

Active Mining in a Distributed Setting ^{*}

S. Parthasarathy, S. Dwarkadas, M. Ogihara

Department of Computer Science

University of Rochester

Rochester, NY 14627-0226

{srini,sandhya,ogihara}@cs.rochester.edu

Abstract

Most current work in data mining assumes that the data is static, and a database update requires re-mining both the old and new data. In this article, we propose an alternative approach. We outline a general strategy by which data mining algorithms can be made *active* — i.e., maintain valid mined information in the presence of user interaction and database updates. We describe a runtime framework that allows efficient caching and sharing of data among clients and servers. We then demonstrate how existing algorithms for four key mining tasks: Discretization, Association Mining, Sequence Mining, and Similarity Discovery, can be re-architected so that they maintain valid mined information across i) database updates, and ii) user interactions in a client-server setting, while minimizing the amount of data re-accessed.

1 Introduction

As we enter the digital information era, one of the greatest challenges facing organizations and individuals is how to turn their rapidly expanding data stores into accessible knowledge. Digital data sources are ubiquitous and encompass all spheres of human endeavor: from a supermarket's electronic scanner to a world wide web server, from a credit card reader to satellite data transmissions. Organizations and individuals are increasingly turning to the extraction of useful information, referred to as data mining, from such databases. Such high-level patterns, or inferences, extracted from the data may provide information on customer buying patterns, on-line access patterns, fraud detection, weather trends, etc.

This work was supported in part by NSF grants CDA-9401142, CCR-9702466, CCR-9705594, CCR-9701911, CCR-9725021, and INT-9726724; DARPA grant F30602-98-2-0133; and an external research grant from Compaq.

A typical data mining technique can be thought of as an exploration over a huge user-defined pattern space, with the role of the data being to select patterns with a desired degree of confidence. The set of accepted patterns is a function of both the data and the user-defined pattern space (controlled by the input parameters). The data mining process tends to be interactive and iterative in nature, i.e., after observing the results from the first round of mining, the user may choose to repeatedly modify the input parameters, thereby affecting the set of accepted patterns. Also, since businesses are constantly collecting data, the data is also subject to change, again affecting the set of accepted patterns.

Most current techniques, which tend to be static in nature, simply re-execute the algorithm in the case of data updates or user interaction. There are several limitations to this approach. First, although many of these techniques have parallel solutions [20, 28, 4] that are efficient in storage, access, and scale, they are still computationally expensive. Second, re-executing the algorithm requires re-examining both the old and new data, and hence I/O continues to be a bottleneck. These problems are further exacerbated in applications such as electronic commerce and stock sequence analysis, where it is important to execute the query in real or near-real time, in order to meet the demands of on-line transactions. Also, more and more such applications are being deployed as client-server applications where the server is physically separate from the client machine. Such a setup is also common within an organization's intra-net when there may be several groups mining, perhaps with separate agendas, from a common dataset. Ensuring reasonable response times in such applications is made more difficult due to the network latency and server load overheads. This leads to the following challenge:

In order to meet the demands of such interactive applications, can existing algorithms be re-architected, making them efficient in the presence of user interactions and data updates, in a distributed client-server setting?

1.1 Proposed Solution

We refer to algorithms that maintain valid mined information in the presence of user interaction and database updates as *active* algorithms. The main challenge is to perform the active mining in a storage and time efficient manner. This paper describes a general strategy by which data mining tasks can be re-architected to work efficiently with the constraints outlined above.

We accomplish our objective by maintaining a *mining summary structure* across database updates and user interactions. On a database update, the revamped algorithm replaces accesses to the old data with accesses to the mining summary structure whenever possible. This ensures that information that is previously mined can be re-used when the database is updated. On a user interaction, the hope is that the mining summary structure can answer the query without accessing the original data.

The design criteria for such a summary structure are: i) it should allow for incremental maintenance as far as possible, i.e., the mining summary structure from the old data along with the data update should ideally be sufficient to produce the new summary structure, avoiding accesses to the old data as far as possible, ii) it should store sufficient information to address a wide range of useful user interactions, and iii) it should be small enough to fit in memory so that accessing it rather than the old data provides a significant performance gain.

While the above solution can potentially solve the active mining problem, deploying these algorithms efficiently in a distributed setting is non-trivial. In typical client-server applications, the client makes a request to the server, the server computes the result, and then sends the result back to the client. The query execution time is significantly influenced by the speed of the client-server link as well as the server load. Since the interactions in our applications are often iterative in nature, caching the aforementioned summary structure on the client side so that repeated accesses may be performed locally eliminates overhead and delays due to network latency and server load. In order for this to be an effective solution, the summary structure should not be very large (re-emphasizing point iii) above), and an efficient mechanism for communicating updates is required.

