Communication Efficient Construction of Decision Trees Over Heterogeneously Distributed Data

Chris Giannella Kun Liu Todd Olsen
Hillol Kargupta
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County, Baltimore, MD 21250 USA
{cgiannel,kunliu1,tolsen1,hillol}@cs.umbc.edu
(H. Kargupta is also affiliated with AGNIK, LLC, USA.)

Abstract

We present an algorithm designed to efficiently construct a decision tree over heterogeneously distributed data without centralizing. We compare our algorithm against a standard centralized decision tree implementation in terms of accuracy as well as the communication complexity. Our experimental results show that by using only 20% *of the communication cost necessary to centralize the data we can achieve trees with accuracy at least* 80% *of the trees produced by the centralized version.*

Key words: Decision Trees, Distributed Data Mining, Random Projection

1 Introduction

Much of the world's data is distributed over a multitude of systems connected by communications channels of varying capacity. In such an environment, efficient use of available communications resources can be very important for practical data mining algorithms. In this paper, we introduce an algorithm for constructing decision trees in a distributed environment where communications resources are limited and efficient use of the available resources is needed. At the heart of this approach is the use of random projections to estimate the dot product between two binary vectors and some message optimization techniques. Before defining the problem and discussing our approach, we briefly discuss distributed data mining to provide context.

1.1 Distributed Data Mining (DDM)

Overview: Bluntly put, DDM is data mining where the data and computation are spread over many independent sites. For some applications, the distributed setting is more natural than the centralized one because the data is inherently distributed. The bulk of DDM methods in the literature operate over an abstract architecture where each site has a private memory containing its own portion of the data. The sites can operate independently and communicate by message passing over an asynchronous network. Typically communication is a bottleneck. Since communication is assumed to be carried out exclusively by message passing, a primary goal of many methods in the literature is to minimize the number of messages sent. Similarly, our goal is to minimize the number of messages sent. For more information about DDM, the reader is referred to two recent surveys [8], [10]. These provide a broad overview of DDM touching on issues such as: association rule mining, clustering, basic statistics computation, Bayesian network learning, classification, and the historical roots of DDM.

Data format: It is commonly assumed in the DDM literature that each site stores its data in tables. Due to the ubiquitous nature of relational databases, this assumption covers a lot of ground. One of two additional assumptions are commonly made regarding how the data is distributed across sites: homogeneously (horizontally partitioned) or heterogeneously (vertically partitioned). Both assumptions adopt the conceptual viewpoint that the tables at each site are partitions of a single global table.[1] In the homogeneous case, the global table is horizontally partitioned. The tables at each site are subsets of the global table; they have exactly the same attributes. In the heterogeneous case, the table is vertically partitioned, each site contains a collection of columns (sites do not have the same attributes). However, each tuple at each site is assumed to contain a unique identifier to facilitate matching across sites (matched tuples contain the same identifier).

[1] It is not assumed that the global table has been or ever was physically realized.

Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM'04)
0-7695-2142-8/04 $ 20.00 IEEE

Note that the definition of "heterogeneous" in our paper differs from that used in other research fields such as the Semantic Web and Data Integration. In particular we are not addressing the problem of schema matching.

1.2 Problem Definition and Results Summary

We consider the problem of building a decision tree over heterogeneously distributed data. We assume that each site has the same number of tuples (records) and they are ordered to facilitate matching, *i.e.*, the i^{th} tuple on each site matches. This assumption is equivalent to the commonly made assumptions regarding heterogeneously distributed data described earlier. We also assume that the i^{th} tuple on each site has the same class label. Our approach can be applied to an arbitrary number of sites, but for simplicity, we restrict ourselves to the case of only two parties: Adam and Betty. However, in section 4.3 we describe the communication complexity for an arbitrary number of sites. At the end, Adam and Betty are to each have the decision tree in its entirety. Our primary objective is to minimize the number of messages transmitted.

One way to solve this problem is to transmit all of the data from Adam's site to Betty. She then applies a standard centralized decision tree builder and finally, transmits the final tree back to Adam. We call this method the *centralized approach (CA)*. While straightforward, the CA may require excessive communication in low communication bandwidth environments. To address this problem, we have adapted a standard decision tree building algorithm to the heterogeneous environment. The main problem in doing so is computing the information gain offered by attributes in making splitting decisions. To reduce communication, we approximate information gain using a random projection based technique. The technique converges on the correct information gain as the number of messages transmitted increases. We call this approach to building a decision tree the *distributed approach (DA)*.

