamic ordering allows information sent by a slow agent to be taken into account even if it is received very late in the prediction process. This allows for accurate prediction by the agents even when some agents are slow in communicating. Another virtue of DDT is that it is fairly simple to implement given an implementation of a centralized decision tree learner.

Most importantly, we show empirically that DDT is effective in solving a realistic distributed classification task that arises in our domain of concern. We present results in a calendar management domain using the CMRadar simulator [6]. The simulator is populated with a set of agents with a given set of preferences. The agents negotiate to schedule a meeting and log the outcome of the process. Each scheduling episode is a training example. Then as new meetings arise, agents predict if the meeting is likely to be successfully scheduled using the prior episodes as training data. Accuracy of the prediction is verified by engaging in the scheduling process for the new meeting and logging its outcome. We show empirically that DDT outperforms a single agent learner and performs as well as a centralized learner with hypothetical access to all the features.

2. The Distributed Classification Task

The input to a classification task consists of a set of pre-classified training examples E, with each example described by a vector of features $\mathcal{A}$. The goal is to construct a mapping from feature values to classes. We formulate the *distributed classification task* as follows: Given a collection α of n agents, we assume the feature vector $\mathcal{A}$ is divided into (not necessarily disjoint) subsets, $A_i \in \mathcal{A}$ $(\bigcup A_i = \mathcal{A}), i \in 1...n$. 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.

Finally, we assume that agents are not willing to share the values of their local features with other agents. The goal of the agents is the same as in the centralized task: to accurately predict the class of new unseen example. 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 meeting scheduling in which each meeting can be assigned a unique id.

3. Algorithm

3.1. Decision Tree Learning

We give a brief introduction to decision tree learning and refer the reader to [8] for a more detailed explanation. Decision tree learners perform a search through the space of decision trees by recursively choosing a feature on which to partition the training examples. The best feature 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

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

where $E(v)$ is the examples with $a = v$. 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.

3.2. DDT

One key idea behind the DDT algorithm is to realize 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 while maintaining privacy of their features. A second key idea in DDT is to exploit lazy learning to enhance privacy. Each agent stores its training data for prediction so there is no training stage and no explicit hypothesis is learned. The benefits of lazy decision tree learning are described in [4]. In DDT, agents asynchronously communicate to determine a path through an implicit decision tree at prediction time. The path is determined using the training data and the feature values of the test example. The leaf of this path is used to classify the test example. We describe the DDT algorithm in more detail next.

The pseudo-code of DDT is depicted in Figure 1. Let $agent_i$'s local view of the training examples E_i and the unclassified test example e_i, be given. Each $agent_i$ begins by choosing the local attribute with maximum information gain over E_i (line 4). 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 (line 7) and creates a tuple we call a *TreePath* (line 8). A TreePath has two fields, a list of real numbers and a set of example ids. 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 2). Suppose a_1 takes on the value v_1 in e_i, 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.

Every agent goes through the above process and broadcasts its TreePath to the rest of the agents (line 10). As agents receive TreePaths from others, they insert them into an ordered queue according to the ordering specified in Figure 3. The best TreePath received by each agent is then popped off the top of the queue (line 13) and deepened by one level, except instead of computing information gain over all the examples in E_i as before, gain is computed over the list of example ids in the chosen TreePath (line 15). For example, suppose $agent_j$ chooses to extend the TreePath sent to it by $agent_i$ in the first step above. Referring to Figure 2, $agent_j$ chooses attribute $a_2 \in A_j$ because it has maximum information gain over the examples 1,2,8,9 and 11, equal to say, 0.97. Suppose a_2 takes on the value v_2 in e_j, and $agent_j$ finds that the examples 1,2, and 8 are the ones that have value v_2 for a_2. It then creates the following TreePath: ((0.24, 0.97) (1,2,8)). This is a path of depth two. In this way, each agent extends TreePaths made by others using its own features to split the data.

