Learning by Cooperating Agents

Raj Bhatnagar
ECECS Department, ML-0030
University of Cincinnati
Cincinnati, OH 45221
bhatnagr@ucunix.san.uc.edu

Abstract

Most algorithms for learning and pattern discovery in data assume that all the needed data is available on one computer at a single site. This assumption does not hold in situations where a number of independent databases reside on different nodes of a network. These databases cannot be moved to a common shared site due to size, security, privacy, legal, and data-ownership concerns but all of them together constitute the dataset in which patterns must be discovered. These databases, however, may be made accessible for certain types of queries and all such communications for a database may be channeled through an intelligent agent interface. In this paper we show how a decision-tree induction algorithm may be adapted for such situations and implemented in terms of communications among the interface agents.

Introduction

Most learning and pattern discovery algorithms have been designed for environments in which all relevant data is available at one computer site. Increasingly, pattern discovery tasks are encountering situations in which the relevant data exists in the form of a number of networked databases that are geographically distributed. A common constraint in these situations is that the data in databases cannot be moved to other sites due to size, security, privacy, legal and data-ownership concerns. In this paper we present details of adapting a decision-tree induction algorithm for the case of such constrained sets of databases.

Summary of Relevant Research

Learning from databases is a widely investigated field and decision-tree induction is a very well known and well researched topic (Ming 89a; Ming 89b; Brei 84; Quin 86). Algorithms that use information as a heuristic for guiding towards smaller decision-trees are discussed in (Brei 84; Quin 86). A number of heuristics to guide the search towards smaller decision trees have been reviewed in (Ming 89a; Bunt 92). However, all these heuristics and algorithms assumes that the data from which decision trees are to be induced is available in the form of a relation on a single computer site.

In the context of database research much work has been done towards optimization of queries from distributed databases. It was pointed out in (Yu 84) that a distributed query is composed of the following three phases: (i) *Local Processing Phase* in which *selection* and *projection* etc. operations are performed at individual sites; (ii) *Reduction Phase* in which reducers such as semijoins and joins are used to reduce the size of relations; and (iii) *Final Processing Phase* in which all resulting relations are sent to the querying site where final query processing is performed. However, discovery of patterns from geographically distributed databases does not require that the relevant data and relations be necessarily gathered at the site initiating the learning task. The learning site can do with only need some statistical summaries about data from the distributed sites. In some situations individual sites do not allow any data to be sent out of the site but permit sending statistical summaries to some authorized sites. Phases (ii) and (iii) of distributed query processing are therefore not needed if our goal is limited only to discovery of patterns; and databases from which data transfer is not allowed can not perform distributed querying but can still learn patterns by seeking only the statistical summaries.

Intelligent Query Answering and Data clustering in large databases have been addressed in (Han 96; Zhang 96) and their treatment also is limited to databases residing and available at a single network site.

Example Situation

An example situation in which geographically distributed databases with constraints on data transfer are encountered is as follows. Consider the case of a financial institution that maintains credit card and various other types of customer accounts. A number of databases used by this institution, that typically reside in different cities, are: (i)A database containing fixed data about customers such as employer and address information; (ii) A database of credit card charges and payments made by the customer; (iii) A database containing information about vendors that accept the card; (iv) A database containing deposit

and withdrawal transactions; and (v) A database containing credit-rating information about customers. A knowledge discovery task may require discovery of interesting patterns in the dataset that is formed by considering all these databases together.

A number of other examples from military and civilian domains exist in which data from various sources must be used without moving it over the networks and compromising its security.

Cooperation among such independent and geographically distributed databases is typically governed by a number of constraints.

Constraints on Knowledge Sources

The most common restrictions in distributed knowledge environments and the reasons for their existence are as follows:

Immobility of Databases: The actual data from a database may not be transported to any other site. This restriction may arise due to any of the following: the database size is too large; security reasons demand that data not be placed on networks, data-ownership and privacy issues prevent its owners from sharing the data with others. However, these sites may be willing to share summaries of data with other authorized sites.

Update Protection: A database may not permit an agent outside its own site to update its data due to security and data-integrity considerations.

One-Agent interface: At each database site only one software agent may be allowed to coordinate all flow of information into and out of the database. This may be required to guarantee the integrity and security of the database.

Availability: All databases may not be available for cooperation with other databases at all times. For example, if one credit rating company's database is unavailable at any time, we may use another database with similar information. At the time a learning task is to be performed we must check and determine the availability of other databases for cooperation.

Abstraction of Knowledge Environment

The situation of n knowledge sources located at n different sites and represented by their respective agents can be represented as shown in Figure-1 below. A db_i represents a knowledge source present at the i^{th} site.

