1 Introduction

Data mining has become increasingly important as a key to analyzing, digesting and understanding the flood of digital data. Achieving this goal requires scaling mining algorithms to large databases. Many classic mining algorithms require multiple database scans and/or random access to database records. Work in this area focuses on overcoming limitations imposed when scanning a large database multiple times or accessing records at random is costly or impossible, as well as innovative algorithms and data structures to speed up computation. In this paper, we focus on illustrating scalability principles by highlighting some of the key innovations and techniques.

2 Prediction Methods

We focus on two classes of predictive modeling algorithms: decision-tree methods and support-vector machines.

The input into a predictive modeling algorithm is a dataset of training records. The goal is to build a model that predicts a designated attribute value from the values of the other attributes. Many predictive models have been proposed in the literature: neural networks, Bayesian methods, etc. There are excellent overviews of predictive methods in [9, 5, 7, 6].
Figure 1: Magazine Subscription Example Classification Tree

2.1 Decision Tree Construction

Decision trees, are especially attractive in a data mining environment, since the resulting model is understandable by human analysts, their construction does not require any input parameters from the analyst, and no prior knowledge about the data is needed. Figure 1 shows an example decision tree. A record can be associated with a unique leaf node by starting at the root and repeatedly choosing a child node based on the splitting criterion at the current node. There exist excellent surveys of decision tree construction [3, 8].

Decision tree construction algorithms consist of two stages: a tree-building stage and a pruning stage. In the tree-building stage, most decision tree construction algorithms grow the tree top-down in the following greedy way. Starting with the root node, the database is examined by a split selection method to select a splitting criterion. Then the database is partitioned and the procedure is applied recursively. Sophisticated split selection methods have been developed. In the pruning stage, the tree constructed in the tree-building phase is pruned to control the size of the tree, and sophisticated pruning methods have been developed that select the tree that minimizes prediction errors.

The training database is accessed extensively while constructing the tree, and if the training database does not fit in-memory, an efficient data access method is needed to achieve scalability.

Many algorithms used the observation that only a small set of sufficient
statistics, e.g., various aggregate measures such as counts, is necessary for applying several popular split selection methods. The aggregated data is much smaller than the actual data. These sufficient statistics can be constructed in-memory at each node in one scan over the database partition that corresponds (i.e., satisfies the splitting criteria leading) to this node.

Although in many cases the sufficient statistics are quite small, there are situations where the sufficient statistics are about as large as the complete dataset. There have been several different algorithms developed for this case. One way of dealing with the size of the sufficient statistics is to observe that a large class of split selection methods searches over all possible split points over all attributes. The sufficient statistics at each step of this search are very small—small enough to fit in-memory. One way to utilize this observation is to create index structures over the training dataset that permit fast incremental computation of the sufficient statistics between adjacent steps of this search. For example, one class of algorithms vertically partitions the dataset, sorts each partition by attribute value, and then searches splitting criteria for each attribute separately by scanning the corresponding vertical partition.

Another way of dealing with the size of the sufficient statistics is to split the problem in two phases. In the first phase, we scan the dataset and construct sufficient statistics in-memory at a coarse granularity. Using the in-memory information, we prune large parts of the search space of possible splitting criteria due to smoothness properties (such as bounds on the derivative) of the splitting criteria. Then, in a second phase, we scan the dataset a second time and construct exact sufficient statistics only for the parts of the search space that could not be eliminated in the first phase of the algorithm. Variations of this idea only eliminate parts of the search space in the first phase with high probability, but then check their decisions in the second phase. Algorithms based on this two-phase approach appear to be the fastest known methods for classification tree construction.

2.2 Support Vector Machines

Support vector machines (SVMs) are powerful and currently popular approaches to predictive modeling. SVMs have had success in applications including: handwritten digit recognition, charmed quark detection, face detection and text categorization [4].

We describe SVMs in the context of classification where the attribute whose value is to be predicted (dependent attribute) has 2 possible values: 0 or 1. SVM classification is performed by a surface in the space of predictor
attributes that separates points with dependent attribute = 0 from those with dependent attribute = 1. An optimal separating surface is computed by maximizing the margin of separation (see Figure 2). The margin of separation is the distance between the boundary of the points with dependent attribute = 0 and the boundary of those with dependent attribute = 1. The margin is a measure of “safety” in separating the two sets of points. The larger the better. In the standard SVM formulation, computing the optimal separating surface requires solving a quadratic optimization problem.

The burden of solving the SVM optimization problem grows drastically with the number of training records. To reduce this burden, the method of chunking iteratively updates the separator parameters over chunks of training cases at a time. The size of a chunk is chosen so that it fits into main memory. To obtain optimal classification, chunks often need to be revisited, implying multiple passes over the data.

Data compression is also applicable to SVMs by applying the concept of squashing. First, the training records are clustered utilizing the likelihood profile of the data. Then the SVM is trained over the clusters, where each cluster is weighted by the number of data points in it.