Such summary structure sharing requires efficient caching support. We have built a general-purpose framework called InterAct that facilitates the development of interactive applications. InterAct supports sharing among interactive client-server applications. The key to the framework is an efficient mechanism to facilitate client-controlled consistency and sharing of objects among clients and servers (this allows applications that can tolerate some information loss to

take advantage of this to increase efficiency by reducing communication). Advantages within the scope of our work include: the ability to cache relevant data on the client to support interactivity, the ability to update cached data (when the data changes on the server) according to application or user preferences while minimizing communication overhead, and the ability to extend the computation boundary to the client to reduce the load on the server. We use this framework to develop our applications.

1.2 Contributions

In this paper:

1. We describe a general methodology for creating *active* mining solutions for existing applications.
2. We present active mining solutions for discretization, association mining, sequence mining, and similarity discovery in a distributed setting.
3. We describe the InterAct framework, which, along with the changes to the algorithms for active mining, allows effective client-server distribution of the computation.

The next section presents the InterAct framework on top of which we implement our active mining algorithms. We also outline our general approach to the problem of making algorithms *active*. Sections 3 (Discretization), 4 (Association/Sequence Mining), and 5 (Similarity Discovery) describe the applications we look at and our specific approach to make each of them *active*. We present and evaluate our experimental results in Section 6. Section 8 details our conclusions.

2 InterAct Framework

InterAct is a runtime framework that presents the user with a transparent and efficient data sharing mechanism across disparate processes. The goal is to combine efficiency and ease-of-use. InterAct allows clients to cache relevant shared data locally, enabling faster response times to interactive queries. Further, InterAct provides flexible client-controlled mechanisms to map and specify consistency requirements for shared data.

In order to accomplish its goal of transparently supporting interactive applications, InterAct:

- Defines a data (structure) format for shared objects that is address space and architecture independent. Our implementation relies on the use of C++ and some programmer annotation to identify the relevant information about shared data to the runtime framework.
- Identifies, defines, and supports the different types of consistency required by such applications

(described in [16]). In many domains (electronic commerce, credit card transactions), the data that is being queried is constantly being modified. The rate at which these modifications are required to be updated in the client’s cached copy may vary on the basis of the domain, application, or specific user. This rate can be controlled by exploiting the appropriate consistency model to enhance application performance.

- Provides an underlying mechanism to transparently handle the consistency demands as well as complex object transfer requirements. The goal here is to reduce programming complexity by hiding as much of the underlying communication from the application developer as possible.

For more framework details and the consistency types supported, see [16].

2.1 Active Mining and InterAct

Interactive Client-Server Mining

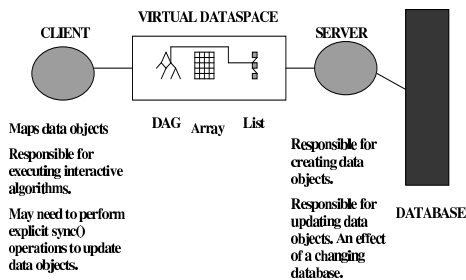


Figure 1: Client-Server Mining using InterAct

Since mining applications typically operate on large data sets that reside on a data server, communicating these datasets to the client would be infeasible. However, in this paper we show that it is possible to design useful summary structures for a range of mining applications, so that subsequent queries can operate on these summary structures. This summary data structure can be generated by the data distiller (server), and subsequently operated on by the client. Hence, the applications can be structured as shown in Figure 1, so that the data server is responsible for creating the data structure(s), mapping them onto a *virtual shared dataspace*, and subsequently keeping them up-to-date. The client can then map the data structure(s) from the *virtual shared dataspace* under a desired consistency model, thus enabling the client to respond to interactive queries without having to go to the data server.

In order for the above setup to be effective the summary data structure should satisfy three key properties. First, it should be able to directly answer a range of interactive queries without requiring access to the data as far as possible. This criterion minimizes the client-server communication, as well as server load. Second, the summary structure should be incrementally maintainable. This criterion ensures that changes to the data can be rapidly reflected in the summary structure. Third, the summary structure should not be very large as otherwise communicating it to the client may be very expensive. In the ensuing sections, we describe how such summary structures can be designed for Discretization, Association and Sequence Mining, and Similarity Discovery.

3 2-D discretization

Discretization is the process by which the range of a given *base* attribute (or independent variable) is partitioned into mutually exclusive and exhaustive intervals based on the value of a related *goal* attribute (or dependent variable), whose behavior we would like to predict or understand. Discretization has primarily been used for summarization [8], as well as for growing a decision tree [19].

Typically, a single attribute is used as the decision variable. However, one can also consider extensions to more than one base attribute (e.g., $X > 5 \wedge Y < 6$) as long as the decisions remain simple. The need for this is often encountered where repetitive applications of the single-attribute discretization do not provide optimal results, while a single, integrated two-dimensional approach does. In [17], the partitioning of a two dimensional base attribute space is defined in terms of *control points*. A single control point partitions the base attribute space into 4 rectangular regions. The rectangular regions are induced by drawing lines through the control point that are perpendicular to the two axes. Two control points partition the base attribute into up to 9 regions. The effect of discretization is to approximate the behavior of the goal attribute as the same for all points in a region. The purpose of the algorithm is to find the position of the control point(s) that optimizes a given objective function.