The tree produced by DA may not be the same as that produced by CA. However, by increasing the number of messages transmitted, the DA tree can be made arbitrarily close. We conducted several experiments to measure the trade-off between accuracy and communication. Specifically, we built a tree using CA (with the standard Weka tree builder implementation) and others using DA while varying the number of messages used in information gain approximation and the depth of the tree. We observed that by using only 20% of the communication cost necessary to centralize the data we can achieve trees with accuracy at least 80% of the CA. Henceforth, when we discuss communication cost or communication complexity, we mean the total number of messages required. A message is a four byte number *e.g.* a standard floating point number.

1.3 Paper Layout

In Section 2 we cite some related work. In Section 3 we describe the basic algorithm for building a decision tree over heterogeneously distributed data using a distributed dot product as the primary distributed operation. Then we propose a method for approximating a distributed dot product using a random projection. In Section 4 we describe the complete algorithm and give the communication complexity. In Section 5 we discuss how different message optimization techniques are employed to further reduce the communication. In Sections 6 we present the results of our experiments. Finally, in Section 7 we describe several directions for future work and conclusions.

2 Related Work

Most algorithms for learning from homogeneously distributed data (horizontally partitioned) are directly related to ensemble learning [9, 3], meta-learning [12] and rule-based [5] combination techniques. In the heterogeneous case, each site observes only partial attributes (features) of the data set. Traditional ensemble-based approaches usually generate high variance local models and fail to detect the interaction between features observed at different sites. This makes the problem fundamentally challenging. The work addressed in [11] develops a framework to learn decision tree from heterogeneous data using a scalable evolutionary technique. In order to detect global patterns, they first make use of boosting technique to identify a subset of the data that none of the local classifiers can classify with high confidence. This subset of the data is merged at the central site and another new classifier is constructed from it. When a combination of local classifiers cannot classify a new record with a high confidence, the central classifier is used instead. This approach exhibits a better accuracy than a simple aggregation of the models. However, its performance is sensitive to the confidence threshold. Furthermore, to reduce the complexity of the models, this algorithm applies a Fourier Spectrum-based technique to aggregate all the local and central classifiers. However, the cost of computing the Fourier Coefficient grows exponentially with the number of attributes. On the other hand, our algorithm generates a single decision tree for all the data sites and does not need to aggregate at all. The work in [2] presents a general strategy of distributed decision tree learning by exchanging among different sites the indices and counts of the records that satisfy specified constraints on the values of particular attributes. The resulting algorithm is provably exact compared with the decision tree constructed on the centralized data. The communication complexity is given by $O((M + |L|NV)ST)$ where M is the total number of records, $|L|$ is the number of classes, N is the total number

of attributes, V is the maximum number of possible values per attribute, S is the number of sites and T is the number of nodes of the tree. However, instead of repeatedly sending the whole indices vectors to the other site, our algorithm applies a random projection-based strategy to compute distributed dot product as the building blocks of tree induction. This kind of dimension reduction technique, together with some other message reusing and message sharing schemas reduce as many unnecessary messages as possible. The number of messages for one dot product is bounded by $O(k)$ $(k << M)$, and the total communication cost of our algorithm is $O((LT + kIT)(S - 1))$ (LT is the number of leaf node and IT is the number of non-leaf node), which is less than that in [2]. The work presented in [4] deals with a privacy preserving two-party decision tree learning problem where no party is willing to divulge their data to the other. The basic tree induction procedure is similar with ours. However, a *secure* dot product protocol is proposed here as the building block such that only the information gain of the testing attribute is disclosed to both parties and nothing else. The communication complexity of only one dot product protocol is $O(4M)$, the total communication cost is higher than ours.

3 Building a Distributed Decision Tree: the Basic Algorithm

For simplicity of exposition, we only discuss discrete data and assume that each node of the tree has a corresponding attribute and a child branch for each distinct value. Our algorithm, however, generalizes to other cases (*e.g. continuous attributes*) without any conceptual difficulties.

3.1 Notation

Both sites have M tuples ordered in such a way that tuple i on Adam's site corresponds to tuple i on Betty's site. Tuples on both sites have an associated class label drawn from a set L. The tuples are labeled consistently across sites *i.e.* the i^{th} tuple on Adam and Betty's site has the same class label. Let N denote the total number of attributes from all sites.

Let $\mathcal{A}$ denote the union of attributes over both sites and $\mathbb{D}$ denote the data set formed by joining the data from both sites (Adam's i^{th} tuple is concatenated with Betty's to form the i^{th} tuple in $\mathbb{D}$). Given attribute $A \in \mathcal{A}$, let $\Pi(A)$ denote the set of distinct values that appear in the A column. Given set of attributes $X \subseteq \mathcal{A}$ and list of values $\vec{x} \in \times_{A \in X} \Pi(A)$, let $\mathbb{D}(X = \vec{x})$ denote the set of tuples t in $\mathbb{D}$ such that the X columns of t agree with $\vec{x}$ *i.e.* for all $A \in X$, $t[A] = \vec{x}[A]$.

