Mining Data Streams under Block Evolution

Venkatesh Ganti
Microsoft Research
vganti@microsoft.com

Johannes Gehrke
Cornell University
johannes@cs.cornell.edu

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

ABSTRACT

In this paper we survey recent work on incremental data mining model maintenance and change detection under *block evolution*. In block evolution, a dataset is updated periodically through insertions and deletions of *blocks* of records at a time. We describe two techniques: (1) We describe a generic algorithm for model maintenance that takes any traditional incremental data mining model maintenance algorithm and transforms it into an algorithm that allows restrictions on a temporal subset of the database. (2) We also describe a generic framework for change detection, that quantifies the difference between two datasets in terms of the data mining models they induce.

1. INTRODUCTION

In this paper we survey two data mining techniques for a special computational model of streaming data, where a dataset is updated periodically through insertions and deletions of sets of tuples at a time; we call this model of changing data *block evolution*. In block evolution, the input dataset to the data mining process is not static, but is updated with a new batch 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. 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 [21]. For example, AT&T updates their call data warehouse once a day with new call data records, and several large retail chains update their data warehouses daily.

Consider two data mining applications that are typical in scenarios of block evolution. Suppose an analyst is modeling the AT&T database for detecting fraudulent call patterns. Processes causing fraud change dynamically because people continuously find newer ways of breaking the system. Therefore, the analyst may only be interested in modeling the most recently collected set of call data records, say, those collected in the past three months. In this case the analyst restricts the scope of the data over which the data mining model should be constructed. As another example, the analyst may be interested in analyzing the impact of a marketing campaign targeted at increasing the sales of long distance service. The goal here is to study the increase in the targeted service and the impact (decrease or increase) on other related services. Therefore, the analyst wants to "compare" the customer behavior in call data records collected before the campaign with the behavior in data collected after the campaign. Going one step further, the analyst wants to know the features in customer data that account for the "largest difference" between the data collected before and after the campaign.

1.1 Data Mining Under Block Evolution

Let us introduce our model of mining data in the block evolution model. In our model, the database consists of a (conceptually infinite) sequence of data blocks $D_1, D_2, \ldots$ that arrive at times $1, 2, \ldots$, where each block D_i consists of a set of records.[1] We call i the *block identifier* of block B_i. Thus at any time t, the database consists of a finite sequence of blocks of data $\langle D_1, \ldots, D_t \rangle$ that have arrived at times $\{1, 2, \ldots, t\}$. One obvious choice for data mining model construction at time t is to construct a series of models over $D[1,1] = D_1, D[1,2] = D_1 \cup D_2, \ldots, D[1,t] = \cup_1^t D_i$ such that at time t we obtain the data mining model $M(D[1,w])$ constructed over $D[1,t]$. In our model of block evolution, a data analyst can specify time-dependent subsets of $D[1,t]$ by defining a *data span* and a *block selection sequence*.

Data Span: In the block evolution model, some applications require mining all data accumulated thus far, whereas other applications are interested in mining only the most recently collected portion of the data.

As an example, consider an application that extracts from a large database of documents a set of document clusters, each consisting of a set of documents related to a common concept [61]. 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 a hypothetical toy store which is updated daily. Suppose the set of frequent itemsets discovered from the database is used by an analyst to devise marketing strategies for new toys. The model obtained from all the data may not interest the analyst for the following reasons. (1) Popularity of most toys is short-lived. Part of the data is "too old" to represent the current customer patterns, and hence the information obtained from this part is stale and does not buy any competitive edge. (2) Mining for patterns over the entire database may dilute some patterns that may be visible if only the most recent window of data, say, the latest 28 days, is analyzed. The marketing analyst may be interested in precisely

[1]General updates typically involve arbitrary insertions and deletions of records.

these patterns to capitalize on the latest customer trends.

To capture these two different requirements, we allow two different *data spans* at time t: In the *unrestricted window (UW)* case, the relevant data consists of $D[1,t]$, all the data collected so far. In the *restricted window* case, a specified number w of the most recently collected blocks of data $D[t-w+1,t] = D_{t-w+1} \cup \cdots \cup D_t$ are selected as input to the data mining activity (we assume without loss of generality $t \geq w$ through the paper). We call the parameter w the *window size.*

Block Selection Sequence: We now introduce an additional selection constraint called the *block selection sequence (BSS)* that can be applied in conjunction with the options on the data span dimension to achieve a fine-grained block selection. The next definition formalizes the notion that a block selection sequence marks relevant blocks in the sequence of database blocks.

Definition 1. A *block selection sequence* is an infinite sequence $\mathcal{B} = \langle b_1, \ldots, b_t, \ldots \rangle$ of bits $b_i \in \{0,1\}$.

We also define two operations on block selection sequences. The first operation is the *projection operator* π which takes as input a block selection sequence $B = \langle b_1, b_2, \ldots \rangle$ and an interval $[t_1, t_2]$ and sets all the bits in $\mathcal{B}$ in the interval to 0, formally $\pi(\mathcal{B}, [t_1, t_2]) = \langle b'_1, b'_2, \ldots \rangle$, where $b'_i = 0$ if $i \notin [t_1, t_2]$ and $b'_i = b_i$, otherwise. We also introduce a *translation operator* τ which takes as input a block selection sequence $B = \langle b_1, b_2, \ldots \rangle$ and an offset t and "shifts" B to the right for t bits, inserting leading 0 bits. Formally $\tau(\mathcal{B}, [t]) = \langle b'_1, b'_2, \ldots \rangle$, where $b'_i = b_{i-t}$ if $i > t$ and $b'_i = 0$, otherwise.

The following hypothetical applications (of interest to a marketing analyst) of a toy store database motivate different types of block selection sequences.

1. The analyst wants to model data collected on all Mondays to analyze sales immediately after the weekend. The required blocks are selected from the unrestricted window by a block selection sequence that marks all blocks added to the database on Mondays.

2. The analyst is interested in modeling data collected on all Mondays in the past 28 days (corresponding to the last 4 weeks). In this case, a block selection sequence that marks all blocks collected on Mondays in the most recent window of size 28 selects the required blocks.

3. The analyst wants to model data collected on the same day of the week as today within the past 28 days. The required blocks are selected from the most recent window of size 28 by a sequence that, starting from the beginning of the window, marks all blocks added every seventh day.

