2009-03-01

A Dynamic Attribute-Based Data Filtering and Recovery Scheme for Web Information Processing

Amit Ahuja

Yiu-Kai D. Ng
ng@cs.byu.edu

A Dynamic Attribute-Based Data Filtering and Recovery Scheme for Web Information Processing

Amit Ahuja
Yiu-Kai Ng
Computer Science Department
Brigham Young University
Provo, Utah 84602
ng@cs.byu.edu, amit_ahuja83@yahoo.com

Abstract

Web data being transmitted over a network channel on the Internet with excessive amount of data causes data processing problems, which include selectively choosing useful information to be retained for various data applications. In this paper, we present an approach for filtering less-informative attribute data from a source Website. A scheme for filtering attributes, instead of tuples (records), from a Website becomes imperative, since filtering a complete tuple would lead to filtering some *informative*, as well as less-informative, attribute data in the tuple. Since filtered data at the source Website may be of interest to the user at the destination Website, we design a data recovery approach that maintains the minimal amount of information for data recovery purpose while imposing minimal overhead for data recovery at the source Website. Our data filtering and recovery approach (i) handles a wide range of Web data in different application domains (such as weather, stock exchanges, Internet traffic, etc.), (ii) is dynamic in nature, since each filtering scheme adjusts the amount of data to be filtered as needed, and (iii) is adaptive, which is appealing in an ever-changing Internet environment.

1 Introduction

In recent years we have seen tremendous change in the way data are transferred over the Internet. As the huge amount of Web data flows through the Internet, it may be difficult to (i) process all the incoming data for an application program, (ii) compute sophisticated functions on large pieces of inputs at the rate it is presented, and (iii) store temporarily or archive data from multiple Websites. To handle these problems, there arises a need to filter out some of these data so that the data can be processed, or stored with ease. This is the need that motivates our data filtering work.

We notice that some of the Web data sources with structured data are in the tabular format, i.e., with tuples and columns, in which columns are attributes. This approach yields a static schema. In these structured data, the involved attributes and their corresponding data types do not change, but they could have a high dynamic data rate. We also notice that some data values of an attribute from a source Website *S* vary more often than the values of other attributes in *S*, whereas others may remain nearly constant. Many applications process this type of data in which all tuples are important, with some attribute values being more "informative" than others. For example, among the desert weather data, temperatures vary tremendously between day and night times, whereas precipitation may be constant over weeks. However, among the weather data of coastal areas, temperatures vary slightly over days, whereas precipitation might change rapidly within the same day. These scenarios demand a new approach towards handling Web data with static schema by filtering less-informative attribute data in tuples, instead of the entire tuples, since a complete tuple may contain informative, as well as less-informative, data. In this paper, we propose an *attribute-based data filtering* approach on Web data with static schema which caters to this need, i.e., to detect and shed less-informative data and retain more-informative data. In addition, due to the presence of a wide range of Web data, such as weather, financial, medical, and traffic information, it is essential that any Web data filtering approach be adaptive so that it can handle the diversity of Web data. Our data filtering approach is unique, since it (i) chooses only the less-informative attributes from a source Website to be filtered, (ii) updates the load shedding scheme in real-time according to the data patterns in which rankings among attributes of a data source for choosing less-informative attributes achieve a high degree of accuracy, and (iii) is adaptive, which means it is applicable to any kinds of Web data.

Besides determining the data filtering scheme of a source Website, we also propose a *data recovery* method on filtered data by maintaining minimal amount of information about the data filtered at the source Website with low storage overhead. This information is used at a later time when filtered Web data are to be recovered. This occurs when the (user of the) destination Website needs some attribute data that have been filtered at the source

Website. To measure the high accuracy of our data recovery approach, we have conducted experiments on different Websites and compared the actual data with the recovered data. In addition, we have also conducted experiments on various components of our data filtering approach to verify its accuracy.

We proceed to present our results as follows. In Section 2, we discuss related works in Web data filtering. In Section 3, we introduce our data filtering approach, along with the proposed data recovery method. In Section 4, we include the experimental results on our data filtering and recovery approach to measure the merits of the overall design. In Section 5, we give a concluding remark.

2 Related Work

Many efforts have been made in the past to reduce less-informative data in structured and semi-structured data sources. In SimSearcher [TL03], a preprocessing unit extracts all common data patterns in a database. According to the user interest based on the user feedback, the data regions with data patterns having low user interest are pruned out as less-important database areas. SimSearcher, however, relies on user feedback and lacks automated nature.

Unlike SimSearcher, BirdsAnts [THA03] does not rely on any user feedback and is designed to provide complex protein mass structured data packed into an easily human-understandable form. Using a small amount of data, detailed information is provided to the user, while the information content is packed as the amount of data increases. The data packing is achieved by representing many cells of similar protein values by a single cell. Since the discovery of similar cells is achieved using clustering calculations, the discovery process of BirdsAnts is computationally expensive.

In [WVMA07], the authors propose a data filtering method applied to time series, i.e., a data stream, which monitors the time series for a predefined pattern. The method merges *similar* patterns together, which has the same effect as filtering redundant data as in our data filtering approach. The similarity between patterns is computed by using the Euclidean distance between two patterns. The authors also introduce a hierarchical

wedge-based comparison approach, which merges large number of patterns into a small set of wedges (with similar patterns being merged together) and then compares the set of wedges against the subsequence in subsequent data stream. Based on this design, the authors claim that monitoring of stream data at higher bandwidths becomes easier.

In [VKH07], the authors develop the Pairwise Attribute Noise Detection Algorithm for detecting and filtering attribute noise, i.e., noisy attribute data, by learning the relationships between the given attribute data or features. They examine pairs of attribute data sequentially and the deviations of the second attribute from its mean, given the discretized value of the first attribute. The deviation of the second attribute from its mean represents a departure from expected behavior. If this behavior occurs often or is severe enough compared with the remainder of the dataset, these specific instances are treated as noisy.

In [CK07], a solution to the data summarization problem of transaction processing with high *compaction gain* and *low information loss* is introduced. The summary of a given set of transactions is obtained by clustering the transaction data using any standard clustering algorithm, and each cluster is replaced by a representation which is its individual summary. Predefined weights are used to calculate the distance between two data transactions in the clustering algorithm, and this determines the number of clusters. Although their approach works well with data that are repeated frequently in transactions, it performs poorly when the data have outliers and less frequent patterns, since the outlying transactions are forced to belong to some cluster and the presence of even a single outlying transaction degrades the accuracy of the summary of the cluster.

Unlike our data filtering approach, none of the authors in [CK07, WVMA07, VKH07] consider filtering less-informative attribute data, the usefulness of which has already been explained.

Another instance of identifying the most-informative attributes in a database is found in the database alignment problem, which is the problem of aligning related databases together. In [PPH05], the authors handle the database alignment problem by using data-driven alignment. The key to their approach is to identify the most-informative

data elements and then match data sources that contain these informative elements. To identify the most-informative data elements, the authors use an information-theoretic model; that is data elements that contain words related to the data source are considered less-informative, whereas columns that contain words unrelated to the data source are considered more-informative. For example, the words *San Francisco* are more related to a data source containing San Francisco traffic information than the words *Los Angeles*, and thus San Francisco is less informative than Los Angeles.

In [FRPG01], the authors propose a tool called DELPHI for local similarity searching in biological sequence databases. DELPHI first preprocesses a database by extracting all common biological patterns. When a user searches this database, the database searching unit uses the common patterns as filters to prune out less-important database areas, leading to higher searching speed. Since DELPHI makes use of patterns in the biological database data to identify database regions, Delphi is not scalable for any databases such that there are no patterns in their data.

[BK05], who identify the pattern of change for a Web page, analyze many occurrences of a Web page *p* (after repeated processing) to define if, when, and where *p* has changed. [BK05] identify the parts of *p* that frequently change and how the changes on the same Web page are related to one another. This helps in breaking down a Web page into change zones and creates change relations. The change zones on any structured data on a Web page can also be identified, and the data within the change zones is considered to be more informative. Since [BK05] need intensive training for every Web page to identify the change zones, the approach in [BK05] lacks the dynamic nature.

RELIEF [KR92] uses a straightforward filter-based approach to determine the set of informative attributes in a database. Each attribute in a set is assigned a weight. If this weight is greater than a threshold, then the attribute is considered relevant to the application domain of the database; otherwise, it is dropped. Each weight is assigned according to the relevance of a random sample of its instances to the predefined application domain. Since RELIEF requires significant user assisted training to form the set of informative attributes, it is not automated.

