Decision Tree Induction in High Dimensional, Hierarchically Distributed Databases

Amir Bar-Or, Assaf Schuster, Ran Wolff
Faculty of Computer Science
Technion, Israel
{abaror, assaf, ranw}@cs.technion.ac.il

Daniel Keren
Department of Computer Science
Haifa University, Israel
dkeren@cs.haifa.ac.il

Abstract

Classification based on decision trees is one of the important problems in data mining and has applications in many fields. In recent years, database systems have become highly distributed, and distributed system paradigms such as federated and peer-to-peer databases are being adopted. In this paper, we consider the problem of inducing decision trees in a large distributed network of high dimensional databases. Our work is motivated by the existence of distributed databases in healthcare and in bioinformatics, and by the vision that these database are soon to contain large amounts of genomic data, characterized by its high dimensionality. Current decision tree algorithms would require high communication bandwidth when executed on such data, which is not likely to exist in large-scale distributed systems. We present an algorithm that sharply reduces the communication overhead by sending just a fraction of the statistical data. A fraction which is nevertheless sufficient to derive the exact same decision tree learned by a sequential learner on all the data in the network. Extensive experiments using standard synthetic SNP data show that the algorithm utilizes the high dependency among attributes, typical to genomic data, to reduce communication overhead by up to 99%. Scalability tests show that the algorithm scales well with both the size of the dataset, the dimensionality of the data, and the size of the distributed system.

Keywords: data mining, distributed algorithms, decision trees, classification, high dimension data.

1 Introduction

The analysis of large databases requires automation. Data mining tools have been shown to be useful for this task, in a variety of domains and architectures. It has recently been shown that data mining tools are extremely useful for the analysis of genomic data as well [12]. Since the number of genomic databases and the amount of data in them increases rapidly, there is a dire need for data mining tools designed specifically to target genomic data specifically.

Classification, the separation of data records into distinct classes, is apparently the most common data mining task, and decision tree classifiers are perhaps the most popular classification technique. Some recent works have shown that classification can be used to analyze the effect of genomic, clinical, environmental, and demographic factors on diseases, response to treatment, and the risk of side effects [9]. Providing efficient decision tree induction algorithms suitable for genomic data is therefore an important goal.

One interesting aspect of genomic databases is that they are often distributed over many locations. The main reason for this is that they are produced by a variety of independent institutions. While these institutions often allow a second party to browse their databases, they will rarely allow this party to *copy* them. There could be a number of reasons for this: the need to retain the privacy of personal data recorded in the database, through questions regarding its ownership, or even because the sheer size of the data makes copying non-permissively costly in CPU, disk I/O or network bandwidth.

Our lead example in this paper is the task of mining genomically enriched electronic medical records (EMRs). Within a few years it is expected that each patient's medical record will contain a genomic fingerprint. This fingerprint will be used mainly to optimize treatment and predict side effects. Existing genomic fingerprinting techniques, such as single nucleotide polymorphisms (SNPs) and Gene Expression Microarrays, yield records with tens of thousands of entries that are usually interpreted as binary (normal/abnormal allele or active/inactive gen, respectively). It is common perception that an illness or treatment side effect can many times relate to just single SNPs or to the expression of few genes.

Data mining of genomically enriched EMRs would be needed for the identification of unknown correlations and for the development of new drugs. It would best be performed on a national scale, using EMRs gathered by many different health maintenance organizations (HMOs). This would naturally extend the functionality of systems such as RODS and NRDM [13] which already collect and analyze health data at a regional (RODS) and national (NRDM) scale. RODS, for example, accesses the database of tens of hospitals using the HL7 protocol to retrieve statistical information and detect disease outbreaks. Nevertheless, it is unlikely that an HMO would allow systems such as RODS to download its entire database. Hence, the need for distributed algorithms.

A distributed decision tree induction algorithm is one that executes on several computers, each with its own database partition. The outcome of the distributed algorithm is a decision tree which is the same as, or at least comparable with, a tree that would be induced were the different partitions collected to a central place and processed using a sequential decision tree induction algorithm. Since decision tree induction poses modest CPU requirements, the performance of the algorithm would usually be dictated by its communication requirements.

