Pattern Discovery in Distributed Databases

Raj Bhatnagar, Sriram Srinivasan
ECECS Department, University of Cincinnati
Cincinnati, OH 45221
Raj.Bhatnagar@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 geographically distributed nodes of a computer network. These databases cannot be moved to a single site due to size, security, privacy and data-ownership concerns but all of them together constitute the dataset in which patterns must be discovered. Some pattern discovery algorithms can be adapted to such situations and some others become inefficient or inapplicable. In this paper we show how a decision-tree induction algorithm may be adapted for distributed data situations. We also discuss some general issues relating to the adaptability of other pattern discovery algorithms to distributed data situations[1].

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 databases that are geographically distributed but are connected by communication networks. A common constraint in these situations is that the databases cannot be moved to other sites due to size, security, privacy or data-ownership concerns. In this paper we examine adaptability of various pattern discovery algorithms for such sets of databases. We present details of a decision-tree induction algorithm's adaptation for the case of distributed set 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 algorithms and heuristics assume 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 is interested in only the description of the pattern and can do with only some statistical summaries from the various 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 when our goal is limited only to discovery of patterns. Databases from which transfer of data is not allowed can not participate in distributed querying but can still be useful for pattern-discovery tasks by engaging in an exchange of statistical summaries only.

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 computer site.

An Example Situation

An example situation in which distributed databases with constraints on data transfer are encountered is as follows. Consider the case of a financial institution that maintains credit card accounts. A number of databases used by this institution, that typically reside in differ-

[1]Copyright ©1997, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

ent 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; and (iv) A database containing credit-rating information about customers.

The above scenario comes with the following constraints:

1. The databases or smaller relations extracted from them cannot be transferred from their home sites due to security, size, data ownership and privacy concerns.
2. Each database is designed and maintained independently and therefore the databases, collectively, do not generally constitute a normalized set of relations. Over time, different databases may become available and be added to the set from which patterns are to be discovered and some older databases may be dropped.
3. Some attributes are repeated in various databases.
4. The databases are write-protected in the sense that an agent from outside their respective sites is not permitted to write to a database.
5. The queries permitted to non-local but authorized agents are those that return statistical summaries from the databases. No actual data tuples can be transmitted out of any site.

Similar constraints exist on many commercial, financial, and defense related databases. Despite these constraints it is possible to discover patterns in the collective dataset and we demonstrate it in this paper.

Formal Description of Problem

A pattern discovery task requires the following:

1. A set of tuples D representing the data.
2. A Pattern discovery algorithm.

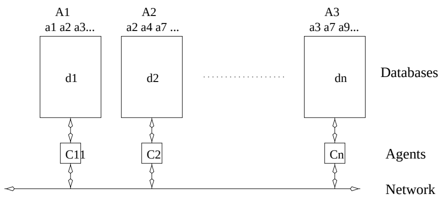

Figure 1: Databases

The case of distributed databases and their associated constraints can be represented as shown in Figure-1. We have n databases located at n different nodes of a network. We model each database site by a relation present at that site and Figure-1 shows relation d_i at the i^{th} site. Each database d_i is represented by an agent C_i that obtains summaries from its database and exchanges them with agents for other databases. Each agent is capable of initiating and completing a pattern discovery task by exchanging summaries with other agents.

The set of attributes contained in relation d_i is represented by A_i. Our discussion in this paper is limited to databases containing nominal valued attributes only. For any pair of relations d_i and d_j the corresponding sets A_i and A_j may have a set of shared attributes given by S_{ij}. That is,

$$S_{ij} = A_i \cap A_j \qquad (1)$$

The dataset D in which patterns are to be discovered is a subset of that set of tuples which would be generated by a *Join* operation performed on all the relations $d_1 \ldots d_n$. However, the tuples of D cannot be made explicit at any one site because data from d_is cannot be transferred to other sites. The tuples of D, therefore, must remain only implicitly defined. This inability to make explicit the individual tuples of D is the most severe limitation of the constrained set of databases.

To facilitate pattern discovery in the implicitly defined set of tuples of D we define a set S that is the union of all the intersection sets defined above. That is,

$$S = \cup_{i,j\ ,\, i \neq j} \quad S_{ij} \qquad (2)$$

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 formed by enumerating all possible combinations of values for attributes in set S.