In [AD91], the authors propose a tool, called FOCUS, which identifies a subset of a database features (attributes) that are sufficient to reconstruct a hypothesis that has been found to be true for the original database. FOCUS, which tries to find the subset of features, called MIN-FEATURES, that is sufficient to reconstruct the hypothesis using a training data set, is designed to perform exhaustive search to determine the MIN-FEATURES. Thus, MIN-FEATURES is the least number of features with which a consistent hypothesis can be constructed. Due to the involvement of exhaustive search, FOCUS has a high complexity, making it computationally very expensive.

In [BM03, SB02], the authors propose solutions to identify approximately duplicate records in databases by using character-based and vector-based textual similarity measures among records to determine if two records are duplicates. However, the systems incorporate domain specific knowledge to determine similar records, and require intensive training. [ME97], on the other hand, handle the problem of approximate duplicate detection by detecting clusters of similar records. A pair of records is considered to be in the same cluster if the minimum-edit distance to transform one record into the other record is less than the threshold value. Unlike [BM03, SB02, ME97], [HS95] uses a sliding window of fixed size to compare nearby records in a sorted database. Records that are found to be similar can either be merged or eliminated.

In [ACG02], the authors propose an algorithm for detecting duplicate tuples in hierarchical databases. Two tuples are considered to be duplicates if the respective pair of tuples in each relation of the hierarchy are duplicates. [ACG02] employ a straightforward duplicate detection algorithm to independently determine sets of duplicate tuples at each level of the hierarchy and then determine duplicate entries over the entire hierarchy. Nevertheless, the proposed method is designed for hierarchical databases, making it inapplicable to non-hierarchical databases.

The authors in [CGM05] propose a solution to detect and eliminate multiple distinct records representing the same real-world entity. To detect duplicated records, the authors suggest that entries that correspond to the same real-world object but have different representation in the database tend to (1) have small distances from each other, referred to

as *compact set property*, and to (2) have only a small number of other neighbors within a small distance, referred to as *sparse neighborhood property*. These two properties differentiate the duplicate elimination approach adopted in [CGM05] from standard clustering approaches. Like most of the duplicate detection approaches, [CGM05] also uses a distance-based approach to determine which records are close, which makes it prone to false positives in the presence of unrelated words that are lexically close.

In [BWL06], the authors introduce a framework for deleting duplicate data records from RFID (Radio Frequency Identification) data streams. RFID data streams are commonly used for tracking and monitoring physical objects in library checkin/checkout, highway tolls, etc., which lead to the creation of duplicate records. [BWL06] suggest retaining only the first (or the earliest) record within a sliding window and eliminating all new records, which are treated as duplicates. Although this approach eliminates all duplicates within a sliding window, it fails to eliminate duplicates that fall in different sliding windows, which increases the number of false positives.

WordNet [RSM94], which is a knowledge base with semantic knowledge, can be applied for information filtering. WordNet presents a semantic similarity measure that can be used for comparing two strings. The information in WordNet is organized around groupings, called synsets, and each synset consists of a list of synonymous words. Even though WordNet is widely used, its performance can deteriorate when the words are relatively rare, due to the scarcity of data.

In [BG04], the authors handle the problem of record linkage, which is the problem of determining if two records refer to the same entity, by considering the contents of both referencing and referenced tuples in order to make an accurate linkage decision. The approach can improve accuracy; however, in order to correctly identify all duplicates, it may need to make multiple passes over the data, making it computationally expensive.

Even though SimSearcher, [BK05], RELIEF, and WordNet are different systems with different design goals, they all require significant user assistance and training and thus lack the automated nature. Although not requiring significant user assistance, BirdsAnts, FOCUS, and [BG04] perform exhaustive searches and clustering, making them computa-

tionally expensive. While [PPH05], [BM03], and [SB02] rely on domain specific knowledge, DELPHI and [ACG02] lack scalability to non-biological and non-hierarchical databases. Comparatively, the data filtering approach presented in this paper is dynamic and automated, and does not impose high computational cost. Our data filtering approach is scalable and is not domain dependent. In addition, neither existing works treats duplicate/similar attributes as less-informative attributes, nor do they propose recovery design for filtered data, which is a significant component of the proposed work in this paper.

3 Our Data Filtering and Recovery Approach

In this section, we first present our data filtering strategy, which filters continuous data of less-informative attributes within the information provided by (i.e., flowed from) a particular source Website. We consider data segments, each of which can be viewed as the newly created data set of the continuous data at a source Website to be processed. The momentarily captured data are used for Web data processing.

With each incoming data segment of a source Website *S*, our data filtering approach first identifies the less-informative attributes, i.e., attributes whose data *vary less* when compared to the data of other attributes, in *S*. The major functions of our data filtering method include (i) creating the data filtering scheme of *S*[1], which enlists data to be filtered from *S*, and (ii) maintaining the information of filtered data of *S*, which are used during the data recovery process.

In the subsequent sections, we present the overall design of our data filtering and recovery model. Our data filtering approach preprocesses all data values from a Website *S* using *moving averages* to smoothen data values. Hereafter, preprocessed data are used to compute/reevaluate the data filtering scheme of *S*, which comprises the designated attributes of *S* and their data to be filtered. The data filtering scheme of *S* is then fed to the data recovery model, which determines the minimal amount of filtered data at the source Website for recovery purpose when the destination Website needs some attribute data that

[1] A *data filtering scheme* comprises of the attributes and their corresponding data to be filtered.

have been filtered at the source Website. During the recovery process, the source Website forwards the filtered data, or their best "approximated" values, to the destination Website to complete the recovery process.

3.1 Exponential moving average

Before the data filtering scheme can be determined from the data of a source Website *S*, we first compute the *Exponential Moving Average* (*EMA*) [SSS00] of the data of each attribute in the current data segment *DS* of *S*, which smoothens the variations of data in *DS*, the core of the preprocessing step.

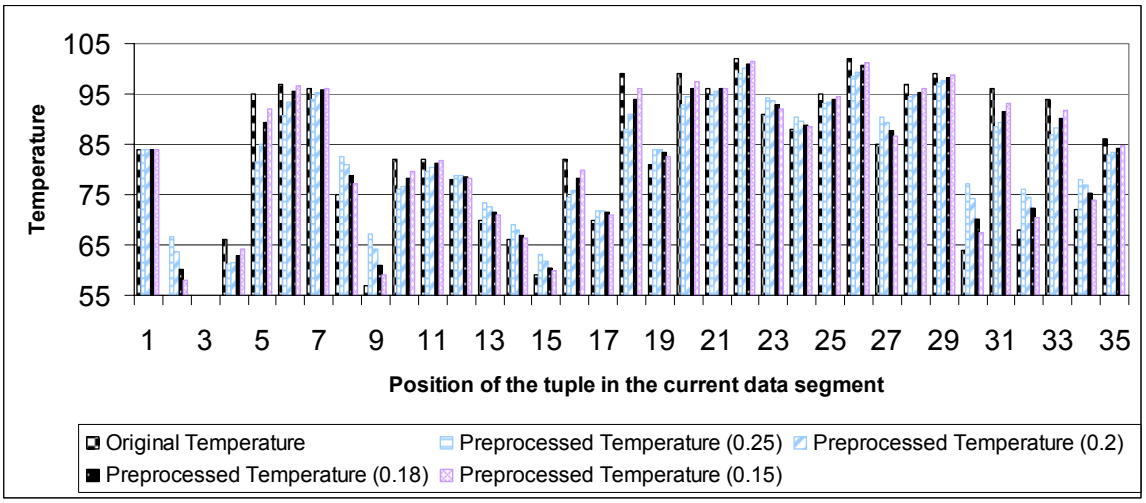
Though the abrupt change in values of a particular attribute *A* in the current data segment *S* does not really represent the data pattern of *A* on a regular, consistent basis, it may cause other informative attributes being treated as less-informative (false positives) and the real less-informative attributes as informative (false negatives). In order to (i) smoothen the attribute data, (ii) suppress any short and sudden change in data, and (iii) reduce the false positives and false negatives, *EMA* is used as a preprocessing step for determining less-, as well as more-informative attributes in *S*, since *EMA*s attempt to tone down the fluctuations to a smoothened trend so that distortions are reduced to a minimum.