Given $\hat{\mathbb{D}} \subseteq \mathbb{D}$, attribute $A \in \mathcal{A}$ and value $a \in \Pi(A)$, let $\#_{A=a}(\hat{\mathbb{D}})$ denote the number of tuples t in $\hat{\mathbb{D}}$ such that $t[A] = a$. Given class label $\ell \in L$, let $\#_{\ell}(\hat{\mathbb{D}})$ denote the number of tuples in $\hat{\mathbb{D}}$ with label ℓ. Let $\#_{\ell, A=a}(\hat{\mathbb{D}})$ denote the number of tuples t in $\hat{\mathbb{D}}$ with $t[A] = a$ and label ℓ. The *class entropy* of A over $\hat{\mathbb{D}}$ is denoted $E_A(\hat{\mathbb{D}})$ and defined as[2]

$$-\sum_{a \in \Pi(A)} \frac{\#_{A=a}(\hat{\mathbb{D}})}{|\hat{\mathbb{D}}|} \sum_{\ell \in L} \frac{\#_{\ell, A=a}(\hat{\mathbb{D}})}{|\hat{\mathbb{D}}|} log_2(\frac{\#_{\ell, A=a}(\hat{\mathbb{D}})}{|\hat{\mathbb{D}}|}).$$

The *information gain* of A over $\hat{\mathbb{D}}$ is denoted $G_A(\hat{\mathbb{D}})$ and defined as

$$-\sum_{\ell \in L} \frac{\#_{\ell}(\hat{\mathbb{D}})}{|\hat{\mathbb{D}}|} log_2(\frac{\#_{\ell}(\hat{\mathbb{D}})}{|\hat{\mathbb{D}}|}) - E_A(\hat{\mathbb{D}}).$$

Our distributed decision tree building approach can be applied without change to other forms of information gain such as the Gini index. For ease of discussion, we stick with entropy based information gain.

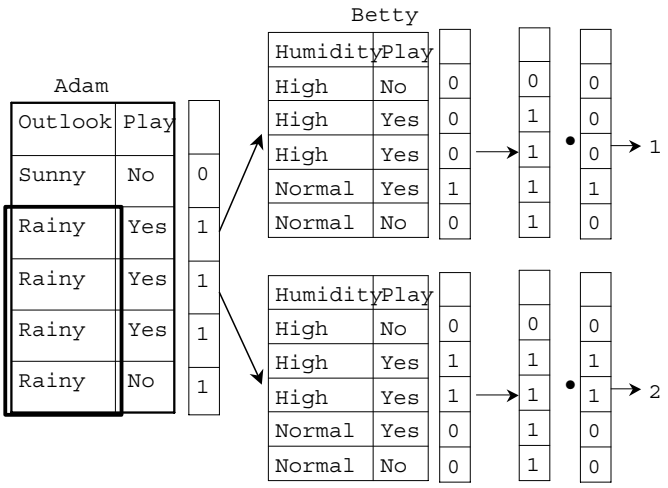

Figure 1. Calculating information gain using the dot product. ("Play" is the class name, and $\cdot$ denotes the dot product.)

3.2 Distributed Decision Tree Building Using a Dot Product

We adapt the following version of the standard, depth-first decision tree building algorithm (on discrete data). Initially the tree is empty and the first call is made to determine the root node. The call chooses the attribute A_1 from $\mathcal{A}$ with the largest information gain over $\mathbb{D}$ to become the root. For each $a \in \Pi(A_1)$, a recursive call is made with list $\{(A_1, a)\}$. Each of these recursive calls will determine the children of the root (with branches labeled with the values in $\Pi(A_1)$).

At any call passed list (A_1, a_1), ... (A_k, a_k), the tuples in $\mathbb{D}(X = \vec{x})$ where $X = \{A_1, \ldots, A_k\}$ and $\vec{x} = (a_1, \ldots, a_k)$ are examined to determine the next splitting attribute. The attribute from $\mathcal{A} - X$ with the largest information gain over $\mathbb{D}(X = \vec{x})$ is chosen.

[2] We assume $0 log_2(0)$ equals zero.