The input to a discretization algorithm could be either the raw data or a joint probability density function (pdf) of the base and goal attributes derived from the data. Using the data directly eliminates errors associated with pdf estimation. However, using a pdf enables one to use more meaningful error metrics such as Entropy [9]. Second, it permits users to encode domain knowledge by altering the shape and type of kernel (normal, poisson, binomial, etc.) used for density estimation. Further, it lends itself to a client-server architecture where density

estimation could be done on the server and a compact representation shipped to the client.

Evaluating any pair of base attributes involves 2 steps: computing the three dimensional probability density (pdf) estimate (two base attributes and the goal attribute), and searching for the optimal (determined by the objective function: Classification Error) discretization.

3.1 Interactivity

The idea in interactive discretization is that an end user ought to be able to modify process parameters. The interactive features currently supported in our algorithm include: i) choosing from a set of algorithms (brute force search, approximate search), and metrics (entropy, error), to compute the optimal discretization, ii) changing the number of control points (1, or 2), iii) changing the position of control points, and iv) changing the parameters for pdf estimation.

Ideally, all these features need to be supported efficiently without excessive I/O or computation.

3.2 Summary Structure

As mentioned earlier, the summary structure required to support such interactions efficiently is the joint pdf $p(base_1, base_2, goal)$. This pdf is estimated at discrete locations. While several techniques exist to estimate the density of an unknown pdf, the most popular ones are histogram, moving window, and kernel estimates [7]. We use the histogram estimate described in [7]. The advantage of this estimate is that it can be incrementally maintained in a trivial manner (a histogram estimate is essentially the frequency distribution normalized to one). Moreover, the more complicated kernel estimates can easily be derived from this basic estimate [7].

With a few modifications, the histogram pdf estimate can handle each of the interactions described above. If there are n points in the estimate, computing the objective function for a given control point takes $O(n)$ time. Since the objective of discretization is to find the control point(s) that optimizes the given objective function, a brute force search will take $O(n^2)$ ($O(n^4)$ for two control points) time. Recently, we have shown that smarter methods can reduce this time complexity to $O(n)$ ($O(n^2)$ for two control points) with additional memory ($O(n)$). Alternatively, fast approximate searches like simulated annealing can be used to generate good discretizations quickly [17].

Changing the control point in small incremental ways (moving the control point along one of the two axes in unit steps), enables one to compute the new objective functions (entropy or error) in unit time at the cost of additional storage ($O(n)$). Changing the parameters for pdf estimation can also be done quickly using the histogram estimate [7].

In order for the summary structure to be cost effective, the size of the structure (n , the number of points in the estimate) must be small enough so that i) it can be cached easily on remote clients, and ii) it can allow for faster interactions. In the next section, we describe an experiment that explores the tradeoff between the size of the summary structure and the accuracy of the discretization obtained.

3.2.1 Accuracy vs. Summary Size

We evaluate the premise that it is possible to create a condensed representation of the data (probability density function) without serious degradation in the quality of discretizations obtained. This enables efficient client-server partitioning, and allows off-loading parts of the computation to another machine, thereby reducing the load on the server, and potentially improving interaction efficiency.

Experimentally, we have found that the number of points at which the pdf needs to be evaluated (determined by grid size) without significant quality degradation is not large. Our results are summarized in Figure 2. The results reported are for two synthetic data sets, XOR and LL. These are described in [17]¹. Both have 100,000 instances and 2 base attributes. It is easy to see that the quality (error minimization) of discretization does not improve much beyond a grid size of 64^2 .

Data Set	Grid Size	Error
XOR	16^2	18.87%
XOR	32^2	18.13%
XOR	64^2	17.90%
XOR	128^2	17.86%
LL	16^2	12.47%
LL	32^2	12.2%
LL	64^2	12.2%
LL	128^2	12.3%

Figure 2: Effect of grid size for pdf evaluation on quality of results

The resulting summary structure for discretization satisfies all the properties for efficient active mining: it is bounded and small, it can be used to handle a wide range of client queries without going back to the original database, and it can be incrementally maintained.

4 Association/Sequence Mining

In this section, we consider two of the central data-mining tasks, i.e., the discovery of association rules

¹Available via anonymous ftp from <ftp.cs.rochester.edu/pub/u/srini>

and the discovery of sequences. The discussion below follows [28] (associations) and [27] (sequences).

The problem of mining association rules over *basket* data was introduced in [2, 3]. It can be formally stated as: Let $\mathcal{I} = \{i_1, i_2, \dots, i_m\}$ be a set of m distinct attributes, also called *items*. Each transaction T in the database \mathcal{D} of transactions, has a unique identifier, and *contains* a set of items, such that $T \subseteq \mathcal{I}$. An *association rule* is an expression $A \Rightarrow B$, where $A, B \subset \mathcal{I}$, are sets of items called *itemsets*, and $A \cap B = \emptyset$. Each itemset is said to have a *support* S if $S\%$ of the transactions in \mathcal{D} contain the itemset. The association rule is said to have *confidence* C if $C\%$ of the transactions that contain A also contain B , i.e., $C = S(A \cup B)/S(A)$, i.e., the conditional probability that transactions contain the itemset B , given that they contain itemset A .