Because the most recent value in a list of values *L* is the newest value of *L*, it is used along with other older values in *L* to calculate its *EMA*, and the older values in *L* are the values which have already been used to calculate its previous *EMA*s. The *EMA* of a particular data value, an, in *L* is defined as

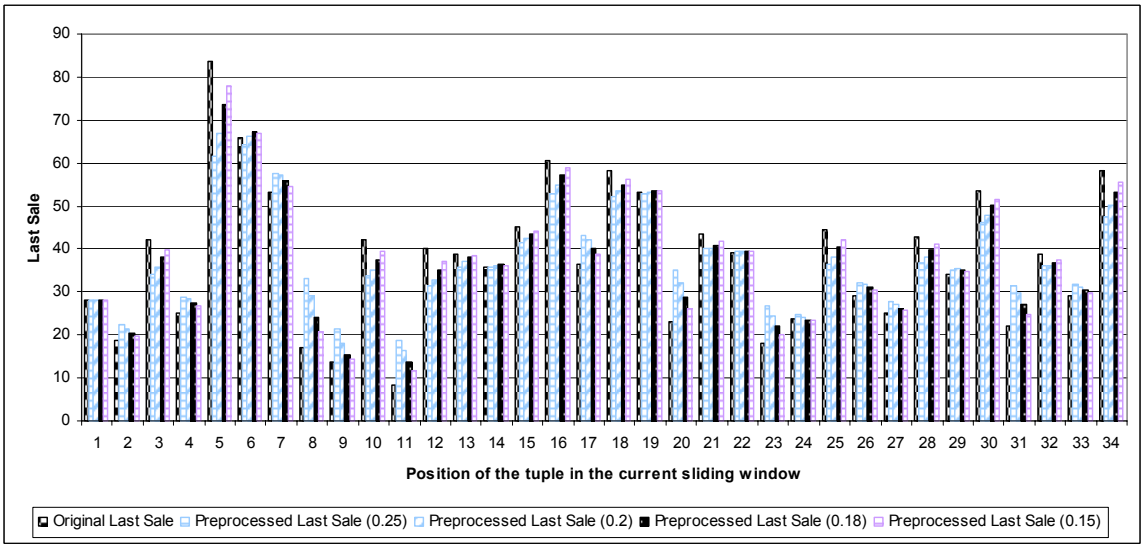
$$EMA(a_n) = (a_n - EMA(a_{n-1})) \times \text{Multiplier} + EMA(a_{n-1}) \quad (1)$$

where $EMA(a_n)$ ($n \geq 1$) is the *EMA* for the most recent value a_n in $L = a_1, ..., a_n$, and Multiplier is the weight used in computing the *EMA* of the most recent value a_n. We calibrate the value of the multiplier to 0.18 as shown in Figure 1, where the preprocessed data with multiplier set to 0.15 are very close to the original data, and thus the preprocessed data would not have been smoothened, whereas the preprocessed data with multiplier set to 0.2 and 0.25 are not close to the original data, creating a chance of losing the variation

characteristics of the original data. The preprocessed data with multiplier set to 0.18 are not too close to the original data, and after they are smoothened, they are still close enough to retain the variation characteristics of the original data. Since *EMA* has the ability to stay closer to the actual data value, it is an obvious choice for our preprocessing step to smooth the data in a data segment. The *EMA* approach is called *exponential* because of the use of the *exponential moving averages*, which considers the exponential allocation, rather than equal allocation, of weights to the past values to smoothen the current value.



(a) Preprocessing the *Temperature* data of http://weather.yahoo.com



(b) Preprocessing the *Last Sale* data of http://quotes.nasdaq.com/quot=nasdaq100

Figure 1: Preprocessing data in a weather and a stock exchange from their corresponding Websites with data downloaded on September 13, 2005 with different multiplier values

3.2 Data filtering scheme generation and reevaluation

There are two major design issues in data filtering scheme generation: (i) how much data should be filtered, and (ii) which attributes should be filtered from a source Website.

3.2.1 *Amount of data to be filtered*

The amount of data to be filtered can be simply fed by the user or computed automatically. We present one such automated process for computing the amount of data to be filtered among numerous other choices.

According to various studies in computer networks, it is well-known that the capacity of a channel depends on the noise and bandwidth of the channel. Based on the Shannon theorem [Sha49], if data are transmitted at a rate R *higher* than the channel capacity C, then data transmission *errors* and collisions occur exponentially, and no useful information can be transmitted beyond C. To overcome these problems, R should be lower than C. Thus, when $R > C$, attributes are filtered from the corresponding data source being transmitted over the network (i.e., through the Internet), starting with filtering the less-informative attributes in our data filtering approach. Depending on the sizes of the different attributes, one or more less-informative attributes will be filtered so that R falls below or is equal to C. Hence the rate at which data has to be filtered is

$$r = R - C. \tag{2}$$

Example 1. Consider a channel with capacity C of 140 Kbps and an attempt to transmit Web data with seven attributes at a transmission rate R of 184 Kbps. Assume that the sizes of the attributes, starting from the least-informative attribute, are 16, 32, 32, 8, 64, 16, and 16 Kbps, respectively. Since $R > C$, the transmission would produce errors. To attain error-free transmission, the amount of attribute data should be filtered using Equation 2 is 44 = (184 – 140) Kbps. To meet the required amount of attribute data to be filtered, the two less-informative attributes with sizes 16 Kbps and 32 Kbps, i.e., 48 Kbps, are filtered. □

3.2.2 Attribute filtering using standard deviation

Having determined the amount of attribute data r ($r \geq 1$) to be filtered from a data segment

DS of a data source *S*, we must decide which attributes should be filtered from *DS*. *Standard deviation* can be used as a measure to determine a ranking of attributes, which applies to each data segment of *S* and the ranking is recomputed for each data segment.

Standard deviation (*SD* for short) is a measure of the spread of a list of data values (a data segment *DS* in our case) from the mean value. A large *SD* indicates that the corresponding data points in *DS* are far from the mean, whereas a small *SD* indicates that they are clustered closely around the mean. We conjecture that data values that are more closely bound, i.e., have less variation, are "less-informative," whereas values that are less closely bound, i.e., have more variation, are "more-informative." Since *SD* is a measure of how closely bound data values in a list are, we apply *SD* to the data values of each attribute *A* in *DS* to calculate how closely the data values of *A* are. The attributes in *DS* with the lower *SD* are treated as less-informative attributes and are ranked higher, which are candidates to be filtered. If the domains of different attributes cover significantly different ranges of data values, the *SD* is applied to the attribute values that have been normalized; e.g., values in each *DS* are normalized by using the highest value of their corresponding attribute in *DS*, so that the *SD*s of all the attributes in *DS* become compatible.

3.2.3 Reevaluation of a data filtering scheme

It is required that the data filtering scheme of a source Website *S* be regularly reevaluated because the *SD*s of different attributes in subsequent data segments of *S* may change, causing the ranking among the attributes to change. A non-adaptive data filtering scheme reevaluation algorithm reevaluates the data filtering scheme at regular intervals. One major problem with using a non-adaptive reevaluation algorithm is that if the time interval is *too short*, the source Website would be reevaluating the data filtering scheme too often, which imposes the burden on the source Website in terms of computational time required for reevaluation. However, if the time interval is too large, the source Website would not reevaluate the data filtering scheme *often enough*, creating the risk of an obsolete data filtering scheme being used for a long time. The proposed reevaluation algorithm in this paper, i.e., Algorithm 1, resolves the time-interval problem, since Algorithm 1 is adaptive,

which starts out with a very small reevaluation time interval, referred as the original time interval. For the first time, Algorithm 1 reevaluates an existing data filtering scheme after waiting for the original time interval, and then checks if the reevaluated (i.e., the newly generated) data filtering scheme of the current data segment with smoothened data (due to *EMA* preprocessing) is different (in terms of attributes to be filtered) from the previous data filtering scheme (computed by using the previous data segment with smoothened data). If the attributes to be filtered are the same, the time interval is doubled so that the reevaluation is invoked after a *longer* time interval. If the attributes to be filtered are different, then (i) the time interval is reset to the original time interval, since a change in the data filtering scheme has just been detected and we anticipate changes in the data filtering scheme in near future, and (ii) the data filtering scheme is also updated to be the modified data filtering scheme with new attribute(s) and data to be filtered. The time interval grows in its usual manner every time the anticipated change in the attributes to be filtered is proved incorrect. In addition, every time the data filtering scheme is updated, a copy of the data filtering scheme is sent to the destination Website. This practice informs the destination site about what attributes, if any, are being filtered.