Since the attributes are not all on one site, computing the information gain may not be possible. For example, assume at least one of the attributes from X were on Adam's site and consider D an attribute on Betty's site and not in X. To compute the information gain, Betty must compute $\#_\ell(\mathbb{D}(X = \vec{x}))$ and $\#_{\ell,D=d}(\mathbb{D}(X = \vec{x}))$ for all $d \in \Pi(D)$ and $\ell \in L$. These values cannot be computed directly since Betty does not have $\mathbb{D}(X = \vec{x})$. Adam must send Betty information to carry out this computation. To reduce the amount of messages we approximate the values using a technique based on random projections.

Each of the values can be modeled as a dot product computation (similar to [2] and [4]). Let X_A denote the attributes from X on Adam's site and $\vec{x}_A$ their associated values from the passed list; likewise define X_B and $\vec{x}_B$. Let $\vec{V}(X_A = \vec{x}_A)$ be a length M vector of zeros and ones. The i^{th} entry is one if the i^{th} tuple t_i in $\mathbb{D}$ satisfies $t_i[X_A] = \vec{x}_A$. All other entries are zero. Likewise, let $\vec{V}(D = d, X_B = \vec{x}_B, \ell)$ be the 0/1 vector whose i^{th} entry is one if $t_i[D] = d$, $t_i[X_B] = \vec{x}_B$ and t_i has label ℓ. It can be easily seen that the dot product of $\vec{V}(X_A = \vec{x}_A)$ and $\vec{V}(D = d, X_B = \vec{x}_B, \ell)$ equals $\#_{\ell,D=d}(\mathbb{D}(X = \vec{x}))$. Moreover, the dot product of $\vec{V}(X_A = \vec{x}_A)$ and $\vec{V}(X_b = \vec{x}_B, \ell)$ equals $\#_\ell(\mathbb{D}(X = \vec{x}))$. Figure 1 illustrates this concept. Adam sends Betty a binary vector representing the tuples with "Outlook = Rainy". Betty constructs two vectors representing "Humidity = Normal && Play = Yes" and "Humidity = High && Play = Yes", respectively. The dot products gives the number of tuples in the whole database that satisfy the constrains "Outlook = Rainy && Humidity = Normal && Play = Yes" and "Outlook = Rainy && Humidity = High && Play = Yes". Note that the notation above deals with the case where Betty computes the information gain of her attributes. However, our algorithm will also require the reverse case: Adam computes the information gain of all his attributes. The notation is analogous. Actually, in our algorithm, instead of sending the original binary vectors directly to the other site, we project the vectors into a lower dimensional space first and transmitting the new vectors to all other sites. This leads to the distributed dot product computation in the next section.

3.3 Distributed Dot Product

In the previous section, we observed that distributed dot product of boolean vectors is the building block of decision tree induction. In this section, we propose a random projection-based distributed dot product technique that can greatly reduce the dimensionality of the vector, thereby reducing the cost of building the tree. Similar form of this algorithm appears elsewhere in a different context [7].

Given vectors $\vec{a} = (a_1, \ldots, a_m)^T$ and $\vec{b} = (b_1, \ldots, b_m)^T$ at two distributed site A and B, respectively, we want to approximate $\vec{a}^T\vec{b}$ using a small number of messages between A and B. Algorithm 3.3.1 gives the detailed procedure.

Algorithm 3.3.1 Distributed Dot Product Algorithm($\vec{a}, \vec{b}$)

1. A sends B a random number generator seed. **[1 message]**
2. A and B cooperatively generate $k \times m$ random matrix R where $k \ll m$. Each entry is generated independently and identically from any distribution with zero mean and unit variance. A and B compute $\hat{a} = R\vec{a}$, $\hat{b} = R\vec{b}$, respectively.
3. A sends $\hat{a}$ to B. B computes $\hat{a}^T\hat{b} = \vec{a}^T R^T R\vec{b}$. **[k messages]**
4. B computes $D = \frac{\hat{a}^T\hat{b}}{k}$.

So instead of sending a m-dimensional vector to the other site, we only need to send a k-dimensional vector where $k \ll m$ and the dot product can still be estimated.

The above algorithm is based on the following fact:

Lemma 3.1 *Let R be a $p \times q$ dimensional random matrix such that each entry $r_{i,j}$ of R is independently and chosen from some distribution with zero mean and unit variance. Then,*

$$E[R^T R] = pI, \text{ and } E[RR^T] = qI.$$

Proof Sketch: The (i, j) entry of $R^T R$ is the dot product of the i^{th} and j^{th} columns of R. If $i = j$, then the expected value of the dot product equals the p times the variance plus the square of the mean, hence, p. If $i \neq j$, then the expected value of the dot product equals p times the square of the mean, hence zero. The second part of the lemma is proven analogously. □

Intuitively, this result echoes the observation made elsewhere [6] that in a high-dimensional space vectors with random directions are almost orthogonal. A similar result was proved elsewhere [1].