Sequence Mining can be thought of as association mining over temporal datasets. A *sequence* is an ordered (over time) list of nonempty itemsets. A sequence of itemsets $\alpha_1, \dots, \alpha_n$ is denoted by $(\alpha_1 \mapsto \dots \mapsto \alpha_n)$. The *length* of a sequence is the sum of the sizes of each of its itemsets. The database is divided into a collection of customer sets where each customer set contains the set of transactions that customer is involved in in order of occurrence. For a database D and a sequence α , the *support* or *frequency* of α in D , is the number of customers in D whose sequences contain α as a subsequence. A rule $A \Rightarrow B$ involving sequence A and sequence B is said to have *confidence* c if $c\%$ of the customers that contain A also contain B .

The basic approach to mining associations and sequences is a two step iterative approach. First, identify the set of candidate associations/sequences for a given number of items. Second, compute the set of associations/sequences from the candidate set that meet the user-specified criteria, forming the basis for the candidates in the next iteration (adding one to the number of items considered). We use the ECLAT [28] (associations) and SPADE [27] (sequences) algorithms as the basis for our work.

4.1 Interactivity

The idea in interactive association mining (or interactive sequence mining) is that an end user be allowed to query the database for association rules at differing values of support and confidence. The goal is to allow such interaction without excessive I/O or computation. Interactive usage of the system normally involves a lot of manual tuning of parameters and re-submission of queries that may be very demanding on the memory subsystem of the server. In most current algorithms, multiple passes have to be made over the database for each $\langle \text{support}, \text{confidence} \rangle$ pair. This leads to unacceptable response times for online queries. Our approach to the problem of support-

ing such queries efficiently is to create pre-processed summaries that can quickly respond to such online queries.

A typical set of queries that such a system could support include: i) **Simple Queries:** identify the rules for support $x\%$, confidence $y\%$, ii) **Refined queries:** where the support value is modified ($x + y$ or $x - y$) involves the same procedure, iii) **Quantified Queries:** identify the k most important rules in terms of support, confidence pairs or find out for what support/confidence values can we generate exactly k rules, iv) **Including Queries:** find the rules including itemsets i_1, \dots, i_n , v) **Excluding Queries:** compute the rules excluding itemsets i_1, \dots, i_n , and vi) **Hierarchical Queries:** treat items i_1, \dots, i_n , as one item and return the new rules.

4.2 Summary Structure

In [1], the concept of an Association Lattice L is defined. An association X is said to be adjacent to an association Y if one of them can be obtained from the other by adding a single item. Specifically, an association X is a parent of Y if Y can be obtained by adding one item to X . We allow directed edges from parents to children. It is then easy to see that if a directed path exists from a vertex V to a vertex U then $V \subset U$. Further, each node in the lattice is weighted by the support S of the given association it represents. The sequence lattice is obtained in a similar manner.

The preprocessing step of the algorithm involves computing such a lattice for a small enough support S_{min} , such that all future queries will involve a support S larger than S_{min} . If the above holds, and given such a lattice, we can produce answers to all but one (**Hierarchical queries**)² of the queries described in the previous section at interactive speeds without going back to the original database. This is easy to see as all of the queries will basically involve a form of pruning over the lattice. A lattice, as opposed to a flat file containing the relevant associations/sequences, is an important data structure as it permits rapid querying for associations [1] and sequences [18]. This lattice can also be incrementally maintained for associations [25] and sequences [18]. Due to limited space, we do not describe it here.

4.2.1 Accuracy vs. Summary Size

Unlike in discretization, the size of the summary structure is not bounded for a choice of S_{min} . It depends on the data and the choice of S_{min} . For small enough S_{min} the lattice can be very large (larger than the original data itself in some cases!). However, for most practical cases (e.g., $S_{min} = 0.05\%$, dataset

²These queries require recomputation on the server. However, because of the way we access the data, and the way it is stored we limit accesses to the old data.

= 170MB) the resulting lattice (4MB) is manageable and the applications can benefit from client side caching of the data structure.

The summary structure satisfies the three properties for efficient active mining: it can be used to handle a wide range of client queries without going back to the original database, it can be incrementally maintained, and for most practical instances, the size of the lattice is not too large.

5 Similarity Discovery

Similarity is a central concept in data mining. Discovering the similarity among attributes enables reduction in dimensions of object profiles as well as provides useful structural information on the hierarchy of attributes. Das et al [6] proposed a novel measure for attribute similarity in transaction databases. The similarity measure proposed compares the attributes in terms of how they are individually correlated with other attributes in the database. The choice of the other attributes (called the probe set) reflects the examiner’s viewpoint of relevant attributes to the two. Das et al show that the choice of the probe set strongly affects the measurement.