Algorithm 1. *Data filtering scheme reevaluation*

Input: (i) The set of tuples S in the current data segment, on which data filtering has to be performed, and (ii) the current data filtering scheme C

Output: The (updated) data filtering scheme

1. Initialize time T:= current clock time; Δt:= t:= 1 sec
2. Loop
   If current clock time = $T + \Delta t$, then recompute the data filtering scheme using S
   (i) If the recomputed data filtering scheme $RS = C$, then
   $\Delta t := 2 \times \Delta t$
   Else (a) $\Delta t := t$
   (b) C:= RS
   (ii) T := current clock time
   End Loop

Our adaptive data filtering scheme reevaluation algorithm enjoys a major advantage over its non-adaptive counterpart, since the adaptive version notices the change more accurately. Consider a data source in which attribute *A* is the less-informative attribute during the first thirty seconds of every minute and attribute *B* is the less-informative attribute during the last thirty seconds of every minute. Assume that one attribute needs to be filtered, and a non-adaptive data filtering scheme reevaluation algorithm *E* is invoked every minute, starting five seconds past each minute. Since *A* is the less-informative attribute during the first thirty seconds of every minute, every time *E* is invoked, *A* is found to be the less-informative attribute and is filtered, and the data filtering scheme never changes. In such a scenario, *E* would fail to notice the change in the data filtering scheme. However, if Algorithm 1 is used instead, every time Algorithm 1 is invoked within the first thirty seconds of the current clock minute, it finds *A* as the less-informative attribute, and it would double the current value of Δt. Eventually Δt would reach a value such that the current clock time $+\Delta t$ would fall within the last thirty seconds of the current clock minute, resulting in *B* being detected as the less-informative attribute. Thus our adaptive data filtering scheme reevaluation approach provides a more accurate mechanism in detecting a change in a data filtering scheme.

3.3 Data segment size

Since the data filtering scheme is generated from the current data segment, there is a need to identify the start and end of a data segment, which determines the *data segment size*. Different segments of a data source Website *S*, which convey up-to-the-moment information, determine the *segment identifier* (*SID*, for short), which is defined as either a single attribute or a combination of attributes of *S*. The *SID* of *S* also serves as the unique identifier in the recovery matrix of our data recovery approach where recovery information of *S* are recorded and extracted. The *SID* values of *S* should follow a fixed-length repetitive cycle of data in *S*, which consists of tuples from *S* such that the order of appearances of various *SID* values in the tuples falls in the same cycle, and the number of tuples in each fixed-length repetitive cycle of *S* is called the *data segment length* of *S*. The data segment

length of S is treated as the *size* of each tuple-based *data segment* of S for data filtering and data recovery purpose. Since the *SID* of S must be transmitted to the destination Website and cannot be filtered as it is used for data recovery, the *SID* should be minimal, i.e., with the fewest possible attributes that individually come with a segment length of non-repetitive values in each data segment of S. In this section, we discuss a method in determining the *SID* of S and thus the *segment length* of S.

The *SID* of S is detected during the training phase of S, which is carried out before our data filtering system actually starts filtering data and retaining recovery information from S. During the training phase of S, we analyze and evaluate all the data values for each attribute in the training set of S. Each attribute in S individually follows a repetitive pattern of their values from the set of replicated attributes (*RepA*s, for short) of S, and whenever a replicated attribute is detected, its segment length is also recorded. The replicated attributes in *RepA*s are partitioned into sets $S_1, S_2, ..., S_n$ ($n \geq 1$) such that each S_i ($1 \leq i \leq n$) contains all the attributes with the same segment length. Furthermore, all the attributes in each S_i have a one-to-one relationship with each other. We define a one-to-one relationship set, which is a set E, such that the value of each attribute in E can uniquely identify the values of all the other attributes in E; i.e., each attribute in E has a one-to-one relationship with every other attribute in E. Thus the one-to-one relationship is simply the functional dependency constraint in the RDBMS; i.e., $A \rightarrow B$ if and only if $t_1[A] = t_2[A] \Rightarrow t_1[B] = t_2[B]$ for each t_1, $t_2 \in r$, where r is the data segment in our data filtering approach that is currently being processed. Since the replicated attributes in each S_i have a one-to-one relationship with all the other attributes in S_i, only one attribute from each S_i is needed to form (a part of) the *SID* of S, and the segment length of each chosen attribute is used to compute the fixed segment length of S.

The replicated attributes in each one-to-one relationship set S_i may have different bit length; i.e., the numbers of bits occupied by the domain values of different attributes in S_i can be different. We choose (one of) the attribute(s) with the *least* bit length from each one of $S_1, S_2, ..., S_n$, which guarantees that the chosen attributes to form the *SID* of S have the minimum bit length among all the other possible combinations of replicated attributes,

and thus the *SID* is *minimal* in size. The amount of computation required to identify the *SID* of *S* is one time and does not impose a lot of burden on the source data Website. Once identified, the destination Website can use the *SID* to determine the *SID* value of a filtered tuple to be recovered.

In finding *RepA*s of *S*, we compare every tuple, starting from the 2nd tuple, in the training data set of *S* with the 1st tuple in the training data set, till we have found the first repeated value of an attribute. The first repeated value of an attribute indicates a potential repetition data segment of the attribute values. Assume that there are p ($p \geq 1$) distinct tuples in the training set, and p is sufficiently large, i.e., there are sufficient training data to identify all the replicated attributes in *S*. Further assume that the first match in the comparisons to find the repetition data segment of the values of any attribute A is found in the j^{th} ($1 < j \leq p$) tuple and the 1st tuple. This discovery will be followed by the comparison on the values of A in the j+1th tuple and the 2nd tuple to determine whether the two tuples have the same value on A. If the values of A are the same, the comparison is followed by yet another comparison between the attribute values of A in the j+2nd tuple and the 3rd tuple, and so on till all the tuples in the training set are covered. If any one of these comparisons fails, then A is not replicated. For every attribute A in *S*, a total of p comparisons for A would have to be carried out. Assuming that there are q ($q \geq 1$) attributes in *S*, discovering the *RepA*s of *S* would require O($p \times q$) computations to find all the replicated attributes of *S*.

Example 2. Suppose a data Website *S* has attributes A_1, A_2, A_3, and A_4, and suppose $RepA_s = \{A_1, A_2, A_3\}$ such that $Seglen(A_1) = 3$, $Seglen(A_2) = 3$, and $Seglen(A_3) = 4$. Partitioning the attributes in $RepA_s$ into sets, with each set containing attributes of the same segment length, yields sets $S_1 = \{A_1, A_2\}$ and $S_2 = \{A_3\}$. Assume that the bit lengths for attributes A_1, A_2, and A_3 are 4, 8, and 2 bytes, respectively. Thus, the *SID* of *S* is $Min(S_1) \cup Min(S_2) = Min(\{A_1, A_2\}) \cup Min(\{A_3\}) = \{A_1\} \cup \{A_3\} = \{A_1, A_3\}$, and the segment length of *S* is $SegmentLength(S) = Seglen(A_1) \times Seglen(A_3) = 3 \times 4 = 12$. □

The accuracy of our *SID* discovery method relies on the correctness of training set data. If the attribute values in a training set data have errors, causing the loss of repetitions

of attribute values, then the *SID* of the corresponding data of the source Website may not be detected correctly. These errors are sometimes referred to as *bit-errors*, because the bit(s) in a byte of a data value is (are) changed from a '0' to '1', or vice versa. One widely used method to detect and correct these errors is the *Hamming Code* [Ham50], an error correcting code, which is a widely accepted error detection technique in computer networks; and it can be adopted by our *SID* discovery method to ensure that each training data set is error free.

3.4 Recovering filtered data

As discussed earlier, when needed, less-informative attributes are filtered at the source Website. It is possible that an end user at the destination Website is interested in some of the attribute data filtered at the source Website. We present a *synopsis recovery approach* to recover data at the destination Website.

The basic idea behind our synopsis recovery approach is to store the minimal amount of information on the filtered data in a matrix at the source Website before attribute data are filtered. Since storing all the filtered values would require a significant amount of memory/disk space and computational power at the source Website, one of the design goals of the proposed synopsis recovery method is to minimize the storage and computational power requirements.

As mentioned before, we use the *SID* of a tuple for the recovery of the tuple. Our data recovery process involves two major modules. The first module, Module 1, maintains a synopsis matrix of the filtered data at the corresponding source Website *S*, whereas the second module, Module 2, is the real recovery process when the destination Website requests some filtered data from *S*, which requires *S* to extract (approximated) filtered data values in its synopsis matrix in response. Here, we first introduce the concept of error threshold value, which is the acceptable error rate in data recovery that dictates the amount of filtered data to be stored in a synopsis matrix and can be determined by the channel capacity or other constraints. (Note that we do not consider the processing capacities of the source Website and the destination Website in determining the error threshold value, since

we assume that the two sites have high computational power.)