3.4 Accuracy Analysis

We give a Chernoff-like bound to quantify the accuracy of our distributed dot product for decision tree induction as follows:

Lemma 3.2 *Let $\vec{a}$ and $\vec{b}$ be any two boolean vectors. Let $\hat{a}$ and $\hat{b}$ be the projections of $\vec{a}$ and $\vec{b}$ to $\Re^k$ through a random matrix R whose entries are identically, independently chosen from N(0,1) such that $\hat{a} = R\vec{a}$ and $\hat{b} = R\vec{b}$, then for any $\epsilon > 0$, we have*

$$Pr\{\vec{a}^T\vec{b} - \epsilon m \leq \frac{\hat{a}^T\hat{b}}{k} \leq \vec{a}^T\vec{b} + \epsilon m\} \geq 1 - 3\left(\left((1+\epsilon)e^{-\epsilon}\right)^{\frac{k}{2}} + \left((1-\epsilon)e^{\epsilon}\right)^{\frac{k}{2}}\right)$$

k	Mean	Var	Min	Max
100(1%)	0.1483	0.0098	0.0042	0.3837
500(5%)	0.0795	0.0035	0.0067	0.2686
1000(10%)	0.0430	0.0008	0.0033	0.1357
2000(20%)	0.0299	0.0007	0.0012	0.0902
3000(30%)	0.0262	0.0005	0.0002	0.0732

Table 1. Relative errors in computing the dot product.

Proof: Omitted due to space constraints.

This bound shows that the error goes to 0 exponentially fast as k increases. Note that although the lemma is based on normal distribution with zero mean and unit variance, it is also true for other distributions that are symmetric about the origin with unit variance. Table 1 depicts the relative error of the distributed dot product between two synthetically generated binary vectors of size 10000. k is the number of randomized iterations (represented as the percentage of the size of the original vectors). Each entry of the random matrix is chosen independently and uniformly from $\{1, -1\}$. In practice, this bound can be used to find the suitable k.

4 Algorithm Details

4.1 Main Procedure

At the commencement of the algorithm, each site determines which local attribute offers the largest information gain. No communication is required to accomplish this. The best attribute from each site is then compared and the attribute with the globally largest information gain, A_G, is selected to define the split at the root node of the tree. For each distinct value $a \in \Pi(A_G)$, a new branch leading down from the root is created. For each these branches, the site containing A_G constructs a binary vector representing which tuples correspond to this new branch, $\vec{V}(A_G = a)$, and sends the projection of it to the other site. Upon receiving each vector, the other site indexes it according to it's path and stores it in a *vector cache* for later use.

At each non-root node Z, each party, P attempts to find the nearest closest ancestor of Z that splits on an attribute *not* local to P (one may not exist). Consider Figure 2 with $P = Adam$. When considering node $Z1$, path (1), the nearest non-local ancestor would be the grandparent of $Z1$. For node $Z2$, path (2), the nearest non-local ancestor would be the parent of $Z2$. For node $Z3$ no non-local ancestor exists.

If P fails to find a non-local ancestor for Z (*i.e.*. the search terminated at *root*)then P does not require any information from the other party to compute the information gain of it's attributes. In this case the evaluation of the information gain proceeds as it does at *root* and can be calculated exactly. Otherwise, P retrieves the appropriate entry from it's *vector cache* and uses it to approximate the information gain of it's local attributes using the distributed dot product. Note that,in either case, no communication is required.

As before, once each site determines the local attribute with the largest information gain, the attribute with the globally largest information gain, A_G, is selected to define the split at Z. Following this, each party now executes one of the following actions

- If A_G is local to P then, for each $a \in \Pi(A_G)$, a new branch leading down from the root is created. For each branch, P constructs a binary vector representing which tuples corresponding to this new path and sends the projection of it the other party.
- If A_G is non-local then P waits until it receives the projection vector from the other party, indexes it according to it's respective path, and stores it in the local *vector cache*.

The total number of messages required the above actions is k (the number of columns of R).

In order to reduce the memory signature of the algorithm each site will occasionally check the contents of it's *vector cache* and delete any invalid entries. A vector becomes *invalid* when (1) every path associated with that vector terminates in a leaf node, or (2) the node which generated the vector is no longer the nearest non-local ancestor to any of it's descendants.

We made one minor change to the algorithm presented above. When the number of ones/zeros in a binary vector is less than the number of iterations k, we can just transmit the list of indices directly. Not only does this reduce the communication cost of the algorithm even further, it allows the calculation of information gain further down the tree to be made in an exact, rather than approximate, manner.