An agent comes to a prediction when the best TreePath available to it cannot be extended because none of its features provides positive information gain (line 17). However, an agent may later revise its prediction if it later receives a better TreePath. All agents terminate when no agent can extend its best TreePath. This can be detected using existing quiescence detection algorithms [3]. We can show that all agents will terminate with the same prediction and this prediction is equivalent to the one that would be made by an agent with access to all the features. For example, consider the special case where DDT execution proceeds in synchronous cycles, i.e., each agent waits until it receives a TreePath from all other agents in each cycle before extending a TreePath in the next cycle. All agents will choose

Given:
(1) E_i, training examples with attribute vector A_i
(2) e_i, unclassified example to be labeled
(3) Q, an empty data structure for holding a sorted list of TreePaths

initialize
(4) $a \leftarrow$ attribute with max info gain over E_i
(5) $gain \leftarrow$ info gain of a over E_i
(6) $v \leftarrow$ value of a in e_i
(7) [ids] $\leftarrow$ list of examples in E_i with value v for a
(8) $T \leftarrow$ **new** TreePath([gain], [ids])
(9) insert_in_order(Q, T)
(10) broadcast T to all agents
(11) **until** termination
(12) make_prediction()

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

procedure **make_prediction**()
(14) TreePath:([gains], [ids]) $\leftarrow$ pop(Q)
(15) $a \leftarrow$ attribute with max info gain over [ids]
(16) $gain \leftarrow$ info gain of a over [ids]
(17) $v \leftarrow$ value of a in e_i
(18) **if** $gain$ == 0
(19) predict the most common class in [ids]
(20) **else**
(21) [new_ids] $\leftarrow$ list of examples in [ids] with value v for a
(22) $T \leftarrow$ **new** TreePath([gains,gain], [new_ids])
(23) insert_in_order(Q, T)
(24) broadcast T to all agents
(25) **endif**

Figure 1. *DDT algorithm*

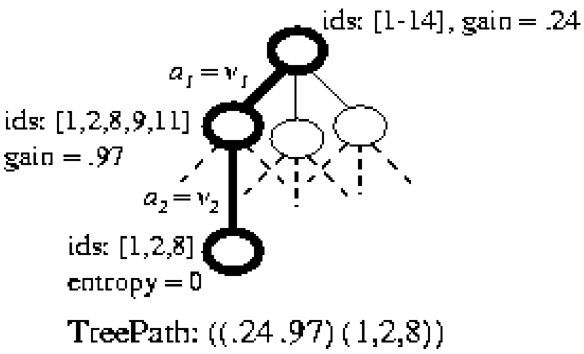

Figure 2. *A path through a decision tree.*

//Procedure for sorting Q from Figure 1.
//Returns true if TreePath $T1$ is better than TreePath $T2$.
procedure **best_info_gain_first**($T1$,$T2$)
  $gains1 \leftarrow$ list of information gains in $T1$
  $gains2 \leftarrow$ 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**
    endif
  endfor
  if length of $gains1 <$ length of $gains2$
    return **TRUE**
  else return **FALSE**

Figure 3. *Heuristic for ordering TreePaths.*

to extend the same TreePath in each cycle thereby eventually coming to the same prediction.

4. Application Domain

4.1. Distributed Meeting Scheduling

Our development of the DDT algorithm is motivated by the distributed meeting scheduling problem. We evaluate DDT in the context of the CMRadar Project [6] whose goal is to develop personalized assistant agents that automate many routine everyday tasks such as scheduling of meetings. Importantly, CMRadar is developed as an agent that also interacts with other users or agents. Maintaining privacy during such interactions is an important consideration.

We have built a simulator testbed to enable development of learning and scheduling techniques. The CMRadar agents live in a simulated distributed environment and are able to pass simulated email messages between them. Each agent manages its own calendar schedule and negotiates with the other agents for meeting times. Each training example is generated by one run of the simulator in which agents employ a distributed meeting scheduling protocol, described next, to schedule a new meeting M_{m+1}.

The distributed meeting scheduling protocol employed by CMRadar agents proceeds in a sequence of rounds. In each round, an initiator sends a message to the other agents with a proposed start time T for a new meeting M_{m+1}. Based on their respective user's preferences, each agent accepts or rejects the proposal. If any agent rejects the proposal, the initiator begins a new round by proposing another start time. The protocol continues in rounds until an agreement is reached by all agents (success) or the initiator has no more values to propose or a max number of rounds elapses (failure). Times cannot be double-booked, so if an existing meeting M has been previously scheduled at time

T and M_{m+1} also becomes scheduled at time T, M must be bumped and rescheduled for another time in order for the epsiode to be classified as a success.