Along with the error threshold value, the criticality of the application domain of the source data is also considered, since the error threshold value varies from one application domain to another. For example, a patient information monitoring system would require more accuracy in recovered data than a weather information monitoring system. The synopsis matrix stores a value to be filtered only when the value has been changed more than the error threshold value from what its last stored value was. Recovery data values are stored in a 3-dimensional synopsis matrix, with one dimension corresponding to each component: (i) the value of the unique identifier of a current tuple t, i.e., $SID(t)$, (ii) attribute A, i.e., $t[A]$, to be filtered, and (iii) the timestamp for $SID(t)$. A timestamp is recorded in the synopsis matrix whenever a filtered data value is recorded, which could be used to extract filtered data values from the synopsis matrix at a later point in time, which is used in Module 2.

Algorithm 2 is the synopsis recovery algorithm for Module 1. Every data value before being filtered is compared with the most recent value of the same attribute stored in the matrix. If the change, i.e., the difference between the most recent value mv stored in the matrix and the value to be filtered, sv, is more than the error threshold value E, then sv is stored in the matrix; otherwise, sv is not stored, i.e., sv is stored in the synopsis recovery matrix only when $(|mv - sv| / sv) \times 100 > E$.

Algorithm 2. *Synopsis Recovery Algorithm*

Input: (i) The set of tuples S in the current data segment from where attribute data are to be filtered, (ii) attribute A to be filtered, where the value of A in tuple t is denoted by $t[A]$, (iii) $SID(t)$, (iv) the error threshold value E, and (v) the synopsis matrix, *Synopsis*

Output: The (updated) synopsis matrix, *Synopsis*

For each tuple $t \in S$

(a) mv := Synopsis[SID][A][$Timestamp(SID)$]

(b) $sv := t[A]$

/* The change in the value to be filtered from the most recent value stored in the matrix

is greater than the error threshold */

(c) If $(|mv - sv| / sv) \times 100 > E$, then /* Increase the timestamp and store the value to be filtered in the synopsis matrix */

(i) If $Timestamp(SID) = null$, then

$Timestamp(SID) := 0$

Else $Timestamp(SID) := Timestamp(SID) + 1$

(ii) $Synopsis[SID][A][Timestamp(SID)] := t[A]$

The correctness of our filtered data recovery approach is verified by experiments that show the accuracy in recovering filtered data using the synopsis recovery approach to store recovery data in the synopsis matrix. Preliminary experiments showed that high recovery accuracy can be achieved at the cost of storing a very low percentage of the data filtered as recovery data. We performed preliminary experiments on randomly chosen weather and stock exchange source Websites to verify the gain of using the synopsis recovery approach to maintain recovery data. Table 1 shows that, using our synopsis recovery approach, we store much less filtered data, 1.11% - 19.44% (4.88% -51.11%, respectively), of the original data with 90% - 99% data recovery accuracy for a weather Website (stock exchange Website, respectively) in the synopsis recovery matrix as recovery data.

	Weather Information				**Stock Exchange Information**			
	90%	**95%**	**98%**	**99%**	**90%**	**95%**	**98%**	**99%**
Percentage of filtered data to be stored in the synopsis matrix as recovery data for different data segments	2	2	4	19	4	17	35	47
	1	3	5	21	6	21	50	56
	0	2	6	18	5	18	41	51
	2	2	5	16	4	17	38	48
	1	4	4	22	7	19	56	62
	1	1	7	23	3	15	32	42
	0	2	3	17	6	21	52	56
	1	3	2	19	4	17	37	46
	2	4	5	20	5	18	43	52
Average%	**1.11**	**2.55**	**4.55**	**19.44**	**4.88**	**18.11**	**42.66**	**51.11**

Table 1: Different data recovery accuracy ratios over nine experiments for the source Web-site: (i) weather (www.yahoo.com/weather) retrieved on September 11, 2005 and (ii) stock exchange (http://quotes.nasdaq.com/quote.dll?page=nasdaq100) retrieved on September 13, 2005 with filtered data that were stored in the synopsis matrix

3.4.1 Error threshold value

Our data recovery method applied to a source Website *S* must satisfy two criteria in order to perform well: (i) the amount of data in *S* to be stored in its synopsis matrix should be low, since we do not have infinite disk space for storage, and (ii) the error in recovered data, i.e., the recovery error rate, of *S* should be low. Considering the two tasks closely, these two measures are inversely proportional to each other. When we attempt to decrease the amount of synopsis data to be stored, the recovery error rate increases and thus suffers, whereas when we attempt to improve the recovery error rate by decreasing it, the amount of synopsis data to be stored would increase proportionally. The optimal performance can be achieved by maintaining a balance between these two trade-offs, which vary from one particular data application domain, i.e., data processed at a source Website, to another. For example, in critical data application domains (such as medical information), a recovery error rate of 10% may be too high (i.e., inadequate). On the other hand, in a less critical data application domain (such as weather), a balance with higher recovery error rate at the benefit of less amount of data to be stored in the synopsis matrix is acceptable. Here, we introduce (i) the fixed point for a Web data source, (ii) the various categories that define the criticality of the data source, and (iii) the category recovery error rate. The minimum of (i) the fixed point for a data source *S* and (ii) the category recovery error rate of the corresponding category with certain degree of criticality to where *S* belongs determines the error threshold value of *S*.

3.4.2 The fixed point

We first consider the size of the synopsis data of a Web data source S. Based on our observation, as the recovery error rate of *S* increases from 0% to 100%, the amount of synopsis data of *S* decreases to a point *P*, beyond which any further increase in the recovery error rate does not affect the amount of synopsis data; i.e., the amount of synopsis data remains constant beyond *P*, which is referred as the *fixed point* of *S*, and is computed for *S* only once. The existence of such a fixed point for *S* can be justified by virtue of the fact that the variation in the values of a less-informative attribute *A* in *S* (to be filtered) is finite,

which is further strengthened by the fact that since *A* is a less-informative attribute, it is (one of) the least varying among the other attributes of *S*; i.e., the variation between the values of *A* would be between 0% and *x*%, where *x*% is the percentage difference between the minimum and the maximum value of *A* in a data segment of *S*. The amount of data to be stored in the synopsis matrix ceases to decrease as the recovery error rate is greater than or equal to *x*%. We determine the fixed point for a data segment by plotting a graph using the training data of *S*, which captures the amount of synopsis data at each recovery error rate of *S*, increasing from 0% to 100%.

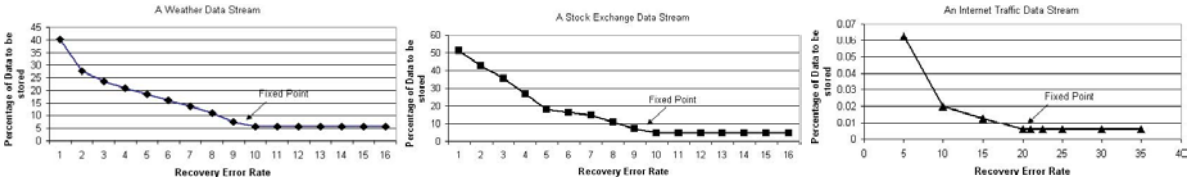
Figure 2 shows the average amount of data to be stored in the synopsis matrix at different recovery error rates using 10 experiments on each of the three different data application domains: weather, stock exchange, and Internet traffic, with data downloaded from these different Web data sources over a 2-hour period on October 25, 2005, which were split into 10 sets to conduct the 10 experiments on each of the Web data sources. The results from the 10 experiments were then averaged. The averages yield the potential recovery error rates (i.e., fixed points) of 9.5%, 9.2%, and 20%, for weather, stock exchange, and Internet traffic, respectively. The graphs in Figure 2 show that the amount of data to be stored in each of the synopsis matrices becomes constant beyond the fixed point, as anticipated.

3.4.3 Category recovery error rate

Although the fixed point approach can be adopted to determine the error threshold value of a Web data source *S* automatically, it lacks the ability to incorporate the criticality of data in *S*. Here, we propose an automated *category recovery error* rate detection mechanism on *S* based on the attributes of *S*, which together with the fixed point approach determine the *error threshold value* of *S*. The detection process is done only once, i.e., prior to processing any data from *S*.

The attributes of *S* are analyzed and automatically matched with the attributes of a data application domain in each of the predefined categories, which include the *Extremely Critical*, *Very Critical*, *Moderately Critical*, *Low Critical*, and *Not Critical* categories.