Associated with each knowledge source is an *Interface Agent* that acts as the only interface for all communications between the knowledge source and other agents in the outside world. This agent is shown by A_is associated with the databases. Any cooperation among the knowledge sources is initiated, negotiated, and transacted by the agents communicating over the network.

In an abstract sense each knowledge source can be viewed as containing tuples of an m-attribute *relation*. The actual underlying data may not be in the form of a relation but the interface agent can create a simulated view in which the external world can see each knowledge source as a database relation. This kind of interface is possible even for databases containing maps and images but the *relation*-like views that may be simulated may be restricted in terms of the nature of attributes contained in them.

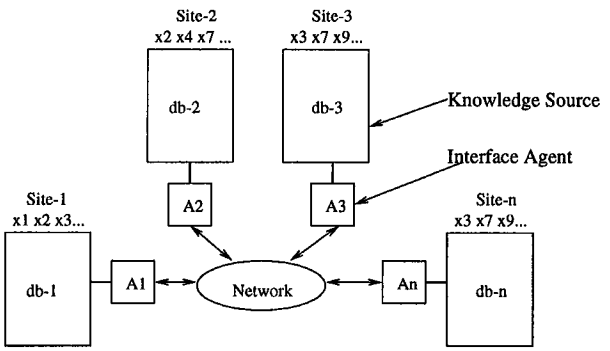

Figure 1: Databases

The set of attributes contained in database db_i is represented by X_i. For any pair of databases db_i and db_j the corresponding sets X_i and X_j may have a set of shared attributes given by S_{ij}. That is, $S_{ij} = X_i \cap X_j$..

For the present discussion we assume that all attributes are nominal-valued. In a later section of this paper we separately address the handling of continuous valued attributes.

The dataset D using which a decision tree is to be constructed is that which is formed by a $Join$ operation performed on all the relations $db_1 \ldots db_n$ identified as relevant for the learning task.

Consequences of Constraints

The tuple space (dataset) from which the decision tree is to be induced is only implicitly specified in terms of the $JOin$ or the cross product of the component databases. That is, this space can be made explicit by performing a cross-product or a $JOIN$-operation on the individual component databases. However, due to the constraints on the movement of data such an explicit dataset can not be created.

Therefore, our pattern discovery algorithms must work with the implicitly specified tuple space. This restriction may not, at first, appear drastic but after some analysis turns out to have severe implications. Any pattern discovery and inference algorithm which demands that explicit data tuples be presented to it can not work in the distributed environments. For example, neural network training requires that each training tuple be presented to the network. But in

the distributed knowledge sources case each tuple has its components residing at individual database sites and the tuple can not be assembled and made explicit. Therefore, the neural network training algorithms are not applicable in the distributed environments.

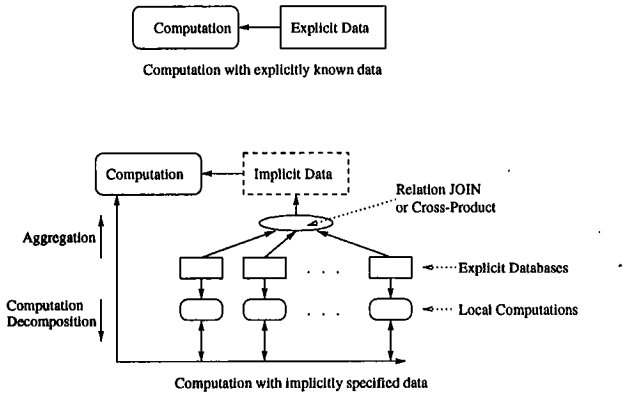

Figure 2: Computations in Explicit vs. Implicit Data Spaces

Only those computations can be performed in the implicit data space that can be decomposed into parts; these parts performed at individual sites; and then the results transmitted and composed at the initiating site. The decomposition and aggregation/composition operations for any particular set of databases are strongly influenced by the sets of shared attributes S_{ij} because all dependences among attributes are mediated through the set shared attributes.

To facilitate handling of the implicitly defined tuples of D we define a set S that is the union of all such intersection sets. That is,

$$S = \cup_{i,j\ ,\ i \neq j} \quad S_{ij} \tag{1}$$

The set S_{ij} is the same intersection set as defined above. The set S contains all those attributes that occur in more than one d_i. We also define a new relation *Shareds* containing all the attributes in set S. The tuples in *Shareds* are made explicit by enumerating all possible combinations of various values for attributes in set S.

