Vertical Data Mining on Very Large Data Sets

William Perrizo
North Dakota State University, USA

Qiang Ding
Chinatelecom Americas, USA

Qin Ding
East Carolina University, USA

Taufik Abidin
North Dakota State University, USA

INTRODUCTION

Due to the rapid growth of the volume of data that are available, it is of importance and challenge to develop scalable methodologies and frameworks that can be used to perform efficient and effective data mining on large data sets. Vertical data mining strategy aims at addressing the scalability issues by organizing data in vertical layouts and conducting logical operations on vertical partitioned data instead of scanning the entire database horizontally in order to perform various data mining tasks.

BACKGROUND

The traditional horizontal database structure (files of horizontally structured records) and traditional scan-based data processing approaches (scanning files of horizontal records) are known to be inadequate for knowledge discovery in very large data repositories due to the problem of scalability. For this reason much effort has been put on sub-sampling and indexing as ways to address and solve the problem of scalability. However, sub-sampling requires that the sub-sampler know enough about the large dataset in the first place in order to sub-sample "representatively". That is, sub-sampling requires considerable knowledge about the data, which, for many large datasets, may be inadequate or non-existent. Index files are vertical structures. That is, they are vertical access paths to sets of horizontal records. Indexing files of horizontal data records does address the scalability problem in many cases, but it does so at the cost of creating and maintaining the index files separate from the data files themselves.

A new way to organize data is to organize them vertically, instead of horizontally. Data miners are typically interested in collective properties or predictions that can be expressed very briefly (e.g., a yes/no answer). Therefore, the result of a data mining query can be represented by a bitmap vector. This important property makes it possible to perform data mining directly on vertical data structures.

MAIN FOCUS

Vertical data structures, vertical mining approaches and multi-relational vertical mining will be explored in detail to show how vertical data mining works.

Vertical Data Structures

The concept of vertical partitioning has been studied within the context of both centralized and distributed database systems for a long time, yet much remains to be done (Winslett, 2002). There are great advantages of using vertical partitioning, for example, it makes hardware caching work really well; it makes compression easy to do; it may greatly increase the effectiveness of the I/O device since only participating fields are retrieved each time. The vertical decomposition of a relation also permits a number of transactions to be executed concurrently. Copeland & Khoshafian (1985) presented an attribute-level Decomposition Storage Model called DSM, similar to the Attribute Transposed File model (ATF) (Batory, 1979), that stores each column of a relational table into a separate table. DSM was shown to perform well. It utilizes surrogate keys to map individual attributes together, hence requiring a surrogate

key to be associated with each attribute of each record in the database. Attribute-level vertical decomposition is also used in Remotely Sensed Imagery (e.g., Landsat Thematic Mapper Imagery), where it is called Band Sequential (BSQ) format. Beyond attribute-level decomposition, Wong et al. (1985) presented the Bit Transposed File model (BTF), which took advantage of encoded attribute values using a small number of bits to reduce the storage space.

In addition to ATF, BTF, and DSM models, there has been other work on vertical data structuring, such as Bit-Sliced Indexes (BSI) (Chan & Ioannidis, 1998, O'Neil & Quass, 1997, Rinfret et al., 2001), Encoded Bitmap Indexes (EBI) (Wu & Buchmann, 1998; Wu, 1998), and Domain Vector Accelerator (DVA) (Perrizo et al., 1991).

A Bit-Sliced Index (BSI) is an ordered list of bitmaps used to represent values of a column or attribute. These bitmaps are called bit-slices, which provide binary representations of attribute's values for all the rows.

In the EBI approach, an encoding function on the attribute domain is applied and a binary-based bit-sliced index on the encoded domain is built. EBIs minimize the space requirement and show more potential optimization than binary bit-slices.

Both BSIs and EBIs are auxiliary index structures that need to be stored twice for particular data columns. As we know, even the simplest index structure used today incurs substantial increase in total storage requirements. The increased database size, in turn, translates into higher media and maintenance costs, and results in lower performance.

Domain Vector Accelerator (DVA) is a method to perform relational operations based on vertical bit-vectors. The DVA method performs particularly well for joins involving a primary key attribute and an associated foreign key attribute.

Vertical mining requires data to be organized vertically and be processed horizontally through fast, multi-operand logical operations, such as AND, OR, XOR, and complement. Predicate tree (P-tree[1]) (Ding et al., 2002) is one form of lossless vertical structure that meets this requirement. P-tree is suitable to represent numerical and categorical data and has been successfully used in various data mining applications, including classification (Khan et al., 2002; Ding et al., 2002; Perrizo et al., 2007), clustering (Denton et al., 2002), association rule mining (Ding et al., 2002; Imad et al., 2004), and outlier detection (Ren et al., 2004).

To convert a relational table of horizontal records to a set of vertical P-trees, the table has to be projected into columns, one for each attribute, retaining the original record order in each. Then each attribute column is further decomposed into separate bit vectors (called bit Sequential format, or bSQ format), one for each bit position of the values in that attribute. Each bit vector, i.e., bSQ file, is then compressed into a tree structure by recording the truth of the predicate "purely 1-bits" recursively on halves until purity is reached.

Figure 1 is an example of an 8×8 bSQ file and its corresponding P-tree (Ding et al., 2002). In this example, 36 is the number of 1's in the entire image, called root count. This root level is labeled level 0. The numbers 16, 7, 13, and 0 at the next level (level 1) are the 1-bit counts for the four major quadrants in raster order. Since the first and last level-1 quadrants are composed entirely of 1-bits (called pure-1 quadrants) and 0-bits (called pure-0 quadrants) respectively, sub-trees are not needed and these branches terminate. This pattern is continued recursively using the Peano or Z-ordering of the four sub-quadrants at each new level. Eventually, every branch terminates.

A variation of the P-tree data structure, the PM-tree, is a similar structure in which masks rather than counts

Figure 1. P-tree for a 8×8 bSQ file