Each of these categories is assigned a number of data application domains, such as the medical information data and emergency response data in the *Extremely Critical* category, and each category is associated with different predefined lists of attributes commonly found in the data of the same nature that best defines the corresponding application domain. We have predefined some of the commonly used data application domains in each category, e.g., medical information, stock exchange, Internet traffic, weather, and population in the *Extremely Critical*, *Very Critical*, *Moderately Critical*, *Low Critical*, and *Not Critical* categories, respectively. If the attribute names in a (new) Web data source *S* do not "match" any predefined list of attributes of any one of the data application domains in any predefined category, then *S* is assigned to the category *others*, which has a category recovery error rate of "infinite."



(a) Weather (b) Stock exchange (c) Internet traffic

Figure 2: Average amount of 10 different data sets to be stored in the synopsis matrix at different recovery error rates for different data sources, weather (http://weather.yahoo.com), stock exchange (http://quotes.nasdaq.com/quote.dll?page=nasdaq100), and Internet traffic (http://www.Internet traffi-creport.com) , extracted on October 25, 2005

In matching the list *L* of attributes for a new Web data source with a predefined list of attributes *P*, we adopt the Fuzzy set IR model in [GN06] to compute the degree of similarity between *L* and *P* using the distance matrix [GN06], in which row and column headings are words appearing in commonly used dictionaries. The distance matrix captures the degrees of similarity (i.e., correlation factors) among different words, which was generated using a set of 880,000 Wikipedia (http://wikipedia.org/) documents to compute the frequency of co-occurrence and relative distance of each pair of words in each Wikipedia document. Furthermore, we adopt the *EQ* function[2], which is defined below, to

[2] The two threshold values in *EQ* are adjusted empirically for the purpose of computing the equality between any two lists of attribute names, and are presented in detail in Section 4.

decide if any two lists of attributes should be treated as the same using the correlation factors in the *Sim* function among the attributes, i.e., words, in the distance matrix.

$$EQ(S_i, S_j) = \begin{cases} 1 & \text{if } MIN(Sim(S_i, S_j), Sim(S_j, S_i)) \geq Permission\ Threshold\ \wedge \\ & \quad |Sim(S_i, S_j) - Sim(S_j, S_i)| \leq Variation\ Threshold \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

where S_i and S_j ($i, j \geq 1$) are lists of attributes, and $Sim(S_i, S_j)$ is the degree of similarity between the attributes in S_i and the attributes in S_j.

After detecting the category *C* that contains a list of attributes that should be treated as (semantically) equal to the list of attributes in a new Web data source *S*, *S* is then assigned to *C*. We conducted 10 different experiments on randomly chosen Web data sources of stock exchange, weather, and Internet traffic application domains, which demonstrate the accuracy of the Fuzzy set IR model approach in assigning various Web data sources to categories, and the results showed 90% accuracy. (The 10% inaccuracy was due to the false positives and false negatives in matching the [semantically the same] attributes between two lists of attributes.)

Note that the list of attributes in a Web data source is quite "narrow," which means that two Web data sources belonging to the same application domain often contain almost (semantically) the same set of attribute names. For example, almost all data sources in the weather application domain contain the (semantically the same) attributes *location*, *temperature*, *humidity*, *precipitation*, *sunrise*, *sunset*, and *wind*, whereas *precipitatio*n and *rain* have very high correlation factor and are treated as the same.

Predefined with each of the first five categories is a category recovery error rate, and the 6th category, i.e., "others," has the category recovery error rate of infinite as mentioned earlier. Each category recovery error rate of the first five categories is computed empirically during the design of our data filtering approach, using the average fixed points of various Web data sources in each category. The empirically determined category recovery error rates of the first five categories are 1.67%, 4.7%, 4.81%, 13.5%, and 20.33% for the *Extremely Critical*, *Very Critical*, *Moderately Critical*, *Low Critical*, and *Not Critical* categories, respectively (see detailed discussion on the error rates in Section 4).

Given (i) the fixed point of the new Web data source *S* that has been automatically detected and (ii) the category recovery error rate of *S*, our data recovery approach assigns the error threshold value of *S* as the lower of the two, since the fixed point and the category recovery error rate are the maximum acceptable error rates in the recovered data for *S*. (Thus, if *S* belongs to the others category, then the fixed point value of *S* is used as the error threshold value of *S*.) Note that the fixed point value of *S* is often different from the category recovery error rate of *S* because the former is computed for each new data source, whereas the latter is the average of the fixed point values for a number of data application domains (including the one for *S*) belonging to that category.

Example 3. Consider the categories for the data sources as shown in Figures 2(a), 2(b), and 2(c), which are *Low Critical*, *Very Critical*, and *Moderately Critical*, respectively, with category recovery error rates 13.5%, 4.7%, and 4.81%, respectively. Comparing the category recovery error rates with the fixed point values, i.e., 9.5%, 9.2%, and 20%, respectively for the Web data sources shown in Figures 2(a), 2(b), and 2(c), respectively, the error threshold values for these Web data sources are 9.5%, 4.7%, and 4.81%, respectively, i.e., the lower of their respective fixed points and their corresponding category recovery error rates. □

4 Experimental Results

Our data filtering and recovery design was implemented in Java and tested on a Windows/Linux PC with a 3.2 GHz processor, 3.25 GB RAM memory, and 150 GB of hard disk space with Linux shell scripts running test scripts. Experiments were conducted to verify the correctness of our data filtering and recovery approach in real time on the four major modules: the (i) SID detection, which also verified our approach on configuring the *data segment size* of a Web data source, in Section 4.1, (ii) error threshold value generation, for which we verified the accuracy in predicting the error threshold value of a Web data source, in Section 4.2, (iii) data filtering scheme (reevaluation) and ranking generation approach, which tested the accuracy of choosing the less-informative attributes in a Web data source to be filtered, in Section 4.3, and (iv) filtered data recovery, for which we

verified the accuracy of recovering filtered data, in Section 4.4. In addition, we compare our data filtering and recovery approach with existing ones in Section 4.5 to show the merits of our work. Test and training data used in the experiments were extracted from various Web data sources of different application domains: weather, stock exchange, and Internet traffic, with the sizes of 15.6 GB, 21 GB, and 12 GB, respectively (as shown in Table 2).

4.1 Experimental Results on SID Detection

If the attribute(s) of a training Web data source *S* chosen automatically as the *SID* of *S* has (have) the same replicated values (in the same sequence) within each data segment of *S*, then the accuracy of our SID detection approach is confirmed. Note that the *data segment size* of *S* is also verified along with the detection of the *SID* of *S*, since the segment length of the detected *SID* of *S* yields the corresponding data segment size. In verifying the correctness of detecting *SID*s, we used the first 10 MB of the data in Table 2 as the training data and the remaining ones as test data. The experimental results are shown in Table 3.

Since each test data set was large in size, which made manual examination on the (correctly) detected *SID*s infeasible, we verified that the detected *SID*s are in fact the *SID*s of the corresponding data sources using scripts; in addition, we manually examined a number of randomly selected *SID*s that were detected automatically. Based on the results as presented in Table 3, we conclude that the *SID* detection approach is 100% accurate.

4.2 Experimental Results on Error Threshold Value Detection

We show how to determine the permissible and variation threshold values of *EQ* in Section 4.2.1 and verify the accuracy in detecting the (i) fixed point and (ii) category recovery error rate, which determine the error threshold value of a Web data source, in Section 4.2.2. Note that the accuracy of *EQ* affects the accuracy of the category recovery error rate.