Previous work on distributed decision tree induction usually focused on tight clusters of computers, or even on shared memory machines [4–6, 10, 11]. When a wide area distributed scenario was considered, all these algorithms become impractical because they use too much communication and synchronization. A kind of decision tree induction algorithm which is more efficient in a wide area system employs meta-learning. However, these produce a heuristic approximation rather than the optimal result produced by the former algorithms and, thus, are not considered in this paper. Because genomic databases contain many (thousands) attributes for each data instance and can be expected to be distributed over many distant locations, current distributed decision tree induction algorithms are ill-fit for them.

In this paper we describe a new distributed decision tree algorithm, Distributed Hierarchical Decision Tree (DHDT). DHDT is executed by a collection of agents which correlate with the natural hierarchy of a national virtual organization. For instance, the leaf level agents may correspond to different HMOs (or clinics within an HMO) while upper levels correspond to regional, state and national levels of the organization. DHDT focuses on reducing the volume of data sent from each level to the next while preserving perfect accuracy (i.e., the resulting decision tree is not an approximation). When tested on genomic SNP data with one thousand SNPs in each data record, DHDT usually collects data about only about a dozen of the SNPs – a 99% decrease in bandwidth requirements. The algorithm is suitable for any high dimention data, provided that the correlations in it are sparse as they are in genomic data. Both the hierarchic organization and the communication efficiency of DHDT give it excellent scalability at no decrease in accuracy.

2 Sequential Decision Tree Induction

The decision tree model was first introduced by Hunt et al. [3], and the first sequential algorithm was presented by Quinlan [7]. This basic algorithm used by most of the existing decision tree algorithms is given here.

Given a training set of examples, each tagged with a class label, the goal of an induction algorithm is to build a decision tree model that can predict with high accuracy the class label of future unlabeled examples. A decision tree is composed of nodes, where each node contains a test on an attribute, each branch from a node corresponds to a possible outcome of the test, and each leaf contains a class prediction. Attributes can be either *numerical* or *categorical*. In this paper, we deal only with categorical attributes. Numerical attributes can be discretisized and treated as categorical attributes; however, the discretization process is outside the scope of this paper.

A decision tree is usually built in two phases: A growth phase and a pruning phase. The tree is grown by recursively replacing the leaves by test nodes, starting at the root. The attribute to be tested at a node is chosen by comparing all the available attributes and greedily selecting the attribute that maximizes some heuristic measure, denoted as the *gain function*. The minimal and sufficient information for computing most of the gain functions is usually contained in a two-dimensional matrix called the *crosstable* of attribute i. The $[v, c]$ entry of the crosstable contains the number of examples for which the value of the attribute is v and the value of the class attribute is c.

The decision tree built in the growth phase can "overfit" the learning data. As the goal of classification is to accurately predict new cases, the pruning phase generalizes the tree by removing sub-trees corresponding to statistical noise or variation that may be particular only to the training data. This phase requires much less statistical information than the growth phase; thus it is by far less expensive. Our algorithm integrates a tree generalization technique suggested in PUBLIC [8], which combines the growing and pruning stages while providing the same accuracy as the post-pruning phase. In this paper, we focus on the costly growth phase.

2.1 Gain Functions The most popular gain functions are information gain [7], which is used by Quinlan's ID3 algorithm, and the Gini Index, which is used by Brieman's Cart algorithm, among others.

Consider a set of examples S that is partitioned into M disjoint subsets (classes) $C_1, C_2, ..., C_M$ such that $S = \bigcup_{i=1}^{M} C_i$ and $C_i \bigcap C_j = \emptyset$ for every $i \neq j$. The estimated probability that a randomly chosen instance $s \in S$ belongs to class C_j is $p_j = \frac{|C_j|}{|S|}$, where $|X|$ denotes the cardinality of the set X. With this estimated probability, two measures of impurity are defined: $entropy(S) = -\sum_j p_j log p_j$, and $Gini(S) = \sum_j p_j^2$.

Given one of the impurity measures defined above, the gain function measures the reduction in the impurity of the set S when it is partitioned by an attribute $\mathbf{A}$ as follows: $Gain_{\mathbf{A}}(S) = \sum_{v \in Values(\mathbf{A})} \frac{|S_v|}{|S|} Imp(S_v)$, where $Values(\mathbf{A})$ is the set of all possible values for attribute $\mathbf{A}$, S_v is the subset of S for which attribute $\mathbf{A}$ has the value v, and $Imp(S)$ can be $entropy(S)$ or $Gini(S)$.