There are some limitations to this basic approach. First, when the examiner does not know what the relevant attributes are, their approach offers no solutions. A brute force search would be impractical. Second, the approach limits the probe elements to singleton attributes and does not allow boolean attribute formulae.

If one is not interested in probe attributes of small frequency, an alternative approach can be to use the associations generated by an algorithm such as ECLAT [28] as the probe set. The similarity metric between attributes “a” and “b” can be defined in terms of association sets (A, the set of all associations involving “a” but excluding “a”. For instance if “adl” were a valid association, then “dl” would belong to the set A. Similarly, B, the set of all associations involving “b” but excluding “b”.) and their associated supports (sup):

$$Sim(A, B) = \frac{\sum_{x \in A \cap B} \max\{0, 1 - \alpha |\sup_A(x) - \sup_B(x)|\}}{\|A \cup B\|}$$

where α is a user-defined normalizing variable that defaults to one. This approach is fast and scales well in practice. It also permits boolean attribute formulae (a limitation of the Das et al [6] approach) as part of the probe set. Since we use associations as the probe set, this approach can also be used to measure the similarity between different homogeneous datasets and is not limited to measuring attribute similarity.

5.1 Interactive Similarity

In our approach the following interactions are currently supported: i) **Boolean Pruning**: Prune the

probe space (association sets) to only those parts of the association sets that satisfy a given boolean formula, ii) **Identifying influential attributes**: Identify the (set of) probe attribute(s) that contribute most to the similarity/dissimilarity metric, and iii) **Changing the minimum support**.

5.2 Summary Structure

In this application, the summary structure required is the association lattice described in Section 4.2. For dataset similarity, the association lattices of both datasets are required.

Once the association lattices are obtained, the basic algorithm computes the similarity measure. The different interactions are supported as follows.

For Boolean Pruning, the algorithm basically prunes both lattices according to the boolean formula, yielding sets A^1 and B^1 . The similarity metric is then recomputed by replacing A with A^1 and B with B^1 . For Identifying Influential Attributes, for a singleton attribute “l”, the algorithm prunes out all elements in the association lattice that do not contain “l”. It then computes the similarity between the two datasets. This step is repeated for all singleton attributes and the resulting similarities are sorted. The higher ranked attributes influence similarity while the lower ranked attributes influence dis-similarity.

5.2.1 Accuracy vs. Summary Size

Like association mining, the size of the summary structure is not bounded for a choice of S_{min} . However, unlike association mining, similarity discovery is less attuned to this choice. Fixing an S_{min} apriori is an acceptable solution for the purpose of computing similarities. This limits the size of the data structure, ensuring that the three properties for active mining are satisfied for this application.

6 Experimental Evaluation

In order to completely evaluate all aspects of our work, we evaluate the impact of our summary structure with respect to three qualities; its interactive performance, its incremental performance, and the efficacy of client-server work distribution by caching the summary structure and executing queries locally. We first describe in detail the queries we evaluated on each of the applications, and their associated datasets.

6.1 Application Properties

We executed a series of queries for each application. For association mining, we executed a simple query (find rules with support x%) followed by a quantified query (find the 400 most important associations). For sequence mining, we executed a combination of including (all sequences including item x) and excluding sequences (all sequences excluding item y). For discretization, we executed the base algorithm

(find the optimal discretization) and then asked the system to move the control points to a new location and compute the new error. In similarity discovery, we asked the system to compute the pairwise similarities for four datasets and then asked it to recompute the similarities under boolean pruning, as well as identify the most influential attributes.

Each of the queries was executed on an appropriate dataset. For association mining, we executed our queries using a synthetic dataset generated adopting the methodology described in [3]. The dataset we used (T10.I6.D3200) contained on average 10 items per transaction, and 3,200,000 transactions. The size of the resulting dataset is 140MB.

For sequence mining, we executed our queries using a synthetic dataset generated by a similar procedure [27]. The dataset we used (C250.I6.T10) contained on average 10 transactions per customer, and 250,000 customers, where each transaction is variable in length. The size of the resulting dataset is 55MB.

For discretization, we used the XOR dataset [17]. The dataset has 100,000 instances and 3 attributes (two base, one goal) per instance. There are 2 categories for the goal attribute, C0 and C1. The size of the resulting dataset is 4MB.

For similarity discovery, we executed our queries against a real dataset, the Reuters dataset³. The data set consists of 21578 articles from the Reuters newswire in 1987. Each article has been tagged with keywords. The size of the dataset is 27MB. For our evaluation, we represented each news article as a transaction with each keyword being an item.

For each of the applications considered: Associations, Sequences, Discretization and Similarity, the size of the summary structures were 3.3MB, 1.0MB, 0.5MB, and 2.0MB, respectively. It is easy to see that in all of the cases the summary structure is a significant reduction from the original dataset and is small enough to enable effective *active* mining.