	On Each Data Set					
Data Source	**Attributes of the Data Source**	**Size (GB)**	**Number of Tuples**	**Sliding Window Size**	**Number of Sliding**	**Collected On**

					Windows	
Weather Information	**15.6**					
http://weather.yahoo.com/ - Sets 1, 2, 3	Location, Temperature, Dew Point, Barometer, Wind, Humidity, Sun Rise, Visibility, Sun Set	2	28036790	75	373823	1-Mar-06
http://www.wunderground.com/ - Sets 1, 2, 3	Location, Temperature, Dew Point, Humidity, Wind, Pressure, Precipitation	1.2	37063068	500	74126	…
http://www.weather.com - Sets 1, 2, 3	Location, UV Index, Wind, Humidity, Pressure, Dew Point, Visibility	2	32819280	250	131277	…
Stock Exchange Information	**21**					
http://quotes.nasdaq.com/quote.dll?page=nasdaq100 - Sets 1, 2, 3	Symbol, Company Name, Last Sale, Net Change, Share Volume, Nasdaq 100 Index, Percentage Change	3	52516326	100	525163	…
http://finance.indiamart.com /markets/bse/ - Sets 1, 2, 3	Company Name, Last Price, Change, Percentage Change, Market Cap, Weight	3	64860370	488	132910	…
http://www.channelnewsasia.com /cna/finance/sg/stockmonitor.htm – Sets 1, 2, 3	Stock, Buy, Sell, Last Done, Volume	1	33288126	1235	26953	29-Mar-06
Internet Traffic Information	**12**					
http://www.Internettrafficreport.com – Sets 1, 2, 3	Router Name, Current Index, Response Time, Packet Loss, Minimum Delay, Average Delay, Maximum Delay	2	16332958	96	170134	1-Mar-06
http://average.miq.net/index.html - Sets 1, 2, 3	Router, Response Time, Packet Loss, Minimum Delay, Average Delay, Maximum Delay	1	11689810	50	233796	…
http://watt.nlanr.net/active/maps/ ampmap_active.php - Sets 1, 2, 3	Site Name, Min, Mean, Max, StdDev, Loss	1	10866072	86	126349	…

Table 2: Web data sources of test/training data collected on March 1, 2006 (except for http://www.channelnewsasia.com/cna/finance/sg/stockmonitor.htm, which was collected on March 29, 2006)

Data Stream Source	**Training Data on Each Set**		**Test Data on Each Set**	
	Size (MB)	**Detected CID**	**Size (GB)**	**Detected CID**
Weather Information	**90**		**15.51**	
http://weather.yahoo.com/ - Set 1, Set 2, Set 3	10	Location	1.99	Location
http://www.wunderground.com/ - Set 1, Set 2, Set 3	…	…	1.19	…
http://www.weather.com - Set 1, Set 2, Set 3	…	…	1.99	…
Stock Exchange Information	**90**		**20.91**	
http://quotes.nasdaq.com/quote.dll?page=nasdaq100 - Set 1, Set 2, Set 3	10	Company Name	2.99	Company Name
http://finance.indiamart.com/markets/bse/ - Set 1, Set 2, Set 3	…	…	…	…
http://www.channelnewsasia.com/cna/ finance/sg/stockmonitor.htm - Set 1, Set 2, Set 3	…	Stock	0.99	Stock
Internet Traffic Information	**90**		**11.91**	
http://www.Internettrafficreport.com - Set 1, Set 2, Set 3	10	Router Name	1.99	Router Name
http://average.miq.net/index.html - Set 1, Set 2, Set 3	…	…	0.99	…
http://watt.nlanr.net/active/maps/ampmap_active.php - Set 1, Set 2, Set 3	…	…	…	…

Table 3: Training and test data results for CID detection

4.2.1 Verification of the permissible and variation threshold values in *EQ*

In defining the threshold values in the *EQ* function, we empirically adjust the *permissible* threshold value, which defines the *minimum similarity* between two lists of attributes S_1 and

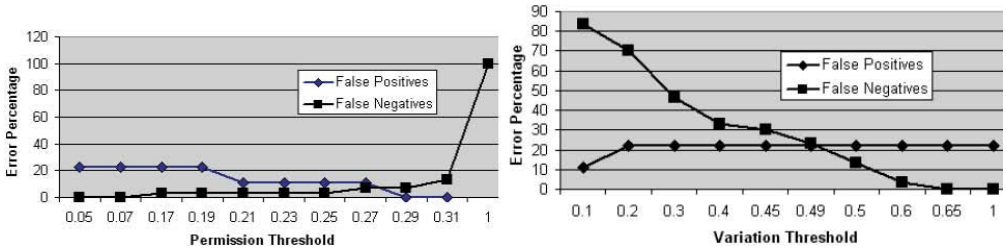
S_2, i.e., $MIN(Sim(S_1, S_2), Sim(S_2, S_1))$, and the *variation* threshold value, which defines the *maximum dissimilarity* between S_1 and S_2, i.e., $|Sim(S_1, S_2) - Sim(S_2, S_1)|$. The threshold values, along with the similarity values among the attributes of S_1 and S_2, are used by the *EQ* function, which decides if S_1 (i.e., a category *C*) should be assigned to a new data source S_2, which in turn determines the *category recovery error rate CR* for S_2, using the list of attributes S_1 in *C*. The smaller of the *CR* and the fixed-point value of S_2 yields the error threshold value of S_2.

Using forty randomly chosen training Websites with the Internet traffic (8), weather (12), and stock exchange (10) application domains in mind, which were collected on April 8, 2006, along with other data sources in financial (3), network data loss (3), and chemical properties (4), we determined the permissible and variation threshold values of *EQ*. The last three application domains have attributes closely related to, but not the same as stock exchange, Internet traffic, and weather, respectively; and these were included in training our category assignment approach to demonstrate its accuracy in assigning categories that have closely related, but different, lists of attributes.

Suppose L_1 is a set of lists of attributes such that each list belongs to one of the randomly selected data application domains listed above. Further assume that L_2 is a set containing three predefined lists of attributes, one for each of the three application domains: weather, stock exchange, and Internet traffic. Each list in L_1 was compared with each list in L_2 to determine the permissible and variation threshold values that yield the least total number of false positives and false negatives in matching[3]. The false positives and false negatives were determined manually for each enumerated pair of lists of attributes, with one list from L_1 and another one from L_2, which is in turn automatically categorized as equal or different according to the *EQ* function. The total number of detected false positives and negatives according to various permissible (variation, respectively) threshold values are plotted in the graph as shown in Figure 3(a) (Figure 3(b), respectively), which indicates that the ideal permissible (variation, respectively) threshold value is 0.29 (0.65, respectively), when the least total number of false positives and negatives occur, instead of

[3]A *false positive* (*false negative*, respectively) occurs when two lists of attributes that are *different* but are termed as *equal* (are *equal* but are termed as *different*, respectively).

the "intersect" point, i.e., 0.28 (0.49, respectively), which has a greater number of false positives and negatives.



(a) Permissible threshold values (b) Variation threshold values

Figure 3: Determining the permissible and variation threshold values in the *EQ* function using (training) data in forty randomly chosen source data Websites

The list of predefined attributes for each of the three application domains, i.e., weather, stock exchange, and Internet traffic, was generated by including all the attributes that occur in more than 80% of Web data attributes belonged to the same application domain, as shown in Table 2. We verified the accuracy of the permissible and variation threshold values detected in Figures 3(a) and 3(b) using a new test set of forty randomly chosen data Websites and the three application domains that we have been considering in mind. Out of the forty randomly selected test Websites, 37 were correctly categorized with zero false positive and 3 false negatives, an accuracy rate of 92.5%, which justifies the accuracy of the *EQ* threshold values and the comprehensiveness of the lists of predefined attribute names for their corresponding application domains, which can easily be extended to other application domains.

4.2.2 Verification of the error threshold value generation method

To verify our error threshold value generation approach partially using the *EQ* function for Web data sources, we conducted experiments using (i) training data, which were used to determine the fixed point values of the three different application domains considered in Section 4.2.1, along with the new population application domain, which were collected on May 12, 2006, and (ii) the randomly chosen consecutive data segments from test data, which evaluated the accuracy of the fixed point values of the corresponding Web data

sources generated by the training data. We observed that the fixed points determined by using the training and test data set are close. The deviation, which shows the differences between the fixed point values computed by using the training and randomly chosen test data, calculated as |(FixedPointTestData - FixedPointTrainingData)| / FixedPointTestData, ranges from 0% to 8.94%.

Recall that the fixed point value of data source *S* achieves the balance between the data recovery accuracy and the amount of recovery information to be stored in the synopsis matrix of *S*. According to the low deviation, i.e., between 0% to 8.94%, with an average of 3.84%, we conclude that our fixed point value detection approach works adequately[4].

4.3 Experimental Results on Data Filtering Scheme Generation

We justify the accuracy of our less-informative attribute detection approach in Section 4.3.1 and assert the correctness of our data filtering scheme generation and reevaluation approach in Section 4.3.2.