3 Bounds on the Gain Functions

The bounds given in this section bound the gain function of a population that is the union of several disjoint subpopulations on which only partial information is available. By using them we can avoid collecting the crosstables of many of the attributes whose gain, as indicated by the bounds, cannot be large enough to change the result.

3.1 Notations The bounds given below are defined for a single attribute of a single decision tree leaf node. Therefore, we simplify the notations by removing references to the attribute and the decision tree node. Let P be a population of size n and let $\{P_1, P_2\}$ be a partition of P into two subpopulations of sizes n_1, n_2 respectively. Let the crosstables of populations P_1, P_2, P be defined as:

$$\overrightarrow{P_1}(value, class) = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix},$$

$$\overrightarrow{P_2}(value, class) = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix},$$

$$\overrightarrow{P}(value, class) = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} \\ a_{21}+b_{21} & a_{22}+b_{22} \end{pmatrix}$$ respectively.

Here, $a_{i,j}$ and $b_{i,j}$ denote the number of learning examples with value i and class j in $\overrightarrow{P_1}$ and $\overrightarrow{P_2}$, respectively.

In the algorithm described here we rely on the following two bounds, the proof of which is omitted due to space considerations:

THEOREM 3.1. *Let P be a population of size n, and $\{P_1, P_2, ..., P_k\}$ a partition of P into k subpopulations of sizes $n_1, n_2, ..., n_k$ respectively. Let $G()$ denote the gain function (information gain or Gini index). Then an upper bound on $G(P)$ is given by:*

$$G(P) \leq \frac{\sum_{i=1}^{k} n_i G(P_i)}{\sum_{i=1}^{k} n_i}.$$

THEOREM 3.2. *Let P be a population of size n, and $\{P_1, P_2\}$ a partition of P into two subpopulations of sizes n_1, n_2 respectively. Assume that the candidate split divides P_1 into two subsets, P_1^{left} and P_1^{right}, with sizes n_1^{left} and n_1^{right} respectively. Let $G()$ denote the gain function (information gain or Gini index). Then, lower bounds on $G(P)$ is given by:*

$$G(P) \geq \frac{G(P_1)}{\left[1 + \frac{n_2}{n_1}\right]\left[1 + \frac{n_2}{\min\{n_1^{left}, n_1^{right}\}}\right]}$$

4 Distributed Hierarchical Decision Tree

The distributed hierarchical decision tree (DHDT) algorithm runs on a group of computers, connected through a wide-area network such as the Internet. Each computer has its own local database. The goal of DHDT is to derive exactly the same decision tree learned by a sequential decision tree learner on the collection of all data in the network. We assume a homogeneous database schema for all databases, which can be provided transparently, if required, by ordinary federated system services. The algorithm relies on a (possibly overlay) communication tree that spans all computers in the group. The communication tree can be maintained by a spanning tree algorithm or can utilize the natural hierarchy of the network. For reasons of locality, communication between nodes in the lower levels of the spanning tree is often cheaper than communication between nodes in the upper levels. Thus, a "good" algorithm will use more communication at the bottom than at the top of the tree. We further assume that during the growth phase of the decision tree, the databases and the communication tree remain static.

Every computer in the group employs an entity called *Agent* that is in charge of computing the required statistics from the local database and participates in the distributed algorithm. Agents collect statistical data from their children agents and from the local database and send it to their parent agent at its request.