The roles sought to be performed by the interface agents associated with database sites are the following:

1. A knowledge discovery task can be initiated by an agent that is located at any one of the n database sites, or possibly any other authorized site.
2. All the X_is (attribute sets) are known to each interface agent and as a consequence, the relation *Shareds* is also known to every agent capable of initiating the decision-tree induction task. With this information, the agent can decompose the tree induction task into suitable queries for agents at other database sites.
3. The agent that initiates the discovery task can send requests to various sites for statistical summaries about their respective d_is. Actual data tuples are not allowed to be exchanged among the agents.
4. The initiating agent can compose responses from other agents and construct the descriptions of decision trees.

We briefly outline here some aspects of the decision tree induction algorithm even though it is a well known algorithm. We do so to facilitate easy reference and comparison with the adapted version of the algorithm.

Decision-Tree Induction Algorithm

Various tree-induction algorithms including ID3 start by considering the complete dataset D as residing at the root of the tree and then repeating the following steps until some specified percentage of tuples at all leaf nodes belong to some unique class (Value of the *Target-Attribute*).

1. Pick one such dataset at a leaf node of the partially constructed tree whose tuples belong to different classes. (By *dataset* here we are referring to any set of tuples belonging to a node of the decision tree.)
2. Select an attribute a_j having m distinct values: $a_{j1}, a_{j2} \ldots a_{jm}$.
3. Split D into m distinct partitions such that the k^{th} partition contains only those tuples for which $a_j = a_{jk}$.
4. The m distinct partitions are added to the tree as child datasets of the partitioned parent dataset. These child nodes reside at the end of m branches emanating from the parent node.

It is desirable to keep the height of the induced decision tree as small as possible. A heuristic that is used to keep the height on the smaller side selects that attribute a_i in Step-2 which minimizes the average informational entropy of the partitions that are formed in step-3. The value of this average entropy can be computed as:

$$E = \sum_{b=1}^{m} \left(\frac{N_b}{N_t} \times \left(\sum_{c} -\frac{N_{bc}}{N_b} log_2 \frac{N_{bc}}{N_b}\right)\right) \tag{2}$$

where N_b is the number of tuples in branch b, N_t is the total number of tuples in all branches, c is the number of possible classes (the values the target attribute can possess), and N_{bc} is the number of tuples in branch b belonging to class c. The attribute that minimizes the average entropy for the resulting partitions is chosen.

Adaptation for Implicit Tuple Space

The following describes one possible adaptation of the tree induction algorithm for distributed databases. The details and complexity analysis have been described in (Bhat 97).

The tree induction algorithm outlined in the preceding section requires an explicit set of tuples at each node of the tree. This set is used for the following operations:

1. Computation of entropy after partitioning a dataset.
2. Testing to determine if all tuples in a dataset belong to the same class.

In case of the constrained distributed databases an explicitly stated set of tuples is not available. Each step of the induction algorithm must adapt itself to work with the implicitly specified set of tuples.

Characterization of a set of tuples:

When a dataset is known explicitly it can be stored as a table in computer memory. After repeated partitionings, smaller datasets can be represented by storing their *identity numbers* along with each tuple as an additional column of the relation.

When the dataset is only implicitly specified there does not exist any facility to store identities of partitions to which individual tuples belong. Description of every partition or cluster must also be implicit. For the case of decision trees the conjunction of tests performed along a path is the implicit description of the dataset at the end of that path. Clustering and pattern discovery algorithms that rely on marking each tuple with their cluster-id as they progress will not be able to work in environments of implicitly specified tuples.

Selecting the Test Attribute:

In step 2 of the algorithm we choose an attribute and use it to partition the selected parent dataset into its children datasets. The attribute that minimizes the average informational entropy is considered the most promising one and is selected. The expression for entropy computation requires the values of the following counts from the parent dataset:

1. N_t;
2. one N_b for each child branch; and
3. one N_{bc} for each *class* for each child branch.

When the tuples are explicitly stated and stored in a table these counts can easily be obtained. For the case of implicitly stated set of tuples we have decomposed the counting process in such a way that each decomposed part can be shipped to an individual database site and the responses composed to reconstruct the counts. The decomposition for obtaining the count N_t is as follows:

$$N_t = \sum_{J_{sk}} \cdots \sum_{J_{s2}} \sum_{J_{s1}} \left(\prod_{i=1}^{n} (N(d_i)_{Sub})\right) \qquad (3)$$

where $Sub = [S1 = S1_{J_{s1}}], [S2 = S2_{J_{s2}}] \ldots, [Sk = Sk_{J_{sk}}]$. In this expression $S1, S2, \ldots Sk$ are the k members of set S defined by expression 1; $J_{S1}, J_{S2}, \ldots J_{Sk}$ are the numbers of possible discrete values that can be assigned to attributes $S1, S2, \ldots Sk$ respectively; and $Si_1, Si_2, \ldots Si_{J_{Si}}$ are all the values that can be assigned to attribute Si. The value n is the number of database sites (d_is) to be considered, and $(N(d_i)_{Sub})$ is the count in relation d_i of those tuples that satisfy the conditions Sub.