4.2 Leaf Nodes Determination

The construction of a path down the decision tree continues until a *leaf node* is reached, which meets at least one of the following criteria: (1) All of the tuples for the node belong to one class. The node is then labeled by that class. (2) If any child of a node is empty, label that child as a leaf representing the most frequent class in this node. (3) There are less than *minNumObj* (4 in our experiments) tuples for the node, regardless of class. The node is then labeled by the most frequent class of all the tuples in this node. Note here that the calculations used to determine this may be approximations based on the distributed dot product.

From the above criteria, we can see that the determination of a leaf node can actually be made by its parent since information gain computation enables the parent to get the

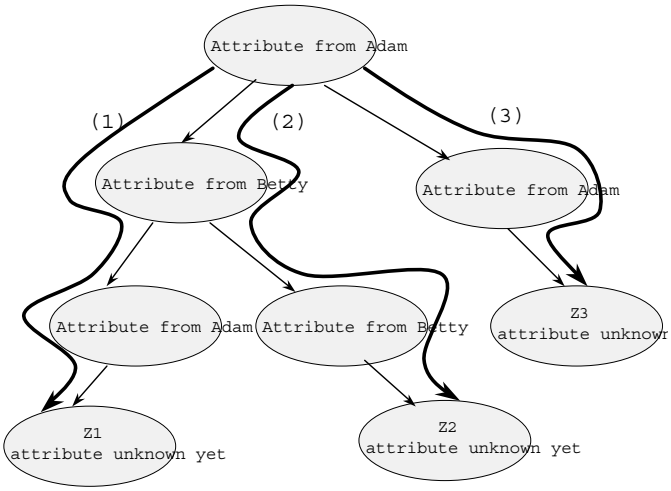

Figure 2. Sample distributed tree structure.

total number tuples for each child, together with the number of tuples belonging to different classes in its child. After a node is split, and if the leaf node is determined, the site who owns the splitting attribute can just send the other site(s) the tree branch name, total number of tuples covered by the leaf and the number of misclassified ones (No projected vector is transmitted). This information will be later used for tree pruning.

4.3 Communication Complexity

For each non-leaf node (except the root where only 2 messages are required to find the best split), to find the best split, only one party needs to send a projected vector to the other (k messages). After each party evaluating its local attributes, they exchange the name and information gain value of their best attribute and decide the split (2 messages). Note that, the leaf node can be decided right away by its parent, then one parent needs to send the other site the branch name, total number of tuples covered by the leaf and total number of misclassified ones (3 message). The total number of messages is bounded by $O(LT + kIT)$ where IT denotes the number of non-leaf node (except root) and LT is the number of leaf nodes. This can be generalized to S sites as $O((LT + kIT)(S - 1))$.

Note that the communication complexity does not depend on the number of distinct values of any attribute. As a result, continuous attributes do not directly increase the communication complexity. However, if continuous attributes are split using the standard single threshold method, they may create deeper trees thus indirectly increasing the communication complexity.

4.4 Tree Pruning

Pessimistic post-pruning approach can be applied. From the discussion in the previous section, we know that each leaf node has the information about total number of records it covers and the number of misclassified ones. We can therefore compute the predicted error rate over the entire population of the records covered by this leaf from the confidence limits. Furthermore, because both parties will have a copy of the tree, and no validation data are required, they can prune the tree independently. Since this procedure requires no message communication and is totally the same for both centralized and distributed models, in our experiment we only compared the performance without pruning.

5 Optimizations

The algorithm described in Section 4.1 implicitly applies two important communication optimization strategies: *message sharing* and *message reusing*.

Message Sharing: *Messages associate with a projected vector from one site can be used to compute the information gains of all the attributes owned by the other site.* As a concrete illustration, consider the tree in Figure 2. To find the best split for node $Z1$ on path (1), Betty needs to evaluate her local attributes. Since the parent node is split by Adam's attribute, Adam will send Betty one projected vector which contains the information about tuples that satisfy the constrains induced by Adam's attributes along path (1). On receiving this information, Betty can approximate the information gain of all her attributes without any other communications.

Message Reusing: *Previously sent messages can also be reused as we descend down the tree.* This property is realized by the cache employed for each active path as we discussed in Section 4.1. As an example, consider again node $Z1$ on path (1) in Figure 2. In order for Adam to evaluate his attributes on node $Z1$, he needs to know the information about tuples that satisfy the constrains specified by Betty's attributes along the path. However, in this case no message are required because Adam has cached this information before when evaluating the parent node of $Z1$. Further more, this cached information can be continuously used if all the nodes along the path starting from N are split on Adam's local attributes. The cache will not be updated until a node on the path is split by Betty's attribute.