Pattern Discovery Task

A pattern discovery task for distributed databases can be performed in one of the following ways:

1. Transfer all relevant relations to a single site and perform a *Join* operation to create a single relation. Then run the pattern discovery algorithm using this single table.
2. Decompose the computations of the pattern discovery algorithm; perform the decomposed parts at individual database sites; transmit the results back to the site performing discovery; and then compose the responses from individual sites to create the result.

For our constrained situation the first option is clearly ruled out. We must devise decompositions of pattern discovery algorithms such that the results produced are identical to those that would have been obtained by the first option. The decomposed versions would have to work as follows:

1. Learning task is initiated at a site called *Learner* which can be any one of the n database sites as shown in Figure-1, or possibly any other authorized site.

2. Attribute names in all the A_i sets, and as a consequence the relation *Shareds*, are known to the site initiating the pattern discovery task.
3. The *learner* site sends requests to various sites for statistical summaries about their respective d_is.
4. The *learner* site composes responses from various sites and constructs the descriptions of discovered patterns.

To demonstrate the decomposability of pattern discovery algorithms and computations we select a simple decision-tree induction algorithm and present an adaptation for all its steps for the case of constrained distributed databases. We briefly describe some aspects of the decision tree-induction algorithm here 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

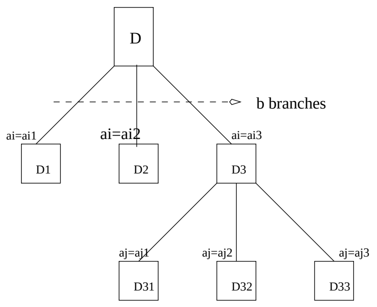

Figure 2: Building a Decision Tree

Various tree induction algorithms including ID3 and others (Brei 84; Quin 86) start by considering the complete dataset D belonging at the root of the tree and then repeating the following steps until all or a large majority of tuples at each leaf node of the tree belong to some unique class (Value of the *Target-Attribute*).

1. Pick one such dataset at a leaf node a majority of 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.

Figure-2 shows how the above steps are repeated to construct the tree. It is desirable to keep the height of the tree as small as possible. A heuristic that is used to keep the height of the tree on the smaller side selects that attribute a_i in Step-2 which minimizes the average informational entropy of the partitions formed in step-3. The value of this average entropy is computed as:

$$E = \sum_{b=1}^{m}(\frac{N_b}{N_t} \times (\sum_{c} -\frac{N_{bc}}{N_b} log_2 \frac{N_{bc}}{N_b}) \qquad (3)$$

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 tree induction algorithm described in the above section requires an explicit set of tuples at each node of the tree. This set is used for the following:

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. In the following section we consider various aspects of the tree induction algorithm and present the version adapted for the implicit 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 belonging to leaf nodes of the tree can be represented by storing a *partition identity number* along with each tuple in 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 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 the constrained environment.

Selecting the 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:

$$N_t = \sum_{J_{sk}} \cdots \sum_{J_{s2}} \sum_{J_{s1}} (\prod_{t=1}^{n} (N(d_t)_{sub1})) \qquad (4)$$

where the subscript *sub1* is: $[S1 = S1_{J_{s1}}], [S2 = S2_{J_{s2}}] \ldots, [Sk = Sk_{J_{sk}}]$. In this expression $S1, S2, \ldots Sk$ are the k members of set S defined by expression 2 above; $J_{S1}, J_{S2}, \ldots J_{Sk}$ 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 $(N(d_t)_{sub1})$ is the count in relation d_t of those tuples that satisfy the conditions stated in subscript $sub1$.

It can be seen that the expression for N_t is in the *sum-of-products* form. Each term in the product is the count of tuples satisfying condition *sub1* in a d_i. The resulting product produces the number of distinct tuples that would be contributed to the imagined *Join* of all d_is for the sharing condition specified by $sub1$. The relation *Shareds* contains tuples specifying all the different ways d_is can have shared attribute values. The summation in the above expression picks up each tuple of the relation *Shareds* as value for the $sub1$ and sums up the product terms obtained for each $sub1$.

This expression, therefore, simulates the effect of a *Join* operation on all the n sites without enumerating the tuples. The simulation only computes the count of tuples that would exist in various partitions of D.