Note that the second application the block selection sequence is independent of the starting position of the window, whereas in the third application it is defined relative to the beginning of the window and thus moves with the window.

1.2 Resulting Problems

Given our model of block evolution with user-specified data span and block selection sequence, we can define two relevant problems: *data mining model maintenance* and *change detection.*

Change detection. The goal of deviation detection is to quantify the difference, in terms of their data characteristics, between two blocks of data D_1 and D_2. Note that one of the difficulties of this informal problem statement is the definition of change: What is a quantitative measure of change between two datasets? We postpone further discussion of change to Section 4, and we continue in this Section by giving a formal defitinion of data mining model maintenance under block evolution.

Data mining model maintenance. The goal of model maintenance is to maintain a data mining model under insertion and deletions of blocks of data according to a specified data span and block selection sequence. Let M be a specific class of data mining models (for example, a decision tree); we denote by $M(D)$ the data mining model of class M induced from dataset D. Let us also define the *selection* of a dataset D given a selection bit b: We define $D^b = \emptyset$ if $b = 0$, and $D^b = D$ if $b = 1$. Similarly, we define the selection of a dataset $D[t_1, t_2] = D_{t_1} \cup D_{t_1+1} \cup \ldots \cup D_{t_2}$ for $1 \leq t_1 < t_2$ as the union of a set of selections on data blocks as follows: We define $D[t_1, t_2]^{\mathcal{B}} = D_{t_1}^{b_{t_1}} \cup D_{t_1+1}^{b_{t_1+1}} \cup \ldots \cup D_{t_2}^{t_2}$ where $\mathcal{B} = \langle b_1, b_2, \ldots \rangle$ for $1 \leq t_1 < t_2$. Informally, $D[t_1, t_2]^{\mathcal{B}}$ consists of the blocks in $D[t_1, t_2]$ selected by $\mathcal{B}$.

Let us now define the problem of data mining model maintenance under block evolution.

Definition 2. Let $D_1, D_2, \ldots$ be a sequence of datasets, let $\mathcal{B} = \langle b_1, b_2, \ldots \rangle$ be a block selection sequence, and let M be a class of data mining models. We can define three variants of mining data under block evolution.

- *Unrestricted window, window-independent block selection sequence.* Given an unrestricted window span, compute the following series of models:

  $M(D[1,1]^{\mathcal{B}}), M(D[1,2]^{\mathcal{B}}), \ldots, M(D[1,t]^{\mathcal{B}}),$
  $M(D[1,t+1]^{\mathcal{B}}), \ldots$

  In this case, the block selection sequence selects individual blocks for inclusion or exclusion from the model being incrementally constructed. At every step, one more bit of the block selection sequence becomes relevant.

- *Restricted window, window-independent block selection sequence.* Given an restricted window span of size w, compute the following series of models:

  $M(D[1,1]^{\mathcal{B}}), M(D[1,2]^{\mathcal{B}}), \ldots, M(D[1,t]^{\mathcal{B}}),$
  $M(D[2,w+1]^{\mathcal{B}}), \ldots, M(D[t-w+1,t]^{\mathcal{B}}), \ldots$

  At every step, one "new" bit of the block selection sequence becomes relevant and one "old" bit of the sequence expires.

- *Restricted window, window-dependent block selection sequence.* Given a restricted window span of size w, compute the following series of models:

  $M(D[1,1]^{\mathcal{B}}), M(D[1,2]^{\mathcal{B}}) \ldots, M(D[1,t]^{\mathcal{B}}),$
  $M(D[2,w+1]^{\tau(\mathcal{B},1)}), M(D[3,w+2]^{\tau(\mathcal{B},2)}), \ldots,$
  $M(D[t-w+1,t]^{\tau(\mathcal{B},t-w+1)}), \ldots$

  In this case, the block selection sequence moves with the window, and at all times only the first w bits of the shifted block selection sequence matter.

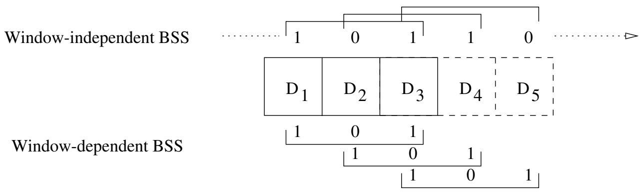

Figure 1: Most Recent Window

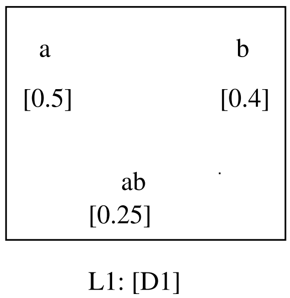

Figure 4: lits-model

Note that the case of unrestricted window, window-independent block selection sequence, with a block selection sequence $\mathcal{B} = \langle 1, 1, \ldots \rangle$ results in the traditional case of incremental model maintenance, and many algorithms have been developed for this case (see the discussion in Section 5). The work on model maintenance that we describe in this article leverages existing algorithms on incremental model maintenance to develop a *generic* model maintenance algorithm for the unrestricted and restricted window case with a user-defined window-independent or window-dependent block selection sequence. The algorithm that we describe in Section 3 *requires* an incremental model maintenance algorithm as input, which is then used as a building block for a generic algorithm for the model maintenance problems described above.

Outline of the article. The remainder of the article is organized as follows. In Section 2, we briefly discuss two classes of data mining models. In Section 3, we describe GEMM, a generic algorithm for model maintenance. In Section 4, we discuss the FOCUS framework for quantitatively measuring changes in data characteristics. We briefly discuss related work in Section 5 and conclude in Section 6.

2. PRELIMINARIES

In this section, we informally introduce two classes of data mining models that we will use to illustrate our techniques in the remainder of the paper: lits-models (short form for frequent itemset models [5]) and dt-models (short form for decision tree models [17]). Our discussion here is intentionally brief as we assume that the reader has familiarity with these models (pointers to detailed introductions can be found in Section 5).

dt-models: Consider a set of records with several attributes. One designated attribute is called the *class label*, the other attributes are called *predictor attributes*. The dependent attribute is a categorical attribute while the predictor attributes can either be categorical or numerical. The goal of predictive modeling is to build a model that takes as input the values of the predictor attributes and predicts a value for the dependent attribute.