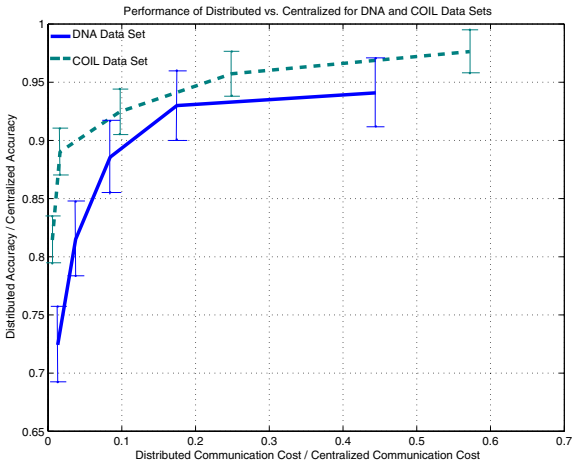

Figure 3. Accuracy vs. Cost of distributed decision tree over DNA and COIL data sets using 10-fold cross validation.

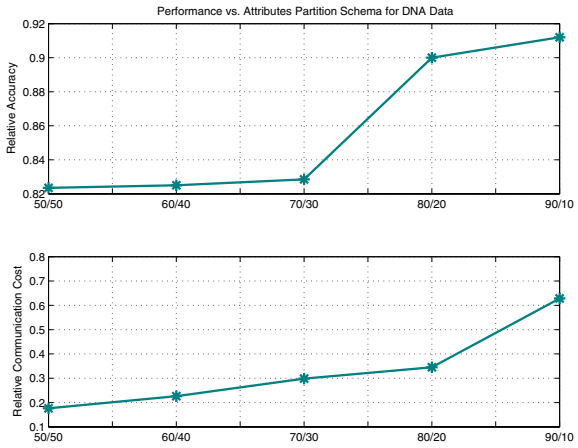

Figure 4. Performance of distributed decision tree with regard to different attributes partition schema for DNA data.

6 Experiments

We conducted experiments on two public domain data sets to evaluate the performance of our algorithm. The first one is StatLog DNA data set [3] which is a processed version of Molecular Biology Databases from UCI Machine Learning Repository. This data set consists of 2000 DNA nucleotide sequences, each with 180 binary attributes and 1 three-valued class label. The second one comes from COIL 2000 Challenge [4]. It contains information on customers of an insurance company and consists of 5822 samples with 86 nominal valued attributes. We vertically partitioned each data set into two separate subsets with alternative sampling of the attribute. The implementation is based on J48 from Weka-3-4.

Figure 3 illustrates the results of our experiments. The X-axis indicates the ratio of the communication cost induced by the distributed algorithm to the cost of centralizing data which is simply half of the size of the original data, *i.e,* $0.5MN$. The Y-axis corresponds to the ratio of the accuracy of the distributed model to the centralized model. We computed the 95% confidence interval of the accuracy of centralized model, together with the confidence interval of distributed model for different communication cost. Then a best case optimistic bound (confidence upper bound of the accuracy of distributed model divided by the lower bound of the centralized model) and worst case pessimistic bound (confidence lower bound of the accuracy of distributed model divided by the upper bound of the centralized model) are calculated and plotted as the error bar.

[3] http://www.liacc.up.pt/ML/statlog/datasets/dna/dna.doc.html
[4] http://kdd.ics.uci.edu/databases/tic/tic.html

In both cases, the figure shows that by using only 20% of the communication cost necessary to centralize the data we can achieve trees at least 80% as accurate as the centralized version. Although we didn't carry out experiments for more than two sites, our analysis shows that the communication complexity scales linearly with the number of sites.

To investigate the effect of the way that attributes are partitioned among different sites on the performance of distributed decision tree, we fixed the random seed and length of the projected vectors, then tested the algorithm while changing the number of attributes present on each site (50 vs.50, 60 vs.40, . . ., 90 vs.10). Figure 4 gives the result. Generally, the accuracy increases as the attributes are partitioned more and more unequally among different sites. We believe that this is due to the increasing likelihood, as the attributes become more and more unbalanced, that the entire paths through the tree, from $root$ to $leaf$, will split on attributes mostly located on the same site. Thus the information gain for the corresponding splits can be computed locally and will not suffer from the randomization inherent in the distributed dot product. In the meantime, we also noticed a corresponding increase of the relative communication cost as the partition becomes more unequally. Although we are still looking into this behavior, we do have an intuitive explanation. Since when centralizing the data, we will always send the data from the site with fewer attributes to the site with more. Therefore, the cost of centralized algorithm decreases linearly with the number of attributes on the smaller site. On the other hand the communication cost of our distributed algorithm depends primarily on the number of interior nodes of the resulting tree, which in turn depends on characteristics of the dataset itself and not on the way the data is partioned. With regard to the *cost-benefit ratio*,

our algorithm might perform at it's best when the attributes are partioned in a balanced fashion. The results shown in Figure 3, which were obtained with attributes evenly distributed between two sites, support this hypothesis.