6.2 Active Mining Performance

We summarize the results on interactive and incremental mining performance here. In this paper, we have described a methodology for off-loading the interactive querying feature onto client machines as opposed to executing on the server, and shipping the results to the data mining client. In order to clearly demonstrate the effectiveness of this approach, we wanted to compare executing queries on slower clients (143Mhz and 270Mhz UltraSparcs) using the designed summary structure versus recomputing the result on the fastest server we have available (600MHz Alpha Station 4100s).

The results of the experiment are shown in Table 1. The first column corresponds to the application we

evaluated, the second column contains the execution time of the query on a 143Mhz client using the appropriate summary structure, the third column similarly contains the execution time on a 270Mhz client, and the fourth column represents the execution time of running the query from scratch on the 600 MHz Alpha, without the use of the summary structure. For all of the applications, the execution time with the summary structure is orders of magnitude faster than re-executing the query from scratch. This is despite the fact that the results obtained for re-executing the query without the summary structure is on a much faster server.

The fifth column of our table represents the speedup obtained from maintaining the structure incrementally, rather than re-creating it on a database update. This part of the experiment was also performed on the 600MHz Alpha Station 4100. The (*incr* 5%) in the column header corresponds to the increment size. The datasets described in the previous sections are divided into two partitions, one containing 95% of the transactions (or instances) and the other containing the remaining 5%. The first partition we assume is the original dataset, while the second partition is treated as the increment dataset. The speedup numbers in this column compare the speedup of using an incremental algorithm as opposed to re-executing the algorithm on the entire (original + increment) data (column 4). The performance gains from the incremental approach ranges from good (speedup of 7) to excellent (speedup of 18). As expected, incrementally maintaining the summary structure for the discretization application results in the best speedup since it is the easiest to maintain.

6.3 Distributed Performance

In typical client-server applications, the client makes a request to the server, the server computes the result, and then sends the result back to the client. Since the interactions in our applications are often iterative in nature, caching the summary data structure on the client side so that repeated accesses may be performed locally can potentially improve query execution times.

We present results on the efficacy of caching the summary structure in a distributed environment consisting of SUN workstations connected by 10 or 100 Mbps switched Ethernet. The clients in each application interact with the server by sharing the summary data structures with the server. The server creates the summary data structure and updates it corresponding to changes in the database (which we simulate). The potential gain from client-side caching depends on a number of factors: the size of the shared data, the speed of the client, the network latency, the server load, and the frequency of data modification. We evaluate the effect of each of these factors.

³www.research.att.com/~lewis/reuters21578.html

Application	Client (143Mhz)	Client (270Mhz)	Recompute	Incremental Speedup (<i>incr</i> 5%)
Association Mining	2.4	1.5	540.7	10
Sequence Mining	0.58	0.35	150	7
Discretization	0.87	0.55	505.8	18
Similarity	0.35	0.11	10	10

Table 1: Active Mining Performance: Execution Times in seconds

Application	Client(143)				Client(270)					
	CSC	SSRC		L-SSRC		CSC	SSRC		L-SSRC	
Ethernet (Mbps)		10	100	10	100		10	100	10	100
Association	2.4	4.05	1.6	7.2	2.5	1.5	2.5	1.4	5.1	2.3
Sequence	0.58	0.85	0.55	1.35	0.86	0.35	0.63	0.5	1.18	0.73
Discretization	0.87	1.35	0.67	2.75	1.08	0.55	0.94	0.6	1.6	0.98
Similarity	0.35	1.5	0.55	2.7	0.98	0.11	0.9	0.37	2.4	0.94

Table 2: Time (in seconds) to Execute Query in a Distributed Environment

We ran each of our applications under the following scenarios:

1. Client-Side Caching (CSC): the client caches the summary structure and executes the query on the local copy (the execution times reported here do not reflect the time to communicate the summary structure, which gets amortized over several queries).
2. Server Ships Results to Client (SSRC): the client queries the server and the server ships the results back to the client. This scenario is similar to the use of an RPC mechanism. In order to better understand the impact of server load, we varied the number of clients serviced by the server from one (SSRC) to eight (Loaded-SSRC).

We measured the time to execute each query under both scenarios. We evaluated each scenario on a range of client machines, from an UltraSparc (143Mhz) machine to an UltraSparc IIi (270Mhz). In each case, our server was an 8-processor 336 MHz UltraSparc II machine. Results are presented in Table 2 for these scenarios under two different network configurations. We varied the network configuration by choosing clients that are connected to the server via a 10 Mbps or a 100 Mbps Ethernet network. For each of the applications considered: Associations, Sequences, Discretization, and Similarity, the size of the results shipped by the server (total data communicated) were 1.5MB, 0.25MB, 0.5MB and 0.75MB respectively.