A dt-model is a graphical prediction model in the form of a tree. The root of the tree does not have any incoming edges. Every other node has exactly one incoming edge and zero or two or more outgoing edges. If a node n does not have any outgoing edges, we call n a *leaf node*, otherwise we call n an *internal node*. Each edge originating from an internal node is labeled with a *splitting predicate*. The set of splitting predicates P on the outgoing edges of an internal node must be *non-overlapping* and *exhaustive*. A set of predicates P is *non-overlapping* if the conjunction of any two predicates in P evaluates to false. A set of predicates P is *exhaustive* if the disjunction of all predicates in P evaluates to true. Each leaf node is labelled with a class label. An example of a decision tree is shown in Figure 2.

lits-models: The analysis of *market basket* data typically relies on lits-models. A market basket is a collection of items purchased by a customer in an individual transaction, where a *transaction* contains items related to a well-defined business activity, the canonical example being a customer's visit to a grocery store Formally, let $\mathcal{I} = \{i_1, \ldots, i_n\}$ be a set of literals called *items*. A *transaction* and an *itemset* are subsets of $\mathcal{I}$. 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. An itemset whose support is greater than a user-specified minimum support threshold is said to be *frequent*. An example of a lits-models is shown in Figure 4.

3. GENERIC DEMON ALGORITHM

We now describe GEMM, a generic model maintenance algorithm. We only consider the restricted window option; we refer to the full paper for the unrestricted window version [36].

Given a class of models $\mathcal{M}$ and an incremental model maintenance algorithm $A_{\mathcal{M}}$ for the unrestricted window option, GEMM can be instantiated with $A_{\mathcal{M}}$ to obtain a model maintenance algorithm (with respect to both window-independent (Section 3.1) and window-dependent (Section 3.2) block selection sequences $\mathcal{B}$) for the most recent window option.

3.1 Window-independent BSS

Let us explain the central idea of GEMM for a window-independent BSS $\mathcal{B}$. Note that we easily incrementally construct the model $M(D[t-w+1,t])^{\mathcal{B}}$ by starting with an empty model at time $t-w+1$ and adding block $D_{t-w+1}^{b_{t-w+1}}$ using the existing Algorithm $A_{\mathcal{M}}$. We continue to add blocks $D_{t-w+1}^{b_{t-w+1}}, \ldots, D_t^{b_t}$ until we have constructed at time t the model $M(D[t-w+1,t])^{\mathcal{B}}$. Our only requirement was sufficient memory to maintain $M(D[t-w+1,i])^{\mathcal{B}}$ at times $i \in [t-w+1,t]$.

This observation leads to a simple but very general algorithmic idea for incremental block maintenance: We maintain at any time t all necessary partially constructed models for all future windows that overlap with the current time t.

Let us give an inductive description of the GEMM algorithm. Assume at step t we constructed model $M(D[t-w+1,t]^{\mathcal{B}})$. There are $w-1$ future windows that overlap with time t: $[t-w+2,t+1], \ldots, [t,t+w-1]$. Inductively, we assume that in addition to $M(D[t-w+1,t]^{\mathcal{B}})$ we also maintained

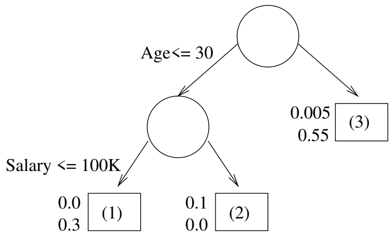

Figure 2: dt-model

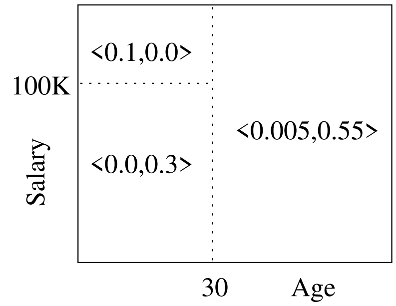

Figure 3: DT:two-components

models $M(D[t'-w+1,t]^{\mathcal{B}})$ for $t' \in \{t+1, t+w-2\}$. Then when the block $t+1$ arrives, we use $A_{\mathcal{M}}$ to update models $M(D[t'-w+1,t]^{\mathcal{B}})$ to $M(D[t'-w+1,t+1]^{\mathcal{B}})$ using $A_{\mathcal{M}}$ and $D_{t+1}^{b_{t+1}}$ for $t' \in \{t+1, t+w-2\}$. We then add model $M(D[t+1,t+1]^{B})$, completing the induction step.

As an example, consider the current database snapshot $D[1,3]$ with $w=3$ in Figure 1. The window-dependent BSS is 101 (shown above the sequence of database blocks), and the window-independent BSS is 10110 ... (shown above the sequence of database blocks). The future windows that overlap with $[1,3]$ are $[2,4]$ and $[3,5]$. At time $t=3$, the auxiliary models that are maintained in addition to the current model $M(D[1,3]^{\mathcal{B}})$ are $M(D[2,3])^{\mathcal{B}}$ and $M(D[3,3]^{\mathcal{B}})$ —the prefixes of $[2,4]$ and $[3,5]$ that overlap $[1,3]$.

When the new block D_4 is added, the collection of auxiliary models is updated as follows:

$M(D[2,4], \langle 1011\ldots\rangle) = A_{\mathcal{M}}(D_4, M(D[1,3], \langle 101\ldots\rangle))$,
$M(D[3,4], \langle 1011\ldots\rangle) = A_{\mathcal{M}}(D_4, M(D[3,3], \langle 101\ldots\rangle))$,
$M(D[4,4], \langle 1011\ldots\rangle) = A_{\mathcal{M}}(D_4, \emptyset)$

3.2 Window-dependent BSS

Consider the database snapshot $D[1,3]$ shown in Figure 1 with the window-dependent $\mathcal{B} = \langle 101\ldots\rangle$, $w=3$, at time $t=3$. The current model on $D[1,3]$ is extracted from the blocks D_1 and D_3. When D_4 is added, the window shifts to the right and the new model over the interval $[2,4]$ is constructed from blocks D_2 and D_4. Observe that the new model can be obtained by updating (using $A_{\mathcal{M}}$) the model extracted from the block D_2, if we would have maintained $M(D_2)$ at times $t=2$ and $t=3$. Note that the relevant set of blocks (for extracting the model from the overlap between $D[1,3]$ and $D[2,4]$) is selected from $D[1,3]$ by the BSS $\langle 0101\ldots\rangle$—the translation of the original BSS $\langle 101\rangle$ with an offset of 1. This example shows us that we can use the same idea as in Section 3.1: We maintain an auxiliary collection of partial models for all future windows that overlap with the current time t. Thus the procedure for maintaining and updating a collection of models for a window-dependent BSS is analogous to the procedure for window-independent BSS.