It can be seen that the expression for N_t is in the *sum-of-products* form. Each product term takes in the counts of tuples satisfying condition Sub in each d_i and produces the number of distinct tuples that would be contributed to the implicit $Join$ of all d_is. Also, each condition Sub for the entire summation in the above expression corresponds to a tuple of relation *Shareds* described earlier. The summation in the expression is over all the tuples of relation *Shareds*.

This expression, therefore, simulates the effect of a $Join$ operation on all the n sites to compute the count of tuples satisfying Sub that would exist in D.

A very desirable aspect of the particular decomposition of N_t given above is that each product term $(N(T_t)_{sub})$ can also be easily translated into an SQL query of the form:

Select count (*) where *sub* AND *path to dataset*

The learning agent can ship the queries to the agents at other relevant sites and compose the responses according to the expression for N_t. For each tuple in relation *Shareds* the agent will have to send the queries to each of the other n database sites. The responses can then be multiplied to obtain a product-value for a tuple, and the product-values for all the tuples of *Shareds* can be summed to obtain the value N_t.

The decompositions for the counts N_b and N_{bc} are similar to that for N_t. The expressions are stated as follows:

$$N_b = \sum_{J_{sk}} \cdots \sum_{J_{s2}} \sum_{J_{s1}} \left(\prod_{t=1}^{n} (N(T_t)_{SUB})\right) \qquad (4)$$

where $SUB = [S1 = S1_{J_{s1}}], [S2 = S2_{J_{s2}}] \ldots, [Sk = Sk_{J_{sk}}], [B = B_{J_B}]$

The expression for N_b differs from that for N_t by containing an additional summation over the partitioning attribute B and the corresponding addition to the condition part of the product term.

$$N_{bc} = \sum_{J_B} \sum_{J_{sk}} \cdots \sum_{J_{s2}} \sum_{J_{s1}} \left(\prod_{t=1}^{n} (N(T_t)_{SUB})\right) \quad where \qquad (5)$$

$SUB = [S1 = S1_{J_{s1}}], [S2 = S2_{J_{s2}}] \ldots, [Sk = Sk_{J_{sk}}], [B = B_{J_B}], [C = C_{J_C}]$

The expression for N_{bc} differs from that for N_b by containing an additional summation over the target attribute C and the corresponding addition to the condition part of the product term.

The counts N_t, N_b, and N_{bc} can be composed by obtaining responses from individual databases in the

manner described above and the the entropy value for each proposed partitioning can be determined.

Splitting A Dataset:

After deciding to partition a dataset into its children datasets (Step-3) the learning agent needs only maintain the decision tree constructed so far. At the learning site a marking can be maintained for each leaf node indicating whether all its tuples belong to only one or more classes. This can be determined by examining various N_{bc} counts at the time of creating the children datasets.

Continuous-Valued Attributes

For the nominal-valued attributes we know the set of possible values an attribute can take and therefore also know the potential set of tests that may be performed at each node of the decision tree by an attribute. The situation for continuous-valued attributes is somewhat more difficult. In most tree induction algorithms a threshold value is chosen as the test and tuples lying on the two sides of this threshold belong to two different partitions at the child nodes of the tree. The set of candidates for the threshold values for a continuous-valued attribute may be large. An efficient system for determining the candidates has been described in (Fayy 92). According to this system the continuous-valued variable is partitioned into sectors such that each sector corresponds to only one class (value of the target attribute). Figure-3 given below shows the various ranges for a continuous valued attribute A, the unique classes belonging to the tuples falling in each range, and the resulting set of candidate threshold values.

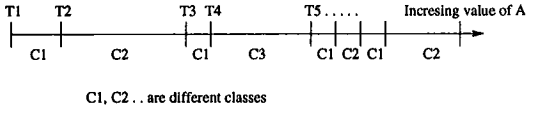

Figure 3: Determining Candidate Threshold Values

testing only for these candidate values is much more efficient than testing for all possible threshold values in the range of the attribute's values.