The root agent is responsible for developing the decision tree and making the split decisions for the new decision tree leaves. First, the root agent decides whether a decision tree leaf has to be split according to one or more stopping conditions (e.g., if the dominance of the majority class has already reached a certain threshold) or according to the PUBLIC method [8], which avoids splitting a leaf once it knows it may be pruned eventually. The class distribution vector, which holds the number of examples that belong to each distinct class in the population, is sufficient for computing these functions, and thus it is aggregated by the agents over the communication tree to the root agent.

```
Definitions
D1. border= maximal lower bound of all attributes which were not sent to the parent
D2. borderAttribute= the attribute whose lower bound defines the border
D3. If agent is root then
D4.     ExtraCondition = There is only a single attribute $A_i$ where $UpperBound(A_i) \geq border$ or
.                                   $max_i(UpperBound(A_i)) = border$
D5. Else
D6.     ExtraCondition = $G_u^i < border$ for all children
Algorithm
Phase 1: Starts when a new leaf is born
01. Receive information from all children
02. While (not (border defines a clear separation and ExtraCondition)) do
03.     If ($G_u^i > border$) then
04.         request $child_i$ to lower its border and send new information
05.     Else if (border does not define a clear separation and
.               crosstable of borderAttribute has only partial information)
06.         request information for borderAttribute from children who did not send complete information
07.     Else
08.         request information for all attributes that cross the border
09.     End if
10. Receive information from all children
11. End while
12. Return attributes $A_i$ where $LowerBound(A_i) \geq border$
Phase 2: Starts when an agent receives a request for more information from its parent
01. If (parent requires more information for attribute $attr_i$) then
02.     If (crosstable of $attr_i$ was not sent to parent) then
03.         Send parent the crosstable of $attr_i$
04.     Else
05.         request information for $attr_i$ from children who sent partial information regarding $attr_i$
06. Else (the case where parent requests that the border be lowered)
07.     Update border and borderAttribute and start phase 1.
08. Endif
```

Algorithm 1: DESAR Algorithm

Recall that if a decision tree leaf has to be split, the split must be done by the attribute with the highest gain in the combined database of the entire network. All that is required to decide on the splitting attribute is thus an *agreement* as to which attribute has the maximal gain; the actual gain of each attribute does not need to be computed. To reach agreement, the agents participate in a distributed algorithm called DESAR (Distributed Efficient Splitting Attribute Resolver). For each new leaf that has to be developed, DHDT starts a new instance of DESAR to find the best splitting attribute. We proceed to describe the DESAR algorithm.

4.1 Distributed Efficient Splitting Attribute Resolver To find the best splitting attribute while minimizing communication complexity, DESAR aggregates only a subset of the attribute crosstables over the communication tree to the root agent. The algorithm starts when the agents receive a message that is broadcast down the communication tree (initiated by the root and transmitted by each agent to all its children), asking for the development of a new leaf in the decision tree. Then, each agent waits for messages from its children. When messages are received from all children, the agent combines the received crosstables with its own local crosstables, picks the most *promising* attributes on the basis of its aggregated data, and sends the corresponding crosstables to its parent agent.

Algorithm 1 describes DESAR pseudocode, uniformly executed by all agents. For space considerations, we only provide pseudo-code here.

5 Experimental Evaluation

The DHDT algorithm is designed to run on datasets with a large number of attributes, such as the genomically enriched EMR. However, such data is not yet available for large-scale data mining. Therefore, we adopted an approach common in bioinformatics studies on the association of phenotype with SNP data. In this approach, synthetic SNP data is generated by a theoretical model, and then one SNP serves as the phenotype we wish to classify. Since some diseases are correlated strongly with a single SNP variation, learning a model which predicts an SNP's allele is equivalent to learning a model which predicts one of these diseases. We synthesized the SNP data using two data generators ([1,2]) with typical parameters to generate two datasets, where each of the generators uses a different theoretical model.

Each dataset contains 250,000 examples describing a single population. A single SNP is described by a binary attribute where '0' denotes the most common allele. An example is composed of 1000 SNPs. An arbitrary SNP is designated the class attribute. The experiments were performed on a simulation of a communication tree that spans all agents in the system. At the beginning of each experiment, each agent builds its local database by sampling a small fraction of the simulated dataset, thus emulating a specific subpopulation.

Unless otherwise stated, experiments have been run assuming a spanning communication tree of degree three and height six, totaling in 1093 Agents. Each agent has a database (population) of 5000 samples and 1000 attributes (alleles) per sample. The average resulting decision tree had 25 nodes and a misclassification rate of 3%.

5.1 Experiments Our first experiment measures the average communication overhead of a single split decision (i.e., a single run of the DESAR algorithm) in terms of the number of messages and the number of sent crosstables. These results are compared with previous distributed decision tree algorithms which collect and aggregate the crosstables of all attributes.