Each agent decides whether to accept or reject a proposal from the initiator using a decision procedure we call a *scheduling strategy*. The idea is that this strategy reflects the human user's preferences about which meetings are important. For simplicity in the following, we will call them agent preferences.

4.2. Features

Each agent's scheduling strategy is based on the values of the following set of private features.

- Schedule Density (**SD**$_i$): This feature represents the current density of $agent_i$'s schedule. It has values *low* (less than 40%) , *medium* (between 40% and 70%), and *high* (greater than 70%).
- Attendee Importance (**AI**$_{ij}$): For a given meeting M_{m+1}, **AI**$_{ij}$ represents $agent_i$'s preference for meeting with $agent_j$. Attendee importance has values *low*, *medium*, *high*, and *notPresent*. The *notPresent* value is a default value for the feature for when $agent_j$ is not an attendee of meeting M_{m+1}.
- Subject Importance (**SI**$_i$): This feature represents $agent_i$'s preference level for the subject or topic of a particular meeting. It has values *very high*, *high*, *medium*, *low*, and *very low*.

4.3. Scheduling Strategies

For our experiments, we populate the simulator with a set of agents that employ scheduling strategies that intuitively reflect plausible user preferences. These may not reflect everyone's intuitions, but our main goal is to simulate non-trivial strategies to demonstrate the viability of our learning approach.

- **Preference 1**: Always reject every proposal.

- **Preference 2**: Always accept every proposal.

- **Preference 3**: If **SI** = *very low* for M_{m+1}, reject the initiator's proposal. Else if the proposed time T is free in the schedule, accept the initiator's proposal. Else (T is occupied by meeting M), apply one of the following rules:

- - **Preference 3a**: Accept the initiator's proposal if the maximum **AI** over all attendees in proposed meeting M_{m+1} is higher than the maximum **AI** in M. Otherwise, reject the proposal.

- - **Preference 3b**: Accept the initiator's proposal if **SI** of new meeting M_{m+1} is higher than **SI** of M. Otherwise, reject the proposal.

In our experiments described in the next section, the agents engage in scheduling episodes where each agent uses one of the four preference rules above (1, 2, 3a, 3b) to make its decisions. All agents log the values of their own features and the outcome of the scheduling episode (success or failure). Clearly, the outcome of any one episode is the result of a complex interplay between who the meeting attendees are, what their strategies are, and what the values of their features are, all of which makes for a non-trivial prediction task.

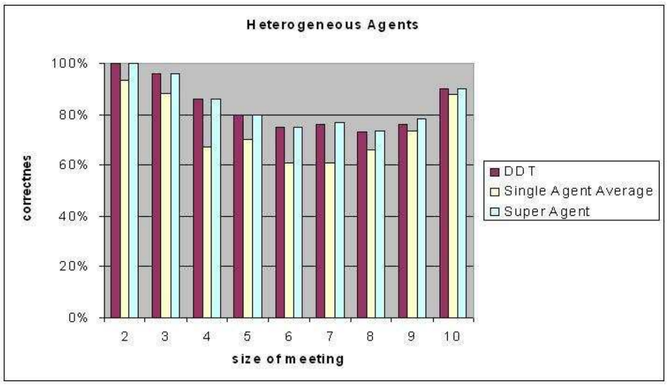

Figure 4. *DDT outperforms the single agent learner and performs as well as an agent with hypothetical access to all the features ("super" agent).*

5. Experimental evaluation

5.1. Setup

We populate the CMRadar simulator with 10 agents with calendars filled to a given density with pre-existing meetings. Each agent's calendar has 50 timeslots to simulate a 5 day, 10 hour per day work week. The number of attendees for each existing meeting is chosen according to a distribution in which meetings of more people are less likely than meetings with fewer people, and every meeting has at least two attendees. The attendees of the new meeting M_{m+1} to be scheduled are chosen to be a random subset of the agents, with the size of the meeting as an input parameter.

The CMRadar simulator executes the scheduling of a meeting M_{m+1} and each agent logs the values of its own features. As described earlier, each agent has access to features that describe its users current calendar and preferences. In each negotiation, each $agent_i$ logs 11 features: **SI**_i, **SD**_i, and 9 features for attendee importance (**AI**), one for each of the other agents in the system. This makes a system-wide total of 110 input features. Each agent also logs the outcome of the process (success or failure).