When all the tuples are stored in one table the above set of candidate threshold values may be obtained by sorting all the tuples according to the continuous valued attribute and then sequentially scanning all the tuples to determine the break points at which class of the tuple changes from one to the other. For determining these break points for a continuous-valued attribute A in the implicit data space we perform the following steps:

1. Find the minimum and the maximum values taken by the the attribute A. This range, if not known to the learning agent, may be determined by enquiring the minimum and maximum values in all those databases in which A exists.
2. Quantize the range for A into an arbitrary number of bins of equal width.
3. Add attribute A and the target attribute to the set S of shared attributes.
4. Construct the tuples in the relation *Shareds* by taking each bin as one possible value for attribute A.
5. Determine the count of all tuples in the implicit dataset corresponding to each tuple of the relation *Shareds*.
6. From the counts obtained construct a histogram by accumulating counts for each range bin of the attribute A. Also keep an account of the classes for tuples belonging to each range bin. This can be done because the target attribute was included in the set S of shared attributes.
7. If all tuples in a range bin belong to the same class we call it a *pure bin*. Adjust the boundaries of the range bins as stated below until all bins are pure or pure to an extent, say, 98
   (a) If two adjoining bins are *pure* and have the same class then merge these bins into one.
   (b) A range bin that is not pure is split into two and the frequencies determined as described above for the smaller bins.
8. The range bins' boundaries are the candidate threshold values.

The above method determines the candidate threshold values without transferring the tuples of data from one site to the other. It seeks from the other sites the count of tuples satisfying certain conditions. Having determined these thresholds the tree induction algorithm can continue in a way similar to that for nominal-valued attributes.

Efficient Agent Implementations:

The decomposition described above can be efficiently implemented and the number of messages that need to be exchanged among agents can be greatly reduced. The counts that need to be obtained from each site for each dataset partitioning are the N_{bc} values for all possible combinations. At the host site these can be appropriately summed up to generate the needed N_b and N_t values. An alternate implementation strategy is to have an application program that travels to each site and accumulates all the needed summaries and then moves to the next database site.

Continuing Research

The research summarized above is currently being extended by us along a number of different directions. The main directions and the preliminary results are as follows:

Complexity Analysis: A number of cost models have been applied to compare the computational cost of decomposed and single-site versions of tree induction algorithm. It turns out that the number of messages that needs to be exchanged among agents increases exponentially with the number of attributes that are shared among the various databases. We are examining the complexity issues for continuous variables.

Other Learning Algorithms: We are examining the decomposed versions for inducing Bayesian classifiers. These results should be available by June 1997. All algorithms that must either present explicit tuples or put markings on individual tuples can not be adapted for above mentioned environments. This includes most neural-net training algorithms and incremental learning algorithms. We are examining other learning algorithms and classifiers for their decomposability.

Conclusion

In this paper we have considered the case of inducing decision trees by a set of cooperating agents. The agents transmit only statistical summaries to other authorized agents and know which summaries should be sought at each level of decision tree induction. We have demonstrated the adaptability of an informational entropy driven decision tree induction algorithm for implementation by such agents. A heuristic for handling continuous-valued attributes has also been shown to be adaptable for such situations.

References

Bhatnagar, R. and Srinivasan, S. *Discovering Patterns in Distributed Databases*, Proceedings of the AAAI-1997.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. 1984. *Classification and Regression Trees*, Belmont, CA: Wadsworth.

Buntine, W., and Niblett, T. 1992. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, vol. 8, pp.75-86.

Fayyad, U. M., Irani, K. B. *On the Handling of Continuous-Valued Attributes in Decision Tree Generation* Machine Learning, vol. 8, pp 87-102, 1992.

Han J., Huang, Y, Cercone, N., and Fu, Y. 1996. Intelligent Query Answering by Knowledge Discovery Techniques. *IEEE Transactions on Knowledge and Data Engineering*, vol. 8. n0. 3, pp 373-390.

Mingers, J. 1989. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, vol. 3, pp 319-342.

Mingers, J. 1989. An empirical comparison of pruning methods for decision-tree induction. *Machine Learning*, vol. 4, pp 227-243.

Quinlan, J. R. 1986. Induction of Decision Trees, *Machine Learning*, vol 1, pp 81-106.

Yu, C, Ozsoyoglu, Z. M., and Kam, K. 1984. Optimization of Distributed Tree Queries, *J, Computer System Science*, vol. 29. no.3 pp 409-445.

Wang, C., Chen, M., 1996. On the Complexity of Distributed Query Optimization. *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 4, pp 650-662.

Zhang, T., Ramakrishnan, R., Livny, M. 1996. *Proceedings of SIGMOD 96*, 6/96, pp 103-114.