Our algorithm demonstrates an average reduction of more than 99% in the number of transmitted bytes, with only a small increase in the average number of sent messages (1.2 per Agent per decision tree node). These results are summarized in Fig. 1.

Additional experiments have proved the algorithm is scalable with respect to the size of the network, the number of total attributes, and the size of the local databases. For space considerations, we only present results for network size scalability (Fig. 2).

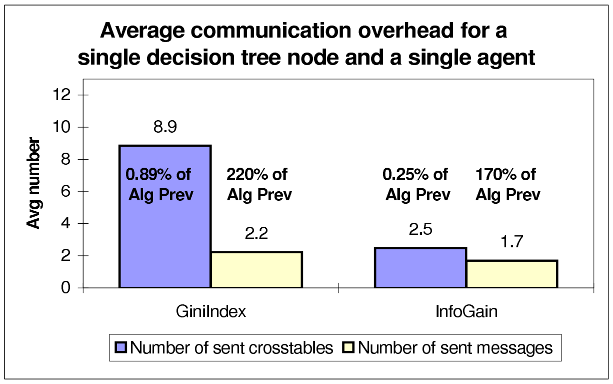

Figure 1: Average communication overhead, comparing to [11] which sends 1000 attributes in a single message

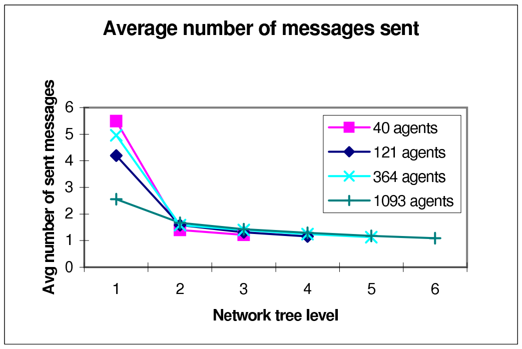

Figure 2: Scalability in network size. The above figure show the distribution of the average communication overhead over the network tree levels for different network sizes (Gini index).

References

[1] G. Greenspan and D. Geiger. Model-based inference of haplotype block variation. *RECOMB*, pages 131–137, 2003.

[2] R. R. Hudson. Generating samples under a Wright-Fisher neutral model of genetic variation. *Bioinformatics*, 18:337–338, 2002.

[3] E. B. Hunt, J. Marin, and P. T. Stone. *Experiments in Induction*. Academic Press, 1966.

[4] R. Jin and G. Agrawal. Communication and memory efficient parallel decision tree construction. In *Proc. of the 3rd (SDM)*, 2003.

[5] M. V. Joshi, G. Karypis, and V. Kumar. A new scalable and efficient parallel classification algorithm for mining large datasets. In *Proc. of International Parallel Processing Symposium*, 1998.

[6] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. In *Proc. of the Fifth Int'l Conference on Extending Database Technology, Avignon, France*, 1996.

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

[8] Rajeev Rastogi and Kyuseok Shim. PUBLIC: A decision tree classifier that integrates building and pruning. *Data Mining and Knowledge Discovery*, 4(4):315–344, 2000.

[9] N. J. Risch. Searching for genetic determinants in the new millennium. In *Nature 405*, pages 847–856, 2000.

[10] J. C. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proc. of the 22nd VLDB Conf.*, 1996.

[11] A. Srivastava, E.-H. (S.) Han, V. Kumar, and V. Singh. Parallel formulations of decision-tree classification algorithms. *Data Mining and Knowledge Discovery: An International Journal*, 3:237–261, 1999.

[12] W. Sthlinger, O. Hogl, H. Stoyan, and M. Muller. Intelligent data mining for medical quality management. In *the Fifth Workshop on Intelligent Data Analysis in Medicine and Pharmacology (IDAMAP-2000), Workshop Notes of the 14th European Conference on Artificial Intelligence (ECAI-2000), pp. 55-67*, 2000.

[13] M. M. Wagner, J. M. Robinson, F.-C. Tsui, J. U. Espino, and W. R. Hogan. Design of a national retail data monitor for public health surveillance. *Journal of the American Medical Informatics Association JAMIA*, 10(5):409–418, Sep/Oct 2003.