For empirical evaluation, the performance of DDT is compared against the following approaches.

- **Single Agent Average**: We apply the standard ID3 decision tree learner to a single agent's features. The accuracy on a test set, averaged over all the agents, is used to measure total correctness.

- **Super Agent**: One agent is given access to the combined features of all the agents and it uses ID3 to make a prediction. That is, the private features of $agent_1$, $agent_2$, $agent_3$, etc are combined into one grand dataset and a centralized decision tree learner is used. This is not a viable solution approach because it violates privacy restrictions, but is done strictly for evaluation purposes.

5.2. Empirical Results

Our results are obtained on 100 training examples and 60 test examples. Correctness is measured as the percent of correctly predicted test examples.

5.2.1. Heterogeneous Agents Figure 4 shows the prediction accuracy from a set of agents using different scheduling strategies. The strategy of each agent is fixed but selected randomly from those described in section 4.3. On the x-axis is the size of M_{m+1} which is the number of attendees in the new meeting, increased from 2 to 10.

These results show that DDT outperforms the single agent learner, and performs on par as the super agent learner. We see that when there are only two agents in the meeting all approaches are fairly accurate because the meetings are easy to schedule and so almost all examples are labeled success. However as more agents are involved in the meeting, we see a bigger difference in performance. In particular, DDT outperforms the single agent learners. Finally, as size of M_{m+1} increases to over 7, the accuracy of all approaches increases again. The reason is that large meetings are exceedingly difficult to schedule and so most examples can be safely predicted to be failures.

5.2.2. Effect of Schedule Density In the above experiment, agent calendars were of varying density. In this experiment, the densities are uniform. The purpose is to determine the effect of schedule density on prediction accuracy. Figure 5 and 6 shows the results where every agent has either *low* or *high* schedule density, respectively. In the *low* case, DDT shows overall higher accuracy than the single agent approach as before. However, in the *high* case there is less of a difference. The reason is that when schedule densities are uniformly high, most meetings are difficult to schedule regardless of the other feature values so the prediction task is fairly easy even for a single agent.

5.2.3. Effect of Subject Importance and Attendee Importance When all agents have similar preferences, we hypothesize that a single agent may be able to predict quite accurately. This is because all agent's feature values are highly correlated and so explicit knowledge of other agent's feature values may not be necessary. We test this hypothesis by varying the agent's values of the Subject Importance **SI** feature and the Attendee Importance **AI** features. We run three conditions: a) Varying feature values for every agent, b) homogeneous **SI** values for every agent but heterogenous **AI** values, and c) homogenous **AI** values but heterogenous **SI** values. The result is shown in Figure 7 with the three test conditions on the x-axis. The result shows that when agents have wide disagreement on which meetings subjects are of importance (condition (c), far right), learning to predict which meetings are schedulable is difficult in isolation, but a distributed learning algorithm like DDT is effective.

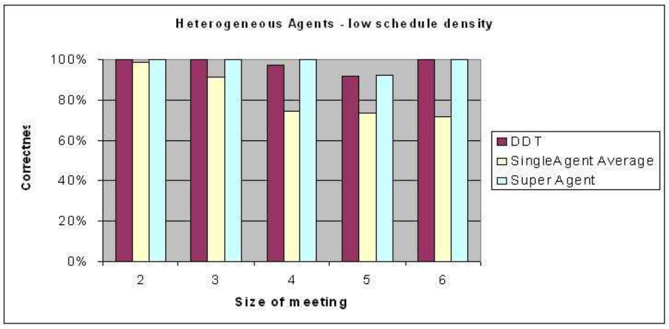

Figure 5. *DDT outperforms the single agent learner when initial schedules are of low density.*

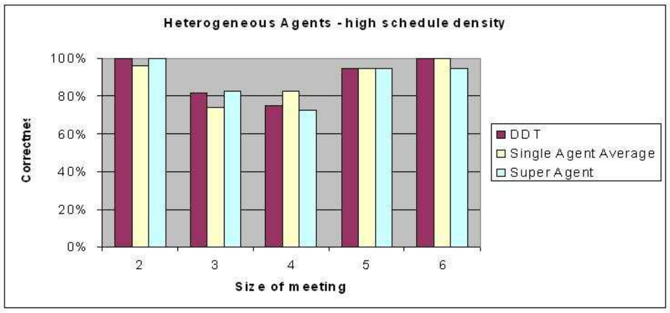

Figure 6. *Performance of DDT relative to the single agent learner is not significant when initial schedules are of high density. This is because most scheduling episodes can be correctly predicted to end in failure.*