7 Conclusions and Future Work

We have presented an algorithm that allows the efficient construction of a decision tree over heterogeneously distributed data. The key of our approach is a random projection-based dot product estimation and message sharing strategy. The experimental results are very promising and show that this technique reduces the communication by a factor of five while still retaining 80% of the original accuracy.

A primary set of directions for future work is motivated by the fact that our distributed algorithm requires more computation (local) than the centralized algorithm. One of the fundamental reasons is that our algorithm must compute a matrix, vector product for each frequency count. The overall benefit of our algorithm hinges on a trade-off: increased local computation, reduced communication. In settings where total computation time is the most important factor, the overall benefit depends on communication delay. One direction for future work involves carrying out a careful timing study to compare the total algorithm times (distributed vs. centralized) taking into account communication delays. The goal would be to determine how large the communication delay need be to offset the extra local computation time.

In settings where factors other than time cannot be ignored (*e.g.* energy consumption, privacy), the reduced communication could offer benefits in spite of an increased total computation time. For example, in energy constrained environments, communication typically requires more energy that computation. Also, if maintaining the privacy of each party's data is a high priority, increased computation time is reasonable sacrifice for reduced communication since this means less information need be protected. The reduced communication offered by our algorithm makes it a decent starting point for developing a more efficient privacy-preserving distributed decision tree induction algorithm than currently reported in the literature. Indeed, another direction for future work involves incorporating secure multi-party computation (SMC) based protocols to address privacy constrains while retaining low communication complexity.

Acknowledgments

The authors acknowledge supports from the United States National Science Foundation (NSF) CAREER award IIS-0093353, NSF Grant IIS-0329143, and NASA (NRA) NAS2-37143.

References

[1] R. I. Arriaga and S. Vempala. An algorithmic theory of learning: Robust concepts and random projection. In *Proceedings of the 40th Foundations of Computer Science*, New York, NY, October 1999.

[2] D. Caragea, A. Silvescu, and V. Honavar. Learning decision trees from distributed heterogeneous autonomous data sources. In *Proceedings of the Conference on Intelligent Systems Design and Applications (ISDA'03)*, Tulsa, Oklahoma, 2003.

[3] T. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting and randomization. *Machine Learning*, 40(2):139–158, 2000.

[4] W. Du and Z. Zhan. Building decision tree classifier on private data. In *Workshop on Privacy, Security, and Data Mining at the 2002 IEEE International Conference on Data Mining (ICDM'02)*, Maebashi, Japan, December 2002.

[5] L. O. Hall, N. Chawla, K. W. Bowyer, and W. P. Kegelmeyer. Learning rules from distributed data. In M. J. Zaki and C.-T. Ho, editors, *Large-Scale Parallel Data Mining*, volume 1759 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[6] R. Hecht-Nielsen. Context vectors: General purpose approximate meaning representations self-organized from raw data. In *Computational Intelligence: Imitating Life*, pages 43–56. IEEE Press, 1994.

[7] H. Kargupta and V. Puttagunta. An efficient randomized algorithm for distributed principal component analysis from heterogeneous data. In *Agnik L.L.C. Technical Report 2004-002*, 1450 S. Rolling Road, Baltimore, MD 21227, USA, 2004.

[8] H. Kargupta and K. Sivakumar. Existential pleasures of distributed data mining. In H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha, editors, *Data Mining: Next Generation Challenges and Future Directions*. MIT/AAAI press, 2004.

[9] D. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.

[10] B. Park and H. Kargupta. Distributed data mining: Algorithms, systems, and applications. In N. Ye, editor, *The Handbook of Data Mining*, pages 341–358. Lawrence Erlbaum Associates, Mahwah, N.J., 2003.

[11] B. Park, H. Kargupta, E. Johnson, E. Sanseverino, D. Hershberger, and L. Silvestre. Distributed, collaborative data analysis from heterogeneous sites using a scalable evolutionary technique. *Applied Intelligence*, 16, January 2002.

[12] A. Prodromidis and P. Chan. Meta-learning in distributed data mining systems: Issues and approaches. In H. Kargupta and P. Chan, editors, *Advances of Distributed Data Mining*. AAAI Press, 2000.