Let us again give an inductive description of the GEMM algorithm for a window-dependent BSS $\mathcal{B} = \langle b_1, b_2, \ldots\rangle$. Assume we are at step t with model $M(D[t-w+1,t]^{\tau(\mathcal{B},t-w+1)})$. There are $w-1$ future windows that overlap with time t: $[t-w+2,t+1], \ldots, [t,t+w-1]$. Inductively, we assume that in addition to $M(D[t-w+1,t]^{\tau(\mathcal{B},t-w+1)})$ we also maintained models $M(D[t'-w+1,t]^{\tau(\mathcal{B},t'-w+1)})$ for $t' \in \{t+1, t+w-2\}$. Then when the block $t+1$ arrives, we use $A_{\mathcal{M}}$ to update models $M(D[t'-w+1,t]^{\tau(\mathcal{B},t'-w+1)})$ to $M(D[t'-w+1,t+1]^{\tau(\mathcal{B},t'-w+1)})$ using $A_{\mathcal{M}}$ and $D_{t+1}^{b_{t+1}}$ for $t' \in \{t+1, t+w-2\}$. We then add model $M(D[t+1,t+1]^{\tau(\mathcal{B},t+1)})$, completing the induction step.

For the example in Figure 1, the collection of auxiliary models maintained at time $t=3$ is
$M(D[1,3]^{\langle 101\ldots\rangle}), M(D[1,3]^{\langle 0101\ldots\rangle}), M(D[1,3]^{\langle 00101\ldots\rangle})$.

When the new block D_4 is added, the collection of models is updated to:

$M(D[2,4], \langle 01011\ldots\rangle) = A_{\mathcal{M}}(D_4, M(D[1,3], \langle 0101\ldots\rangle))$,
$M(D[3,4], \langle 0010\ldots\rangle) = A_{\mathcal{M}}(D_4, M(D[3,3], \langle 00101\ldots\rangle))$,
$M(D[4,4], \langle 0000101\ldots\rangle) = A_{\mathcal{M}}(D_4, \emptyset)$

3.3 Time and Space Requirements

In this section, we denote the model on the window $D[1,w]$ with respect to a (window-independent or window-dependent) BSS $\mathcal{B}$ by $M(D[1,w], \mathcal{B})$. We define the *response time* to be the time elapsed between the addition of a new block D_{w+1} and the availability of the updated model $M(D[2,w+1], \mathcal{B})$. We observe that for either type of BSS, the computation of the new model $M(D[2,w+1], \mathcal{B})$ involves at most a single invocation of $\mathcal{A}_{\mathcal{M}}$ with the two arguments: D_{w+1} and $M(D[2,w], \mathcal{B}')$ (where $\mathcal{B}'$ is a potential translation of $\mathcal{B}$ as shown in Section 3.2). Therefore, the response time is less than or equal to the time taken by $A_{\mathcal{M}}$ to update the model $M(D[2,w], \mathcal{B}')$ with D_{w+1}.

Except for the model $M(D[2,t+1], b)$, the auxiliary models for future windows are not required immediately at time $t+1$. Therefore, these updates are not time-critical and can be performed off-line when the system is idle. However, some of these models need to be updated before the subsequent block arrives. An important implication of the lack of immediacy of these updates is that the collection of auxiliary models (except $M(D[2,t], \mathcal{B}')$ can be stored on disk and retrieved when necessary. Thus main memory is not a limitation as long as a single model fits in-memory. Like all current data mining algorithms, we assume that at least one model fits into main memory, and we maintain $w-1$ additional models on disk. Since the space occupied by a data mining model is likely to be insignificant when compared to the space requirements of each block, the additional disk space required for these models is negligible.

3.3.1 Options and Optimizations

Certain classes of models are also maintainable under deletion of records. For example, frequent itemsets can be maintained under deletions of transactions [24; 25; 26]. In this case, the algorithm proceeds exactly as for the addition of transactions except that the support of all itemsets contained in a deleted transaction is decremented. Maintainability under deletions gives two choices for model maintenance under the most recent window option: (1) GEMM instantiated with the model maintenance algorithm $A_{\mathcal{M}}$ for the addition of new blocks, and (2) $A^u_{\mathcal{M}}$ that directly updates the model to reflect the addition of the new block and the deletion of the oldest block in the current window. We first discuss the space-time trade-offs between the two choices for the special case when the BSS has the special form $\mathcal{B} = \langle 111 \ldots 1 \rangle$, and then for an arbitrary BSS.

Let the BSS be $\mathcal{B} = \langle 111 \ldots \rangle$. The first option (usage of GEMM) requires disk space to maintain $w - 1$ models with response time being the invokation of $A_{\mathcal{M}}$ to add the new block. In the second option (usage of $A^u_{\mathcal{M}}$), we only maintain one model. However, $A^u_{\mathcal{M}}$ has to add a new block to the window and delete the oldest block in the window, and hence this option approximately takes twice as long as GEMM(assuming that deletion of a record takes as much time as addition and the blocks being deleted and added are of the same size). Also note that usage of $A^u_{\mathcal{M}}$ requires storage of the last w data blocks, since we need D_{t-w+1} to delete it from the current window. Therefore, GEMM has better response time characteristics with smaller space requirements.

The full generality of GEMM comes to the fore for classes of models that cannot be maintained under deletions of records, and in cases where model maintenance under deletion of records is more expensive than that under insertion. For instance, the cost incurred by incremental DBScan to maintain a set of clusters when a record is deleted is higher than that when a record is inserted [31].

