DEMON: Mining and Monitoring Evolving Data

Venkatesh Ganti*
UW-Madison
vganti@cs.wisc.edu

Johannes Gehrke†
Cornell University
johannes@cs.cornell.edu

Raghu Ramakrishnan‡
UW-Madison
raghu@cs.wisc.edu

Abstract

Data mining algorithms have been the focus of much research recently. In practice, the input data to a data mining process resides in a large data warehouse whose data is kept up-to-date through periodic or occasional addition and deletion of blocks of data. Most data mining algorithms have either assumed that the input data is static, or have been designed for arbitrary insertions and deletions of data records. In this paper, we consider a dynamic environment that evolves through systematic addition or deletion of blocks *of data. We introduce a new dimension called the* data span dimension, *which allows user-defined selections of a temporal subset of the database. Taking this new degree of freedom into account, we describe efficient model maintenance algorithms for frequent itemsets and clusters. We then describe a generic algorithm that takes any traditional incremental model maintenance algorithm and transforms it into an algorithm that allows restrictions on the data span dimension. In a detailed experimental study, we examine the validity and performance of our ideas.*

1. Introduction

Organizations have realized that the large amounts of data they accumulate in their daily business operations can yield useful "business intelligence," or strategic insights, based on observed patterns of activity. There is an increasing focus on *data mining*, which has been defined as the application of data analysis and discovery algorithms to large databases with the goal of discovering (predictive) models [9]. Several algorithms have been proposed for computing novel models, for more efficient model construction, to deal with new data types, and to quantify differences between datasets.

Most data mining algorithms so far have assumed that the input data is static and do not take into account that data evolves over time. Recently, the problem of mining evolving data has received some attention and incremental model maintenance algorithms for several data mining models have been developed [5, 10, 16, 7, 11]. These algorithms are designed to incrementally maintain a data mining model under arbitrary insertions and deletions of records to the database.

But real-life data often does not evolve in an arbitrary way. Consider a data warehouse, a large collection of data from multiple sources consolidated into a common repository, to enable complex data analysis [4]. The data warehouse is updated with new batches of records at regular time intervals, e.g., every day at midnight. Thus data in the data warehouse evolves through addition and deletion of batches of records at a time. We refer to data that changes through addition and deletion of sets of records as *systematically evolving data*. The main difference between arbitrary and systematic evolution is that in the former an individual record can be updated at any time, whereas in the latter sets of records are added together.

In this paper, we assume a dynamic environment of systematically evolving data and introduce the problem of *mining systematically evolving data*. The main contributions of our work are:

1. We present a DEMONic[1] view of the world by exploring the problem space of mining systematically evolving data (Section 2). We introduce a new dimension called the *data span dimension*, which takes the temporal aspect of the data evolution into account and allows an analyst to "mine" relevant subsets of the data.
2. We describe new model maintenance algorithms with respect to the selection constraints on the data span dimension for two popular classes of data mining models: frequent itemsets and clustering (Section 3.1). These algorithms exploit the systematic block evolution to improve the state-of-the-art incremental algorithms. We also introduce a generic algorithm that takes any traditional incremental model maintenance algorithm and derives an incremental algorithm that allows restrictions on the data span dimension (Section 3.2). In particular, the generic algorithm can be instantiated with our incremental algorithms in Section 3.1.

* Supported by a Microsoft Research Fellowship
†Work done when the author was at UW-Madison
‡ This research was supported by Grant 2053 from the IBM corporation.

[1] **D**ata **E**volution and **MON**itoring

3. In an extensive experimental study, we evaluate our algorithms and compare them with previous work wherever possible (Section 4).

2. DEMON

In this section, we introduce the problem of mining systematically evolving data. We describe our model of systematic data evolution in Section 2.1. In Section 2.2, we enumerate the problem space of mining systematically evolving data by introducing the data span dimension, which allows temporal restrictions on the data being mined. Then we refine the type of restrictions by introducing the notion of a block selection sequence in Section 2.3.

2.1. Systematic data evolution

We now describe our model of evolving data. We use the term *tuple* generically to stand for the basic unit of information in the data, e.g., a customer transaction, a database record, or an n-dimensional point. A *block* is a set of tuples. We assume that the database D consists of a (conceptually infinite) sequence of blocks $D_1, \ldots, D_k, \ldots$ where each block D_k is associated with an *identifier* k. We assume without loss of generality that all identifiers are natural numbers and that they increase in the order of their arrival. Unless otherwise mentioned, we use t to denote the identifier of the "latest" block D_t. We call the sequence of all blocks $D_1, \ldots, D_t$ currently in the database the *current database snapshot*.

Note that we do not assume that block evolution follows a regular period; different blocks may span different time units. For example, the first two blocks of data may be added to the database on Saturday and Sunday, respectively, and the third block on the following Friday. The framework can naturally handle this type of irregular block evolution. The lack of constraints on the time spanned by any block also allows us to incorporate hierarchies on the time dimension. (We just merge all blocks that fall under the same parent.)

2.2. Data span dimension

When mining systematically evolving data, some applications are interested in mining all the data accumulated thus far, whereas some other applications are interested in mining only a recently collected portion of the data.

As an example, consider an application that analyzes a large database of documents. Suppose the model extracted from the database through the data mining process is a set of document clusters, each consisting of a set of documents related to a common concept [18]. The document cluster model is used to associate new, unclassified documents with existing concepts. Occasionally, a new block of documents is added to the database, necessitating an update of the document clusters. Typical applications in this domain are interested in clustering the entire collection of documents.