A very desirable aspect of the particular decomposition of N_t given above is that each product term $(N(d_t)_{sub1})$ can also be easily translated into an SQL query of the form:

Select count (*) where *desired-partition* and *sub1*

and shipped to the site containing relation d_t. The expression *desired-partition* above states the conditions along the decision tree path leading up to the dataset being partitioned. For each tuple in relation *Shareds* we have to send the above query to each of the 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:

$$N_b = \sum_{J_{sk}} \cdots \sum_{J_{s2}} \sum_{J_{s1}} (\prod_{t=1}^{n} (N(d_t)_{sub2})) \qquad (5)$$

where the subscript $sub2$ is: $sub2 = [S1 = S1_{J_{s1}}], [S2 = S2_{J_{s2}}] \ldots, [Sk = Sk_{J_{sk}}], [B = B_{J_B}]$. 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.

$$N_{bc} = \sum_{J_{sk}} \cdots \sum_{J_{s2}} \sum_{J_{s1}} (\prod_{t=1}^{n} (N(d_t)_{sub3})) \qquad (6)$$

where the subscript $sub3$ is: $[S1 = S1_{J_{s1}}], [S2 = S2_{J_{s2}}] \ldots, [Sk = Sk_{J_{sk}}], [B = B_{J_B}], [C = C_{J_C}]$. 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 site 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.

Handling Exception Tuples: The method described above creates an implicit tuple space which is equivalent to the cross-product obtained from the n explicitly known relations $d_i, \ldots, d_n$. All tuples formed in this implicit space may not be valid data combinations. Those tuples known to be unacceptable combinations can be excluded from consideration by the above decomposed algorithm as follows. The learning site maintains a relation called *Exceptions* containing all the attributes of the n datasets. All known unacceptable tuples are stored in this relation. The product terms in the decompositions for N_t, N_b, and N_{bc} can be modified to discount the matching tuples found in the relation *Exceptions*. That is, the decomposition for N_t would become:

$$N_t = \sum_{J_{sk}} \cdots \sum_{J_{s2}} \sum_{J_{s1}} (\prod_{t=1}^{n} (N(d_t)_{sub1}) - Nexc_{sub1}) \qquad (7)$$

where $Nexc_{sub1}$ is the number of tuples in relation *Exceptions* that match the condition $sub1$.

Complexity

The cost of working with an implicitly specified set of tuples can be measured in various ways. One cost model computes the number of messages that must be exchanged among various sites. Complexity for distributed query processing in databases has been discussed in (Wang 96) and the cost model used is total data transferred for answering a query. In our case the amount of data transferred in each message is very little (statistical summaries etc.) but it is the number of messages to be exchanged that grows rapidly. We derive below an expression for the number of messages that need to be exchanged for dealing with the implicit et of tuples.

The exchange of messages occurs predominantly for computing the entropy values. An estimation of the number of messages exchanged while computing entropy values can be obtained as follows. Let us say:

- There are n relations, $d_1 \dots d_n$, residing at n different network sites.
- There are k attributes in set S. Each attribute in this set appears at more than one site.
- There are m distinct attributes in all the sets $(\cup_{i=1}^{n} A_i)$ combined.
- There are l possible discrete values for each attribute in set S.

The number of tuples in relation *Shareds* is l^k because it contains all possible combinations of values for its attributes.

From the previous section we know that each product term requires an exchange of n messages between the learning site *learn* and the remote sites.

For a dataset at depth level d in the decision tree we need to compute $m - d$ entropy values for selecting the most promising attribute. This is because for a dataset at level d, d attribute-value tests have already been made at its ancestor nodes, leaving only $m - d$ attributes as candidates for further partitioning the dataset.

For computing an entropy value according to the expression given in equation 3 above we need to compute:

- one N_t count;
- l N_b counts; and
- l^2 N_{bc} counts.

The expression for N_t contains l^k sum terms (one for each tuple in relation *Shareds*) and each sum term consists of n product terms. Therefore computation of N_t requires $n * l^k$ messages to be exchanged among sites. Using similar arguments it can be determined that the computations of N_b and N_{bc} require $n * l^{k+1}$ and $n * l^{k+2}$ message exchanges respectively.

It is possible to compute the entropy values by requesting from remote sites only the N_{bc} counts because it is possible for the learning site to locally construct N_bs and N_t by appropriately summing the various N_{bc} counts.

So, computation of each entropy value requires an exchange of $n * (l^{k+2})$ messages. For a dataset at depth d in the decision tree the number of messages exchanged would be $(m - d) * n * (l^{k+2})$. If we assume that on the average the decision tree is akin to a filled $l - ary$ tree with p levels then the total number of exchanges would be:

$$(n * l^{k+2}) \sum_{d=0}^{p} (m - d) * l^p \qquad (8)$$

The above expression does not take into account some tuples of *Shareds* that may not be usable for partitioning a dataset because of an attribute in S having been used for partitioning at an ancestor node of the dataset. That exception does not affect the order of the above expression and it can be seen that the number of messages grows exponentially with the number of shared attributes and also with the depth of the induced decision tree.