When we consider a window-relative BSS, a major drawback of using $A^u_{\mathcal{M}}$ for model maintenance is that $A^u_{\mathcal{M}}$ may require deletion and addition of many blocks to update the model. Recall that a (window-dependent) BSS chooses a subset B of the set of blocks $\{D_1, \ldots, D_w\}$ in the window. When the window shifts right, depending on the BSS, a number (≥ 1) of blocks may be newly added to B and more than one block may be deleted from B. Therefore, $A^u_{\mathcal{M}}$ scans all blocks in the newly added set as well as the deleted set. For certain block selection sequences, it may reduce to the naive reconstruction of the model from scratch, for example for the BSS $\mathcal{B} = \langle 10101010 \ldots \rangle$. At any time t, the model constructed at time $t-1$ contains exactly the "wrong" set of blocks.

4. CHANGE DETECTION: FOCUS

We now discuss the FOCUS framework for change detection. In general, a data mining model constructed from a dataset is designed to capture the interesting characteristics in the data. Therefore, FOCUS uses the difference between data mining models as the measure of change, called *deviation*, between the underlying datasets. In this section, we illustrate the concepts and ideas behind the framework's computation of deviation between two datasets first through the class of decision tree models and then through the class of frequent itemsets. The details can be found in [37].

4.1 dt-models

Let the decision tree constructed from a hypothetical dataset D with two classes—C_1 and C_2—be as shown in Figure 2. The decision tree consists of three leaf nodes. The class distribution at each leaf node is shown beside it (on the left side) with the top (bottom) number denoting the fraction of database records that belong to class C_1 (C_2, respectively). For instance, the fractions of database records that belong to the classes C_1 and C_2 in the leaf node (1) are 0.0 and 0.3, respectively. Each leaf node in the decision tree corresponds to two regions (one region for class C_1 and one region for class C_2), and each region is associated with the fraction of records in the dataset that map into it; this fraction is called the *measure* of the region. Generalizing from this example, each leaf node of a decision tree for k classes is associated with k regions in the attribute space each of which is associated with its measure. These k regions differ only in the class label attribute. In fact, the set of regions associated with all the leaf nodes partitions the attribute space.

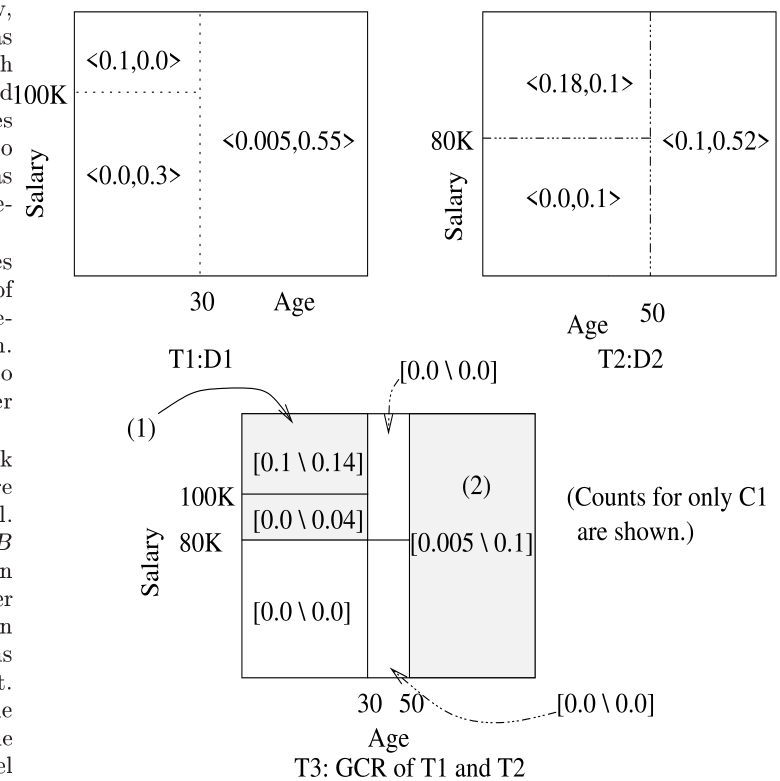

Figure 5: dt-model:$T_3 = \bigwedge(T_1, T_2)$

The set of regions associated with all the leaf nodes in the dt-model is called the *structural component* of the model, and the set of measures associated with each region in the structural component the *measure component* of the model. The property that a model consists of structural and measure components is called the *two-component* property. Figure 3 shows the set of regions in the structural component of the decision tree in Figure 2 where the two regions corresponding to a leaf node are collapsed together for clarity in presentation. The two measures of a leaf node are shown as an ordered pair, e.g., the ordered pair $\langle 0.0, 0.3 \rangle$ consists of the measures for the two collapsed regions of the leaf node (1) in Figure 2.

We now illustrate the idea behind the computation of deviation between two datasets over a set of regions. Let D_1 and D_2 be two datasets. Given a region and the measures of that region from the two datasets, the *deviation* between D_1 and D_2 with respect to the region is a function (e.g., absolute difference) of the two measures; call this function the *difference function*. A generalization to the deviation over a set of regions is a "combination" of all their deviations at each region; represent this combination of deviations by a function called the *aggregate function*, e.g., sum.

If two datasets D_1 and D_2 induce decision tree models with identical structural components, we can combine the two ideas—the two-component property and the deviation with respect to a set of regions—to compute their deviation as follows: the deviation between D_1 and D_2 is the deviation between them with respect to the set of regions in their (identical) structural components.

However, the decision tree models induced by two distinct datasets typically have different structural components, and hence the simple strategy described above for computing deviations may not apply. Therefore, we first make their structural components identical by "extending" them. The extension operation relies on the structural relationships between models, and involves refining the two structural components by splitting regions until the two sets become identical. Intuitively, the refined set of regions is the finer partition obtained by overlaying the two partitions of the attribute space induced by the structural components of both decision trees. We call the refined set of regions the *greatest common refinement* (GCR) of the two structural components. For instance, in Figure 5, T_3 is the GCR of the two trees T_1 (induced by D_1) and T_2 (induced by D_2). In each region of the GCR T_3, we show a hypothetical set of measures (only for class C_1) from the datasets D_1 and D_2. For instance, the measures for the region $\texttt{salary} \geq 100K$ and $\texttt{age} < 30$ for the class C_1 from D_1 and D_2 are 0.0 and 0.04, respectively. The property that the GCR of two models always exists is called the *meet-semilattice* property of the class of models.