Another approach to scaling SVMs involves reformulating the underlying optimization problem, resulting in efficient iterative algorithms.
falling into this category require the solution of a linear system of equations with size $m + 1$ at each iteration or deal directly with the Karush-Kuhn-Tucker optimality conditions to incrementally improve the classifier at each iteration. Here $m$ is the number of predictor attributes.

The SVM predictive function can be decomposed as the linear combination of functions of training data points (kernels). Projection methods attempt to approximate the combination of all training data with a subset of points. Some projection methods use $m$ randomly selected points on which to base the separating surface. Sparse, greedy matrix approximations try to determine the best $m$ points to use. For a detailed overview of these methods, see [10].

3 Clustering

The goal of clustering is to partition a set of records into several groups such that “similar” records are in the same group according to some similarity function between records, identifying similar sub-populations in the data. For example, given data about customers, their purchase history, interactions, etc. a cluster is a group of customers who have similar values over these data sources. For an overview on clustering, see [5, 6].

One scalability technique for clustering algorithms is to incrementally summarize dense regions of the data while scanning the dataset. Since a cluster corresponds to a dense region of objects, the records within a dense region can be summarized collectively through a summarized representation, called a cluster feature (CF) (e.g. the triple consisting of the number of points in the cluster, the cluster centroid, and the cluster radius\(^1\)). More sophisticated cluster features are possible.

Cluster features are efficient for two reasons: (1) they occupy less space than maintaining all objects in a cluster, (2) if designed properly, they are sufficient for calculating all inter-cluster and intra-cluster measurements involved in making clustering decisions. Moreover, these calculations can be performed faster than using all objects in clusters. Distances between clusters, radii of clusters, CFs—and hence other properties—of merged clusters can all be computed very quickly from the CFs of individual clusters.

CFs have also been used to scale iterative clustering algorithms, such as the K-Means and EM algorithm [5, 6]. When scaling iterative clustering algorithms, one can identify sets of discardable points, sets of compressible

\(^1\)The radius of a cluster is the square root of the average mean-squared distance of a point in the region.
points, and a sets of main-memory points. A point is discardable if its membership in a cluster can be ascertained already with high confidence; only the CF of all discardable points in a cluster is retained and the actual points are discarded. A point is compressible if it is not discardable but belongs to a very tight sub-cluster consisting of a set of points that always move between clusters simultaneously; such points may move from one cluster to another but they always move together. The remaining records are designated as main-memory records since they are neither discardable nor compressible. The iterative clustering algorithm then moves only the main-memory points and the CFs of compressible points between clusters until a criterion function is optimized.

Other work on scalable clustering focuses on training databases with large attribute sets. In this case, the search methods involve discovering the appropriate sub-space of attributes in which the clusters best exist. These methods aid in understanding the results as the analyst need only focus on the attributes associated with a given cluster.

4 Association Rules

Association rules [5] capture the set of significant correlations present in a dataset. Given a set of transactions, where each transaction is a set of items, an association rule is an implication of the form \( X \implies Y \), where \( X \) and \( Y \) are sets of items. This rule has support \( s \) if \( s\% \) of transactions include all the items in both \( X \) and \( Y \), and confidence \( c \) if \( c\% \) of transactions that contain \( X \) also contain \( Y \). For example, the rule “[Carbonated Beverages] and [Crackers] \implies [Milk]” might hold in a supermarket database with 5% support and 70% confidence. The goal is to discover all association rules that have support and confidence greater than the user-specified minimum support and minimum confidence respectively. This original formulation has been extended in many directions, including the incorporation of taxonomies, quantitative associations, and sequential patterns.

Algorithms for mining association rules usually have two distinct phases. First, they find all sets of items that have minimum support (frequent itemsets). Since the data may consist of millions of transactions, and the algorithm may have to count millions of potentially frequent (candidate) itemsets to identify the frequent itemsets, this phase can be computationally very expensive. Next, rules can be generated directly from the frequent itemsets, without having to go back to the data. The first step usually consumes most of the time, and hence work on scalability has focused on this step.
Scalability techniques can be partitioned into two groups: those that reduce the number of candidates that need to be counted, and those that make the counting of candidates more efficient.

In the first group, identification of the anti-monotonicity property that all subsets of a frequent itemset must also be frequent proved to be a powerful pruning technique that dramatically reduced the number of itemsets that need to be counted. Subsequent work focussed on variations of the original problem. For instance, for datasets and support levels where the frequent itemsets are very long, finding all frequent itemsets is intractable since a frequent itemset with \( n \) items will have \( 2^n \) frequent subsets. However, the set of maximal frequent itemsets can still be found efficiently by looking ahead; once an itemset is identified as frequent, none of its subsets need to be counted (see survey in [1]). The key is to maximize the probability that itemsets counted by looking ahead will actually be frequent. A good heuristic is to bias candidate generation so that the most frequent items appear in the most candidate groups. The intuition is that items with high frequency are more likely to be part of long frequent itemsets.