The above expression gives an estimate of the number of messages to be exchanged. Since the amount of information transferred in each message is constant, for any particular dataset and expected tree depth we can obtain the estimated data transfer. This can be compared to the amount of data to be transferred if the whole relations were to be transferred to a single site.

Validation and Results

The above described adaptation can be easily verified by substituting the decomposition expressions for N_t, N_b and N_{bc} into the original expression for entropy in equation 3. A few algebraic steps lead one to the needed simplification.

The above adaptation was also implemented by us on a network of PCs in a laboratory setting and tested against a number of small databases Decision trees were generated by (i) bringing all the relations to one site and performing a *Join* and (ii) by keeping them at their respective sites and using our decomposition of computations. The resulting decision trees for both the cases were identical. A comparison of the elapsed-time for a run is as follows:

- Three databases on different sites, one shared attribute for every pair of tables, all binary values attributes, 30 tuples in the implicit *join* of all the tables, and the average depth of the induced decision tree is 4 levels. Elapsed time when all tuples are stored in one table was 5 seconds; Elapsed time when tuples were stored in three tables containing fewer attributes each was 206 seconds.

Adaptability of Other Algorithms

The development of the above adaptation has provided the following insights about the adaptability of other clustering and pattern discovery algorithms for constrained distributed databases.

Many statistical pattern discovery algorithms determine averages of attribute values for representing prototypes for various classes. The decomposed computation for determining the average value of an attribute B in an implicitly specified dataset is as follows:

$$Avg_B = (\frac{1}{N_t}) * \sum_{j=1}^{l} (B_j * N(B_j)) \qquad (9)$$

where N_t is the total number of tuples in the dataset, $B_1 \ldots B_l$ are the possible discrete numeric values of attribute B, and $N(B_j)$ is the count of tuples in the dataset in which B takes the value B_j. The count N_t can be determined as described in expression 4 above. The count $N(B_j)$ can be determined as:

$$N_{B_j} = \sum_{J_{sk}} \cdots \sum_{J_{s2}} \sum_{J_{s1}} (\prod_{t=1}^{n} (N(d_t)_{sub4})) \qquad (10)$$

where $sub4$ is: $[S1 = S1_{J_{s1}}], [S2 = S2_{J_{s2}}] \ldots, [Sk = Sk_{J_{sk}}], [B = B_j]$ and all other symbols have the same meanings as described for equation 4. The capability to compute averages from implicitly stated datasets can easily be extended to compute variances, attribute-weightings, and some other statistical summaries. This demonstrates that many statistics based pattern discovery and clustering algorithms can be adapted for the implicitly stated datasets. Also, instead of informational entropy, many other selection functions have been suggested in (Ming 89a) and all those that depend on such statistical measures can be adapted for implicit datasets.

Algorithms in which presentation of explicit examples (tuples) to the learning agent is a must are not adaptable for constrained distributed database situations. Training of neural nets and other gradient-descent based methods require presentation of individual examples to the net and in case of implicit tuple set these are not available. It is almost impossible to take computations performed within each neuron or a threshold unit in response to a training example, decompose them, and take respective components to individual databases specifying the implicit set of tuples.

Many incremental learning algorithms work by comparing the concept induced so far and the new/next tuple. The constraints of our distributed environment do not permit the next tuple to be made explicit and available to the learning site. However, a completely new decision tree or concept may be inferred, starting *ab initio*. Such incremental learning algorithms would not be adaptable to the case of implicit datasets.

Conclusion

We have considered the case of discovering patterns in those sets of databases that have constraints on transferring data out of their individual sites. They can only transmit statistical summaries to authorized sites. We have demonstrated the adaptability of an informational entropy driven decision tree induction algorithm for the constrained case. We have also discussed some general issues about adaptability of pattern discovery algorithms and have also discussed the types of algorithms that may or may not be adaptable to the constrained situations.

Acknowledgment: This research was supported in part by the National Science Foundation Grant Number IRI-9308868. We are thankful to Hari Chandana Nidumolu who ran the programs on various sets of data. We are also thankful to the reviewers for providing useful comments.

References

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.

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.