To summarize, the deviation between two datasets D_1 and D_2 is computed as follows. The structural components of the two dt-models are extended to their GCR. Then, the deviation between D_1 and D_2 is the deviation between them over the set of all regions in the GCR. In Figure 5, if the difference function is the absolute difference and the aggregate function is the sum then the deviation between D_1 and D_2 over the set of all C_1 regions is given by the sum of deviations at each region in T_3: $|0.0 - 0.0| + |0.0 - 0.04| + |0.1 - 0.14| + |0.0 - 0.0| + |0.0 - 0.0| + |0.05 - 0.1| = 0.13$.

4.2 lits-models

Paralleling the example computation using the class of decision tree models, we now illustrate the deviation computation through the class of frequent itemset models.

Figure 4 shows a simple itemset model where $\mathcal{I} = \{a, b\}$. It has three interesting regions identified by the frequent itemsets $\{a\}$, $\{b\}$, and $\{a, b\}$. Each itemset (equivalently, the corresponding region) is associated with its support: $\{a\}$ with 0.5, $\{b\}$ with 0.4, and $\{a, b\}$ with 0.25. The measure of a region identified by an itemset is the support of the itemset. Generalizing from this example, each frequent itemset X in a lits-model represents a region in the attribute space (where

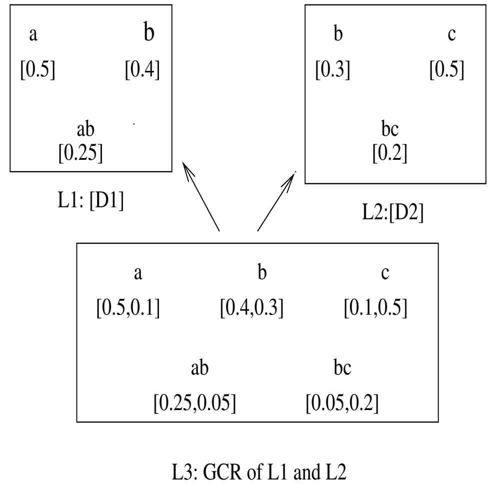

Figure 6: lits-model: $L_3 = \bigwedge(L_1, L_2)$

the support is higher than the threshold) whose measure is the support of X. The set of all frequent itemsets is the *structural component* and the set of their supports is the *measure component*.

As in the case of decision trees, if the structural components of two models are identical we compute the deviation between them to be the aggregate of the deviations between the measures at all regions in either structural component. However, if the structural components are different, we first make them identical by extending both models to their *greatest common refinement*. For the lits-models, the GCR is the union of the sets of frequent itemsets of both models. For example, Figure 6 shows the GCR of two lits-models L_1 induced by D_1 and L_2 induced by D_2. The measures (or supports) obtained by scanning D_1 and D_2 for each itemset in the GCR are shown below it. The deviation between the datasets is the deviation between them over the set of all regions in the GCR. If the difference function is the absolute difference, and the aggregate function is the sum then the deviation between D_1 and D_2 is $|0.5-0.1|+|0.4-0.3|+|0.1-0.5|+|0.25-0.05|+|0.05-0.2| = 1.125$.

Deviation between two datasets

The FOCUS framework essentially generalizes the above two examples showing that popular classes of data mining models (dt-models, lits-models, and cluster-models) always consist of two components: the *structural component* identifying a set of interesting regions of the attribute space, and the *measure component* summarizing, with respect to a dataset, each region with one or several measures. Given the structural components of two different data mining models $M(D_1)$ and $M(D_2)$, it is always possible to find a unifying structural component, called the *greatest common refinement (GCR)*, and to determine the measure components corresponding to the GCR with respect to D_1 and D_2. Let f be a difference function quantifying the deviation between the measures with respect to D_1 and D_2 at a specific region in the GCR, and g be an aggregate function over a combining a set of deviations. The *deviation* between D_1 and D_2 through the data mining models $M(D_1)$ and $M(D_2)$ they induce is the aggregate, computed using g, of the set of deviations computed using f between measures with respect to D_1 and D_2 of each region in the GCR.

4.3 Additional Comments

Focused Deviations: In the above examples, we computed the deviation between two datasets over the entire attribute space. In cases where an analyst is interactively exploring two datasets to find regions where they differ considerably, it is necessary to "focus" the deviation computation with respect to a specific region R. The FOCUS framework covers such requirements. The computation is focused with respect to region R by first intersecting each region in the GCR with R and then combining (using the aggregate function) the deviations over these intersected regions. The intersection with R ensures that the deviation is computed only over regions contained in R. In Figure 5, suppose the analyst is interested only in the difference between T_1 and T_2 over the region R: $\texttt{age} < 30$. The regions in the GCR T_3 intersected with R are the three leftmost regions that satisfy the condition $\texttt{age} < 30$. The deviation between T_1 and T_2 with respect to R is: $|0.0 - 0.0| + |0.0 - 0.04| + |0.1 - 0.14| = 0.08$.
A complementary approach is to declaratively specify a set of "interesting" regions in terms of the structural components of the two models and then rank the interesting regions in the order of their deviations. Towards this goal, FOCUS introduces a set of structural operators and a ranking operator for declarative specification of interesting regions and region-ranking, respectively. Details can be found in the paper [37].

Statistical Significance of Deviation: Suppose the deviation between D_1 and D_2 is 0.005, and that between D_1 and D_3 is 0.01. From just the deviation values, we are able to say the data characteristics of D_1 and D_2 are more similar than those of D_1 and D_3. But, we still do not know whether they have "different" data characteristics; a deviation of 0.01 may not be uncommon between two datasets generated by the same process. In other words, is the deviation value statistically "significant"? FOCUS uses bootstrapping techniques for computing the distribution of deviation values when data characteristics remain the same, and then uses this distribution to answer whether the observed deviation value indicates a significant deviation.

From the FOCUS framework, the misclassification error metric (from Machine Learning and Statistics) and the chi-squared goodness of fit statistic (from Statistics) can be instantiated. Both metrics have traditionally been considered only in the context of dt-models. Thus, the FOCUS framework which covers other classes of models as well is more general than current approaches in Machine Learning and Statistics.

5. RELATED WORK

We first discuss incremental mining algorithms for frequent itemsets, clustering, and classification. We then review work related to the change detection problem spanning Statistics and data mining.