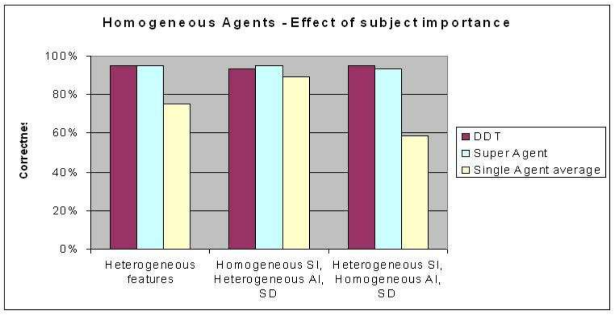

Figure 7. *Evaluating the effect of homogenous values for the features Subject Importance (**SI**) and Attendee Importance (**AI**).*

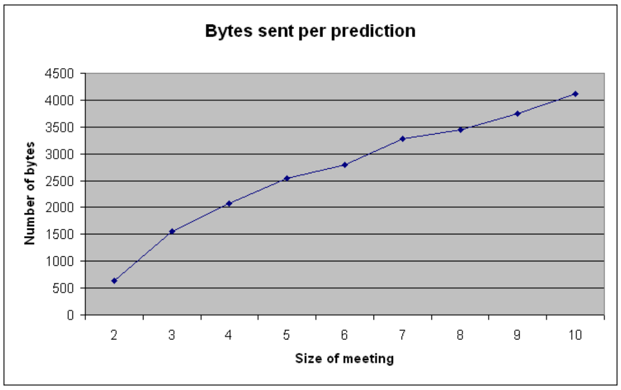

Figure 8. *Communication overhead in DDT.*

When all agents agree on meeting subject importance, the difference is not as great (condition (b), middle).

5.2.4. Communication Overhead Figure 8 shows the total number of bytes communicated by all agents in order to make a prediction. As the size of M_{m+1} increases, the communication overhead increases because there are more agents involved in the prediction. Because each message is broadcast to all other agents, we expect DDT to have high communication overhead with high number of agents. We can conclude however that DDT's communication overhead is quite reasonable for limited numbers of agents– it requires less than 5000 bytes for prediction with 10 agents.

6. Conclusion

We described the DDT algorithm which uses lazy learning and communication of dynamically ordered lists of information gain to maintain privacy of features. Its performance in the domain of distributed meetings scheduling was shown to outperform a single agent learner and a superagent with hypothetical access to all the features. We conclude that DDT is useful algorithm for our domain of interest because it allows agents to keep their private features and predict more accurately as group than any one of them could individually.

Acknowledgements

We thank Wei-Min Shen for collaboration on an earlier version of this work and Manuela Veloso for her insightful comments.

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA or the Department of Interior-National Business Center (DOI-NBC).

References

[1] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 439–450. ACM Press, May 2000.

[2] D. Caragea, A. Silvescu, and V. Honavar. Decision tree induction from distributed, heterogeneous, autonomous data sources. In *Proceedings of the Conference on Intelligent Systems Design and Applications (ISDA 03)*, 2003.

[3] K. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 1985.

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

[5] H. Kargupta, B. Park, D. Hershbereger, and E. Johnson. Collective data mining: A new perspective toward distributed data mining. In *Advances in Distributed and Parallel Knowledge Discovery*. AAAI/MIT Press, 1999.

[6] P. J. Modi, M. Veloso, S. Smith, and J. Oh. Cmradar: A personal assistant agent for calendar management. In *Agent Oriented Information Systems, (AOIS) 2004*, 2004.

[7] F. J. Provost and D. N. Hennessy. Scaling up: Distributed machine learning with cooperation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

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

[9] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *The Eighth International Conference on Knowledge Discovery and Data Mining*, 2002.

[10] J. S. Vaidya. *Privacy Preserving Data Mining over Vertically Partitioned Data (Ch 3.5)*. PhD thesis, Purdue University, 2004.