The results in Table 2 show that client-side caching is beneficial for all but a few of the cases. In particular, the following trends are observed. Client-side caching is more beneficial under the following scenarios: the network bandwidth is low (speedups from client-side caching under the 10Mbps configuration

are larger (1.5 to 23) than the 100Mbps numbers (0.6 to 9)), the server is loaded (comparing the L-SSRC column (speedups of 1.1 to 9) with the SSRC column (speedups of 0.6 to 3.5) with a 100 Mbps network), the client is a fast machine (comparing the columns involving the 270Mhz clients versus the 143Mhz clients), or the time to execute the query is low (comparing the row involving the similarity discovery with the row involving association mining). In other words, the benefits from client-side caching are a function of the computation/communication ratio. The lower the ratio, the greater the gain from client-side caching. The fact that InterAct enables such caching is very useful for such applications especially when deployed on the Internet.

In addition, the client maps the shared summary data structures using one of the set of consistency models provided by InterAct. Choosing the right consistency model for a given application depends on its tolerance for stale data. Updates are then transmitted to the client according to the consistency model chosen. Results obtained show that the average update times are several orders of magnitude faster than existing approaches such as RPC. For a detailed analysis of our update protocol and results pertaining to these applications, see [16].

7 Related Work

7.1 Distributed Data Mining Systems

Several systems have been developed for distributed data mining. The JAM [22](Java Agents for Meta-learning) and the BODHI [13] system assume that the data is distributed. They employ local learning techniques to build models at each distributed site, and then move these models to a centralized location. The models are then combined to build a meta-

model whose inputs are the outputs of the various models and whose output is the desired outcome. The Kensington [12] architecture treats the entire distributed data as one logical entity and computes an overall model from this single logical entity. The architecture relies on standard protocols such as JDBC to move the data. The Id-Vis [23] architecture is a general-purpose architecture designed with data mining applications in mind to work with clusters of SMP workstations. Both this system and the Papyrus system [11] are designed around data servers, compute servers, and clients. The Id-Vis architecture explicitly supports interactivity through the interactive features of the Distributed Doall programming primitive. However, the interactions supported are limited to partial result reporting and bare-bones computational steering.

Our work is complementary to the above distributed data mining systems. Their focus is on how to build data mining systems or specific data mining applications when the data and processing capacity is distributed. Our focus is on making existing algorithms *active*.

7.2 Incremental Mining

In [5], an incremental algorithm for maintaining association rules is presented. A major limitation of this algorithm is that it may require $O(k)$ database (original plus increment) scans, where k is the size of the largest frequent itemset. In [10], two incremental algorithms were presented – the *Pairs* approach stores the set of frequent 2-sequences, while the *Borders* algorithm keeps track of the frequent set and the negative border. An approach very similar to the Borders algorithm was also proposed in [25].

There has been almost no work addressing the incremental mining of sequences. One related proposal in [26] uses a dynamic suffix tree based approach to incremental mining in a single long sequence. However, we are dealing with sequences across different customers, i.e., multiple sequences of sets of items as opposed to a single long sequence of items. To the best of our knowledge there has been no work to date on the incremental mining of discretization and similarity discovery.

7.3 Interactive Mining

A mine-and-examine paradigm for interactive exploration of associations was presented in [14]. The idea is to mine and produce a large collection of frequent patterns. The user can then explore this collection by the use of *templates* specifying what’s interesting and what’s not. They only consider inclusive and exclusive templates (corresponding to our **Including** and **Excluding** queries), whereas our approach handles a wider range of queries, in an efficient manner.

A second approach to exploratory analysis is to integrate the constraint checking inside the mining algorithm. One such approach was presented in [21]. Recently, [15] presented the CAP algorithm for extracting all frequent associations matching a rich class of constraints. Our approach relies on constraining the final results rather than integrating it inside the mining algorithm.

An online algorithm for mining associations at different values of support and confidence, was presented in [1]. Like their approach, we rely on a lattice framework to produce results at interactive speeds. Our approach relies on a different base algorithm [28] for generating associations and this allows us to compute a wider range of queries, as well as, compute such queries faster.

An interactive approach to discretization was presented in [24] for traditional one-dimensional discretization. They also use a probability density estimate of the base attribute to allow for certain user interactions in a manner similar to ours. However, their problem domain is much simpler than ours and therefore the interactive queries supported are relatively easier to compute. We are not aware of any such work on interactive mining, within the domain of sequence and similarity discovery.

Most of the incremental and interactive mining approaches tend to focus on isolated applications leading to a proliferation of solutions with little or no inter-operability. Our approach is the first that tries to integrate the incremental and interactive components in a distributed setting. Furthermore, we outline a general strategy for making mining algorithms *active* in such a setting.

8 Conclusions

In this paper, we described our approach to active data mining in a client-server setting. We presented a general method for creating efficient interactive mining algorithms, and in addition, demonstrated its efficacy in a distributed setting using the InterAct framework. We applied this general methodology to several data mining tasks: discretization, association mining, sequence mining, and similarity discovery.