In a different application consider the database of the hypothetical

Note that the block selection predicate is independent of the starting position of the window in the first and second applications whereas in the third application, it is defined relative to the beginning of the window and thus moves with the window. We now formally define the *block selection sequence (BSS)*. Informally, the BSS is a bit sequence of 0's and 1's: a 1 in the position corresponding to a block indicates that the block is selected for mining, and a 0 indicates that the block is left out.

Definition 2.1 *Let $D[1,t] = \{D_1, \ldots, D_t\}$ be the current database snapshot and let $D[t-w+1, t]$ be the most recent window of size w. A* window-independent block selection sequence *is a sequence $\langle b_1, \ldots, b_t, \ldots\rangle$ of 0/1 bits. A* window-relative BSS *is a sequence $\langle b_1, \ldots, b_w\rangle$ of bits ($b_i \in \{0,1\}$), one per block in the most recent window.*

3. Model maintenance algorithms

In this section, we discuss incremental model maintenance algorithms. In Section 3.1, we describe the model maintenance algorithms for frequent itemsets and clustering under the unrestricted window option.[2] In Section 3.2, we describe a generic model maintenance algorithm called GEMM[3] for the most recent window option. The instantiation of GEMM requires a model maintenance algorithm for the unrestricted window option. The instantiated algorithm has identical performance characteristics (time between the arrival of a new block and the availability of the updated model) and main-memory requirements as the algorithm instantiating GEMM at the cost of a small amount of additional disk space and off-line processing. GEMM can be instantiated for any class of data mining models, and with any incremental model maintenance algorithm besides the ones we discuss in Section 3.1. Therefore, GEMM can take full advantage of specialized application-dependent incremental model maintenance algorithms to deliver better performance. Before describing our algorithms, we formally define the problems of frequent itemset computation and clustering.

Set of Frequent Itemsets: Let $\mathcal{I} = \{i_1, \ldots, i_n\}$ be a set of literals called *items*. A *transaction* and an *itemset* are subsets of $\mathcal{I}$. Each transaction is associated with a unique positive integer called the *transaction identifier*. A transaction T is said to *contain* an itemset X if $X \subseteq T$. Let D be a set of transactions. The *support* $\sigma_D(X)$ of an itemset X in D is the fraction of the total number of transactions in D that contain X: $\sigma_D(X) \stackrel{\text{def}}{=} \frac{|\{T : T \in D : X \subseteq T\}|}{|D|}$. Let κ $(0 < \kappa < 1)$ be a constant called the *minimum support*. An itemset X is said to be *frequent* on D if $\sigma_D(X) \geq \kappa$. The set of *frequent itemsets* $L(D,\kappa)$ consists of all itemsets that are frequent on D; formally, $L(D,\kappa) = \{X : X \subset \mathcal{I}, \sigma_D(X) \geq \kappa\}$. The *negative border* $NB^-(D,\kappa)$ of D at minimum support threshold κ is the set of all infrequent itemsets whose proper subsets are all frequent. Formally, $NB^-(D,\kappa) = \{X : X \subset \mathcal{I}, \sigma_D(X) < \kappa \wedge \forall Y \subset X, \sigma_D(Y) \geq \kappa\}$.
The *TID-list* $\theta_D(X)$ of an itemset X is the list of transaction identifiers, sorted in the increasing order, of transactions in D that contain the itemset X. The *size* of $\theta_D(X)$ is the (disk) space occupied by $\theta_D(X)$. We write $\theta(X)$ and $\sigma(X)$ instead of $\theta_D(X)$ and $\sigma_D(X)$ if D is clear from the context.

Clustering: The clustering problem has been widely studied and several definitions for a cluster have been proposed to suit different target applications. In general, the goal of clustering is to find interesting groups (called *clusters*) in the dataset such that points in the same group are more similar to each other than to points in other groups. The quality of a set of clusters is typically captured by a distance-based *criterion function*, e.g., the sum of mean-squared distances of all points from the centers of clusters they belong to. We adopt the following (semi-)formal definition from the Statistics literature for the clustering problem [13]. *Given the required number of clusters K, a dataset of N points, a distance-based measurement function, and a criterion function, partition the dataset into K groups such that the criterion function is optimized.*

3.1. Unrestricted window

We now describe incremental model maintenance algorithms for frequent itemsets and clusters for the unrestricted window option with respect to a user-specified BSS.

3.1.1. Set of frequent itemsets

When a new block D_{t+1} is added to $D[1,t]$ and $b_{t+1} = 1$, the set of frequent itemsets needs to be updated. (If $b_{t+1} = 0$, the current set of frequent itemsets carries over to the new snapshot.) In this section, we discuss two new algorithms, called ECUT[4] and ECUT$^+$, for dynamically maintaining the set of frequent itemsets. These algorithms improve upon the previous best algorithm[5], BORDERS, which was independently developed by Feldman et al. [10] and Thomas et al. [16]. (The improvements exploit the systematic data evolution.) First, we briefly review the BORDERS algorithm before discussing the new algorithms.

The BORDERS algorithm consists of two phases. (1) The *detection phase* recognizes that the set of frequent itemsets has changed. (2) The *update phase* counts a set of new itemsets required for dynamic maintenance. The detection phase relies on the maintenance of the negative border along with the set of frequent itemsets. When a new

[2] In prior work, we developed an algorithm for incremental decision tree construction [11]. Hence we do not address this problem here.

[3] **GE**neric **M**odel **M**aintainer

[4] **E**fficient **C**ounting **U**sing **T**ID-lists

[5] To the best of our knowledge, this is the only algorithm for incrementally maintaining frequent itemsets.