5.1 Model Maintenance

The FUP algorithm and its derivatives are the first to address the problem of incrementally maintaining frequent itemsets [24; 25; 26]. It makes several iterations and in each iteration, it scans the entire database (including the new block and the old dataset). The BORDERS algorithm improves the FUP algorithm by reducing the number of scans of the old database. Ester et al. [31] extended DBScan [32] to develop a scalable incremental clustering algorithm. In prior work, we developed a scalable incremental algorithm for maintaining decision tree classifiers [38]. Utgoff et al. [59] developed ID5, an incremental version of ID3, which assumes that the entire dataset fits in main memory and hence is not scalable.

5.2 Data Streams

Data structures that hold summary information over data streams are examples of *synopsis data structures* [16; 40; 4; 3; 10; 2]. Construction of summary data structures over data streams has been of much interest recently [47]. Algorithms and systems have been proposed for the computation of approximate frequency moments [9], L^1 and L^p distance functions [33; 35], property testing [34], and signatures [27]. There has also been recent work on mining data streams, such as the construction of decision trees over data streams [38; 30; 48] and clustering data streams [62; 43; 54]. Recent work by Agrawal et al. [7], Gibbons et al. [39], Alsabti et al. [11] , Manku et al. [52; 53], and Greenwald and Khanna [42] consider how to compute the approximate median and other quantiles in a single pass over a finite data set. Correlated aggregates are considered in by Chatziantoniou and Ross [20], Chatziantoniou [19], Akinde et al [8], and Gehrke et al. [23] with the focus on exact and approximate computation of correlated aggregates over finite data sets in multiple passes.

The maintenance of aggregate queries is a special case of the the problem of incremental view maintenance; in particular, the maintenance of basic statistical aggregates in the presence of database updates was considered in [55]. The synopsis data structures of Matias et al [41] consider the approximate maintenance of more fancy aggregates in the presence of updates. In online aggregation, Hellerstein et al. study the convergence of basic aggregates over finite data sets [46] and they describe access methods that retrieve records in random order in order to use statistical estimators based on independence assumptions. This work has been extended to online computation of joins [45], online reordering [56] and to adaptive query processing [15].

Related is also work on the processing on continuous queries [58], such as in the OpenCQ System [51], the NiagaraCQ Project [22], and the XFilter System [12]. Data streams also have an interesting relationship to the problem of concept drift in the machine learning community [49; 50; 60].

5.3 Change Detection

The general approach in Statistics for detecting significant differences relies on the goodness of fit tests (e.g., [28]) which try to measure how well a dataset fits a model or a hypothesis. The chi-squared test is a commonly used goodness of fit test. Similar approaches are employed in the context of time series analysis (e.g., [13]). The FOCUS framework generalizes these approaches in two ways. First, we consider popular classes of data mining models instead of traditional statistical models. Second, our bootstrapping-based procedure for computing the significance of deviation is a generalization of traditional methods which assume that the test statistic follows a known distribution (say, chi-squared or normal).

Interestingness measures to monitor variation in a single pattern were proposed in [57]. A similar problem of monitoring the support of an individual itemset was addressed in [6; 18]. Given a pattern (or itemset) their algorithms propose to track its variation over a temporally ordered set of transactions. However, they do not detect variations at levels higher than that of a single pattern.

6. CONCLUSIONS

We explored the problem space of systematic data evolution, typical of most data warehouses, for two important data mining goals: model maintenance and change detection. We then reviewed algorithms for both objectives.

Acknowledgements. We acknowledge gifts from Microsoft and Intel; support from the NSF under grants IIS-0121175 and IIS-0084762; and an IBM Faculty Development Award. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF or Microsoft, Intel, or IBM.

7. REFERENCES

[1] A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K.-Y. Whang, editors. *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt.* Morgan Kaufmann, 2000.

[2] S. Acharya, P. B. Gibbons, and V. Poosala. Aqua: A fast decision support systems using approximate query answers. In Atkinson et al. [14], pages 754–757.

[3] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In Delis et al. [29], pages 574–576.

[4] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In Delis et al. [29], pages 275–286.

[5] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast Discovery of Association Rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 12, pages 307–328. AAAI/MIT Press, 1996.

[6] R. Agrawal and G. Psaila. Acive data mining. *Proceedings of the first international conference on knowledge discovery and data mining*, 1995.

[7] R. Agrawal and A. Swami. A one-pass space-efficient algorithm for finding quantiles. In S. Chaudhuri, A. Deshpande, and R. Krishnamurthy, editors, *Proceedings of the 7th International Conference on Management of Data (COMAD)*, December 1995.

[8] M. Akinde, D. Chatziantoniou, T. Johnson, and S. Kim. The MD-join: An operator for complex OLAP. In *Proceedings of the IEEE International Conference on Data Engineering*, 2001.

[9] Alon, Matias, and Szegedy. The space complexity of approximating the frequency moments. *JCSS: Journal of Computer and System Sciences*, 58, 1999.

[10] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania*, pages 10–20. ACM Press, 1999.

[11] K. Alsabti, S. Ranka, and V. Singh. A one-pass algorithm for accurately estimating quantiles for disk-resident data. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 346–355. Morgan Kaufmann, 1997.

[12] M. Altinel and M. J. Franklin. Efficient filtering of xml documents for selective dissemination of information. In Abbadi et al. [1], pages 53–64.

[13] T. W. Anderson. *The statistical analysis of time series.* John Wiley & Sons, Inc., 1971.

[14] M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, editors. *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK.* Morgan Kaufmann, 1999.

[15] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In W. Chen, J. F. Naughton, and P. A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pages 261–272. ACM, 2000.

[16] D. Barbará, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, T. Johnson, R. T. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The new jersey data reduction report. *Data Engineering Bulletin*, 20(4):3–45, 1997.

[17] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, Belmont, 1984.

[18] S. Chakrabarti, S. Sarawagi, and B. Dom. Mining surprising patterns using temporal description length. In *Proceedings of the 24th International Conference on Very Large Databases*, pages 606–617, August 1998.

[19] D. Chatziantoniou. Ad hoc OLAP: Expression and evaluation. In *Proceedings of the IEEE International Conference on Data Engineering*, 1999.

[20] D. Chatziantoniou and K. A. Ross. Querying multiple features of groups in relational databases. In *Proceedings of the International Conference on Very Large Databases*, pages 295–306, 1996.