To summarize our method, we maintain a *mining summary structure* that is valid across database updates and user interactions. On a user interaction, the mining summary structure can answer the query without re-accessing the actual data. On a database update, the amount of the original database that needs to be re-examined is minimized. Lastly, by caching the summary structure on the client using InterAct, we can eliminate overheads due to network latency and server loads.

Experimental results show that executing queries using the appropriate summary structure can improve

performance by several orders of magnitude. Furthermore, for all the applications considered, the summary structures can be incrementally maintained with up to an 18-fold improvement over re-creating the summary structures on a database update. Finally, up to a 23-fold improvement in query execution times was observed when the clients cache the summary structure and execute the query locally.

References

- [1] C. Aggarwal and P. Yu. Online generation of association rules. In *IEEE International Conference on Data Engineering*, February 1998.
- [2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Conf. Management of Data*, May 1993.
- [3] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In U. Fayyad and et al, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, Menlo Park, CA, 1996.
- [4] D. Cheung, J. Han, V. Ng, A. Fu, and Y. Fu. A fast distributed algorithm for mining association rules. In *4th Intl. Conf. Parallel and Distributed Info. Systems*, December 1996.
- [5] D. Cheung, J. Han, V. Ng, and C. Wong. Maintenance of discovered association rules in large databases: an incremental updating technique. In *12th IEEE Intl. Conf. on Data Engineering*, February 1996.
- [6] G. Das, H. Mannila, and P. Ronkainen. Similarity of attributes by external probes. In *Proceedings of the 4th Symposium on Knowledge Discovery and Data Mining*, 1998.
- [7] L. Devroye. A course in density estimation. In *Birkhauser: Boston MA*, 1987.
- [8] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. *12th ICML*, 1995.
- [9] U. Fayyad and K. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. *14th IJCAI*, 1993.
- [10] R. Feldman, Y. Aumann, A. Amir, and H. Mannila. Efficient algorithms for discovering frequent sets in incremental databases. In *2nd ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, May 1997.
- [11] R. Grossman, S. Bailey, S. Kasif, D. Mon, A. Ramu, and B. Malhi. Design of papyrus: A system for high performance, distributed data mining over clusters, meta-clusters and super-clusters. In *Proceedings of Workshop on Distributed Data Mining, alongwith KDD98*, Aug 1998.
- [12] Y. Guo, S. Rueger, J. Sutiwaraphun, and J. Forbes-Millot. Meta-learning for parallel data mining. In *Proceedings of the Seventh Parallel Computing Workshop*, 1997.
- [13] H. Kargupta, I. Hamzaoglu, and B. Stafford. Scalable, distributed data mining using an agent based architecture. In *KDD*, Aug 1997.
- [14] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *3rd Intl. Conf. Information and Knowledge Management*, pages 401–407, November 1994.
- [15] R. T. Ng, L. Lakshmanan, J. Jan, and A. Pang. Exploratory mining and pruning optimizations of constrained association rules. In *ACM SIGMOD Intl. Conf. Management of Data*, June 1998.
- [16] S. Parthasarathy and S. Dwarkadas. Shared state for client server applications. TR716, Department of Computers Science, University of Rochester, June 1999.
- [17] S. Parthasarathy, R. Subramonian, and R. Venkata. Generalized discretization for summarization and classification. In *PADD98*, January 1998.
- [18] S. Parthasarathy, M. Zaki, M. Ogihara, and S. Dwarkadas. Incremental and interactive sequence mining. TR715, Department of Computers Science, University of Rochester, June 1999.
- [19] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, Los Altos CA, 1993.
- [20] J. Shafer, R. Agrawal, and M. Mehta. Sprint: A scalable parallel classifier for data mining. In *22nd VLDB Conference*, March 1996.
- [21] R. Srikant, Q. Vu, and R. Agrawal. Mining Association Rules with Item Constraints. In *3rd Intl. Conf. on Knowledge Discovery and Data Mining*, August 1997.
- [22] S. Stolfo, A. Prodromidis, and P. Chan. Jam:java agents for meta-learning over distributed databases. In *KDD*, Aug 1997.
- [23] R. Subramonian and S. Parthasarathy. A framework for distributed data mining. In *Proceedings of Workshop on Distributed Data Mining, alongwith KDD98*, Aug 1998.
- [24] R. Subramonian, R. Venkata, and J. Chen. A visual interactive framework for attribute discretization. In *Third International Conference on Knowledge Discovery and Data Mining*, pages 82–88, 1997.
- [25] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka. Incremental updation of association rules. In *KDD97*, Aug 1997.
- [26] K. Wang. Discovering patterns from large and dynamic sequential data. *J. Intelligent Information Systems*, 9(1), August 1997.
- [27] M. J. Zaki. Efficient enumeration of frequent sequences. In *7th Intl. Conf. on Information and Knowledge Management*, November 1998.
- [28] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New parallel algorithms for fast discovery of association rules. *Data Mining and Knowledge Discovery: An International Journal*, 1(4):343–373, December 1997.