Another variation is when the user is interested in a fixed consequent, and also specifies that every rule in the result should offer a significant predictive advantage (i.e., higher confidence) over its simplifications. In this case, it is possible to compute bounds on the maximum possible improvement in confidence (as a result of adding items to the antecedent of the rule) and use these bounds to prune the search space.

In the second group of techniques, nested hash tables can be used to efficiently check which candidate itemsets are contained in a transaction. This is very effective when counting shorter candidate itemsets, less so for longer candidates. Techniques for longer itemsets include database projection, where the set of candidate itemsets is partitioned into groups such that the candidates in each group share a set of common items. Then, before counting each candidate group, the algorithm first discards those transactions that do not include all the common items, and for the remaining transactions, discards the common items (since the algorithm knows they are present) and also discards items not present in any of the candidates. This reduction in number and size of the remaining transactions can yield substantial improvements in the speed of counting.
5 From Incremental Model Maintenance to Streaming Data

Real-life data is not static, but is constantly evolving through additions or deletions of records, and in some applications such as network monitoring, data arrives in such high-speed data streams that it is infeasible to store the data for offline analysis. We describe these different models of evolving data through a framework that we call block evolution. In block evolution, the input dataset to the data mining process is not static, but is updated with a new block of tuples at regular time intervals, for example, every day at midnight. A block is a set of tuples that are added simultaneously to the database. For large blocks, this model captures common practice in many of today’s data warehouse installations, where updates from operational databases are batched together and performed in a block update. For small blocks of data, this model captures streaming data, where in the extreme the size of each block is a single record.

For evolving data, two classes of problems are of particular interest: data mining model maintenance and change detection. The goal of change detection is to quantify the difference, in terms of their data characteristics, between two blocks of data. The goal of model maintenance is to maintain a data mining model under insertion and deletions of blocks of data.

There has been much recent work on mining evolving data. Incremental model maintenance has received much attention, since due to the very large size of the data warehouses, it is highly desirable to go from incremental updates of the data warehouse to only incremental updates to existing data mining models. Incremental model maintenance algorithms have concentrated on computing exactly the same model as if the original model construction algorithm was run on the union of old and new data. One scalability technique that is very widespread throughout model maintenance algorithms is localization of changes that insertions of new records can have. For example, for density-based clustering algorithms [6], the insertion of a new record only affects clusters in the neighborhood of the record, and thus efficient algorithms can localize the change to the model without having to recompute the complete model. As another example, in decision tree construction, the split criteria at the tree might change only within certain confidence intervals under insertions of records assuming that the underlying distribution of training records is static.

When working with high-speed data streams, algorithms are required that construct data mining models while looking at the relevant data items
only once and in a fixed order (determined by the stream-arrival pattern), with limited amount of main memory. Data-stream computation has given rise to several recent (theoretical and practical) studies of on-line or one-pass algorithms with limited memory requirements for data mining and related problems. Examples include computation of quantiles and order-statistics, estimation of frequency moments and join sizes, clustering and decision tree construction, estimating correlated aggregates and computing one-dimensional (i.e., single-attribute) histograms and Haar wavelet decompositions.

Scalability techniques used include sampling, usage of summary statistics, sketches (small random projections that have provable performance guarantees in expectation), and online compression of sufficient statistics [2].

6 Conclusion

Large databases are commonplace in today’s enterprises and continually growing. Prior to the invention of scaling techniques, sampling was the primary method used to run conventional machine learning, statistical, and other algorithms on these databases. However, drawbacks of this approach include determining sufficient sample size, and validity of discovered patterns or models (i.e., the patterns may be an artifact of the sample). We view sampling as orthogonal and complementary to scaling techniques, since the latter allow the use of much larger dataset sizes.

The research effort on scaling data mining algorithms to large databases has provided analysts with the ability to model and discover valid, interesting patterns over these large datasets. We have provided an introduction to some strategies that have been successfully used to scale data mining algorithms to large database. Some general scaling principles include use of summary statistics, data compression, pruning the search space, and incremental computation. We have described these principles in the concrete setting of different mining algorithms, and expect that the principles will be useful in other areas of computer science as well.

Scalability in data mining is currently an active area of research, and many challenging problems remain. We conclude by highlighting three key problems. First, can we mine patterns from huge datasets while preserving the privacy of individual records and the anonymity of individuals who have provided the data? Second, what are suitable data mining models for high-speed data streams, and how can we construct such models? Finally, there
is a plethora of huge sets of linked data available - the Internet, newsgroups, news stories etc. What type of knowledge can we mine from these resources and can we design scalable algorithms for this domain?

References