[21] S. Chaudhuri, U. Dayal, and V. Ganti. Database technology for decision support systems. In *IEEE Computer*, volume 34, pages 48–55, December 2001.

[22] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. Niagracq: A scalable continuous query system for internet

databases. In W. Chen, J. F. Naughton, and P. A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, volume 29, pages 379–390. ACM, 2000.

[23] Z. Chen, J. Gehrke, and F. Korn. Query optimization in compressed database systems. In *SIGMOD 2001, Proceedings ACM SIGMOD International Conference on Management of Data, 2001, Santa Barbara, California, USA*. ACM Press, 2001.

[24] D. Cheung, J. Han, V. Ng, and C. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proceedings of the twelfth international conference on data engineering (ICDE)*, February 1996.

[25] D. Cheung, S. Lee, and B. Kao. A general incremental technique for maintaining discovered association rules. In *Proceedings of the fifth DASFAA Conference*, April 1997.

[26] D. Cheung, T. Vincent, and W. Benjamin. Maintenance of discovered knowledge: A case in multi-level association rules. In *Proceedings of the second international conference on knowledge discovery in databases*, August 1996.

[27] C. Cortes, K. Fisher, D. Pregibon, and A. Rogers. Hancock: a language for extracting signatures from data streams. pages 9–17. ACM Press, 2000.

[28] R. B. D'Agostino and M. A. Stephens. *Goodness-of-fit techniques.* New York: M.Dekker, 1986.

[29] A. Delis, C. Faloutsos, and S. Ghandeharizadeh, editors. *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadephia, Pennsylvania, USA*. ACM Press, 1999.

[30] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining*, pages 71–80, Boston, MA, August 2000. ACM.

[31] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment. In *Proceedings of the 24th International Conference on Very Large Databases*, pages 323–333, August 1998.

[32] M. Ester, H.-P. Kriegel, and X. Xu. A database interface for clustering in large spatial databases. In *Proc. of the 1st Int'l Conference on Knowledge Discovery in Databases and Data Mining*, Montreal, Canada, August 1995.

[33] Feigenbaum, Kannan, Strauss, and Viswanathan. An approximate L1-difference algorithm for massive data streams (extended abstract). In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1999.

[34] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. Testing and spot-checking of data streams. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms*, 2000.

[35] Fong and Strauss. An approximate L^p-difference algorithm for massive data streams. In *STACS: Annual Symposium on Theoretical Aspects of Computer Science*, 2000.

[36] V. Ganti, J. Gehrke, and R. Ramakrishnan. Demon: Mining and monitoring evolving data. *IEEE Transactions on knowledge and data engineering*, 13(1):50–63, January-Febraury 2001.

[37] V. Ganti, J. Gehrke, R. Ramakrishnan, and W.-Y. Loh. A framework for measuring changes in data characteristics. In *Proceedings of the 18th Symposium on Principles of Database Systems*, 1999.

[38] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.-Y. Loh. Boat-optimistic decision tree construction. In Delis et al. [29], pages 169–180.

[39] P. Gibbons, Y. Mattias, and V. Poosala. Fast Incremental Maintenance of Approximate Histograms. *Proceedings of VLDB, Athens Greece*, pages 466–475, Aug. 1997.

[40] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In Haas and Tiwary [44], pages 331–342.

[41] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 909–910, N.Y., Jan. 17–19 1999. ACM-SIAM.

[42] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD 2001, Proceedings ACM SIGMOD International Conference on Management of Data, 2001, Santa Barbara, California, USA*. ACM Press, 2001.

[43] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *In Proceedings of the Annual Symposium on Foundations of Computer Science*. IEEE, November 2000.

[44] L. M. Haas and A. Tiwary, editors. *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*. ACM Press, 1998.

[45] P. J. Haas and J. M. Hellerstein. Ripple joins for online aggregation. In Delis et al. [29], pages 287–298.

[46] J. M. Hellerstein, P. J. Haas, and H. Wang. Online aggregation. In J. Peckham, editor, *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 171–182. ACM Press, 1997.

[47] M. R. Henzinger, P. .Raghavan, and S. Rajagopalan. Computing on data streams. *Technical Report 1998-011, Digital Eqipment Corporation, Systems Research Center*, May, 1998.

[48] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining*, pages 97–106, San Francisco, CA, 2001.

[49] F. Kilander and C. G. Jansson. COBBIT - A control procedure for COBWEB in the presence of concept drift. In P. B. Brazdil, editor, *Proceedings of the European Conference on Machine Learning (ECML-93)*, volume 667 of *LNAI*, pages 244–261, Vienna, Austria, Apr. 1993. Springer Verlag.

[50] M. Klenner and U. Hahn. Concept versioning: A methodology for tracking evolutionary concept drift in dynamic concept systems. In A. G. Cohn, editor, *Proceedings of the Eleventh European Conference on Artificial Intelligence*, pages 473–477, Chichester, Aug. 8–12 1994. John Wiley and Sons.

[51] L. Liu, C. Pu, W. Tang, D. Buttler, J. Biggs, T. Zhou, P. Benninghoff, W. Han, and F. Yu. Cq: A personalized update monitoring toolkit. In Haas and Tiwary [44], pages 547–549.

[52] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In Haas and Tiwary [44], pages 426–435.

[53] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In Delis et al. [29], pages 251–262.

[54] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. High-performance clustering of streams and large data sets. In *Proceedings of the 18th International Conference on Data Engineering*, 2002.

[55] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In Abbadi et al. [1], pages 144–155.

[56] V. Raman, B. Raman, and J. M. Hellerstein. Online dynamic reordering for interactive data processing. In Atkinson et al. [14], pages 709–720.

[57] A. Silbershatz and A. Tuzhilin. What makes patterns interesting in knowledge discovery systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(6), 1996.

[58] D. B. Terry, D. Goldberg, D. Nichols, and B. M. Oki. Continuous queries over append-only databases. In M. Stonebraker, editor, *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, San Diego, California, June 2-5, 1992*, pages 321–330. ACM Press, 1992.

[59] P. Utgoff. ID5: An incremental ID3. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 107–120. Morgan Kaufmann, 1988.

[60] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

[61] P. Willett. Recent trends in hierarchical document clustering: A critical review. *Information Processing and management*, 24(5):577–597, 1988.

[62] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2), 1997.