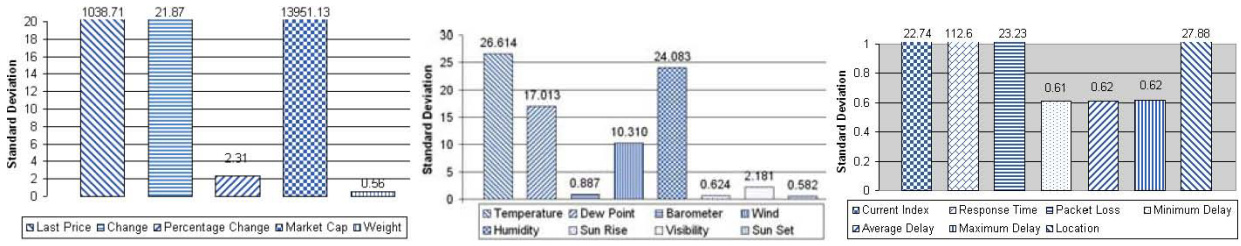
4.3.1 Verifying the Accuracy of Detecting Less-Informative Attributes

To verify the accuracy of our approach to determining less-informative attributes in each data segment, i.e., the ranking of attributes in each data segment, of a Web data source, which determines the attributes to be filtered, we performed experiments on the test data of various data sources using *SD*s (since the detection of less-informative attributes is always computed in real-time using real source data) and verified that detected less-informative attributes are indeed less varying than other attributes in a data source.

Figures 4(a), 4(b), and 4(c) show the *SD* (rankings) for different attributes of stock exchange, weather, and Internet traffic test data sources, respectively, except the *SID* attributes, i.e., "Company Name," "Location," and "Router Name" respectively, which cannot be filtered for data recovery purpose. The attributes with the highest ranking (i.e., the lowest *SD*) are "Weight," "Sun Set," and "Minimum Delay," respectively, which are

[4] Details of this experiment are not included in this paper due to the page-limit constraint; however, the details can be found in Section 4.2.1 of [AHU06].

exactly the less-informative attributes of the corresponding data sources, i.e., matching the ones examined manually. Such high accuracy is achieved because *SD* is mathematically sound and is a widely accepted concept in statistics for detecting the variations in the values of a data set.



(a) Stock (b) Weather (c) Internet Traffic

Figure 4: Experimental results generated by using various Web data source test data, stock exchange (http://finance.indiamart.com/markets/bse/), weather (http://weather.yahoo.com/), and Internet traffic (http://www.Internettrafficreport.com) downloaded on March 1, 2006 for detecting the less-informative attribute(s)

4.3.2 Verification of the Correctness of the Data Filtering Scheme Generation and Reevaluation Approach

Our data filtering approach uses an adaptive data filtering scheme (which is reevaluated at various time intervals) that defines the less-informative attribute(s) to be filtered, and the data filtering scheme is generated in real-time on real source data, and not on training data. The verification of the data filtering scheme involves verifying the correctness of the reevaluation of the data filtering scheme in between the various time intervals. We manually determined each less-informative attribute *A* for each randomly chosen data segment of a Web data source (100 data segments for each application domain chosen from the data shown in Table 2) and compare *A* to the automatically detected less-informative attributes of various data segments of the same Web data source generated by our data filtering approach. Each match is called a *hit*, whereas each mismatch is called a *miss*. We observe that the misses occur when the ranking of the attributes in a data source changes in between two consecutive reevaluations of the data filtering scheme, which is not reflected

in (i.e., integrated into) the currently adopted data filtering scheme after the change has occurred and before the data filtering scheme is reevaluated. This scenario occurs when the time interval between two subsequent reevaluations of the data filtering scheme is sometimes *larger* than the time interval between changes in the ranking of the attributes of a data source. These misses could be minimized by decreasing the time interval between two subsequent reevaluations of the data filtering scheme, which would in turn increase the workload on the system. The decrease in the time interval can be achieved by replacing '2' in Step 2(i) of Algorithm 1 with a value less than 2.

According to Table 4, the data filtering scheme has an average accuracy, or average number of *hits*, of 94.3%, whereas the average number of *misses* is 5.7%, a high accuracy.

We have also conducted experiments to verify the ability of our data filtering approach in determining the number of attributes (and their corresponding data items) to be filtered at each of the different data (transmission) rates, i.e., 60, 90, and 120 Kbps, with three different channel capacities, i.e., 72, 98, and 132 Kbps. According to Table 5, the number of attributes to be filtered decreases as the channel capacity increases, and vice versa, while the data rate remains constant. Also, we notice that no attributes are dropped when the data rate is lower than the channel capacity, as expected.

Data Source (Sets S_1, S_2, S_3)	**No. of Hits**			**No. of Misses**		
	S_1	S_2	S_3	S_1	S_2	S_3
Weather Information						
http://weather.yahoo.com/	94	94	93	6	6	7
http://www.wunderground.com/	95	94	95	5	6	5
http://www.weather.com	96	96	96	4	4	4
Stock Exchange Information						
http://quotes.nasdaq.com/quote.dll?page=nasdaq100	94	95	95	6	5	5
http://finance.indiamart.com/markets/bse/	95	95	95	5	5	5
http://www.channelnewsasia.com/cna/¯nance/sg/ stockmonitor.htm	93	94	93	7	6	7
Internet Traffic Information						
http://www.Internettrafficreport.com	94	95	95	6	5	5
http://average.miq.net/index.html	92	93	93	8	7	7
http://watt.nlanr.net/active/maps/ampmap	95	94	94	5	6	6
Average %	**94.3**			**5.7**		

Table 4: Experimental results of testing our data filtering scheme generation and reevaluation approach with an average number of *hits* of *94.3*%and *misses* of *5.7*%

Data Source	D	C	N	C	N	C	N
Weather Information							
http://weather.yahoo.com/ - Set 1	60	72	0	98	0	132	0
Set 2	90	…	2	…	0	…	…
Set 3	120	…	4	…	2	…	…
http://www.wunderground.com/ - Set 1	60	…	0	…	0	…	…
Set 2	90	…	2	…	0	…	…
Set 3	120	…	3	…	2	…	…
http://www.weather.com – Set 1	60	…	0	…	0	…	…
Set 2	90	…	2	…	0	…	…
Set 3	120	…	3	…	2	…	…
Stock Exchange Information							
http://quotes.nasdaq.com/ quote.dll?page=nasdaq100 - Set 1	60	…	0	…	0	…	…
Set 2	90	…	2	…	0	…	…
Set 3	120	…	3	…	2	…	…
http://finance.indiamart.com/ markets/bse/ - Set 1	60	…	0	…	0	…	…
Set 2	90	…	2	…	0	…	…
Set 3	120	…	3	…	2	…	…
http://www.channelnewsasia.com/cna/ Finance/sg/stockmonitor.htm - Set 1	60	…	0	…	0	…	…
Set 2	90	…	1	…	0	…	…
Set 3	120	…	2	…	1	…	…
Internet Traffic Information							
http://www.Internettraffi creport.com - Set 1	60	…	0	…	0	…	…
Set 2	90	…	2	…	0	…	…
Set 3	120	…	3	…	2	…	…
http://average.miq.net/index.html - Set 1	60	…	0	…	0	…	…
Set 2	90	…	2	…	0	…	…
Set 3	120	…	3	…	2	…	…
http://watt.nlanr.net/active/maps/ ampmap_active.php - Set 1	60	…	0	…	0	…	…
Set 2	90	…	2	…	0	…	…
Set 3	120	…	3	…	2	…	…

(D)ata transmission Rate (Kbps); (C)hannel Capacity (Kbps); (N)umber of Attributes to be filtered

Table 5: Experimental results for showing the number of attributes to be filtered with changes in *data* (*transmission*) *rate* and *channel capacity*

4.4 Experimental Results on Recovering Filtered Data

We have verified the correctness of our data recovery approach by (i) filtering data from test data (since data recovery is carried out in real-time on source data), (ii) recovering the filtered data values, and (iii) graphically comparing the recovered data values to the original data values. The results of the experiments on data recovery on three different

application domains using 2 GB, 3 GB, and 2 GB of data, respectively, which are portions of the data in Table 2, are shown in Table 6, which lists attribute values that were not recovered from the synopsis matrix of the three Web data sources. The data filtering schemes for the three data sources are {Minimum Delay}, {Percentage Change}, and {Sun Set}, respectively. Table 7, which includes error threshold values of various application domains that are different from the ones shown in Example 3, summarizes the experiments conducted on our data recovery approach.

Recall that when there is an attempt to recover a filtered value that was stored in the corresponding synopsis recovery matrix, the recovered value has no error; otherwise, an error occurs, which has the error rate value less than the error threshold value, since if this is not the case, then the filtered data value would have been stored in the synopsis recovery matrix. Hence, our data recovery approach would have low error value in the recovered data if the corresponding error threshold value were low. We claim that our filtered data recovery method achieves high accuracy in recovering filtered data, i.e., low error percentage in the recovered data, which is in the range of 0.37% and 9.83% with an average of 4.36%[5] (as shown in Table 7), at low information storage cost, i.e., percentage of the filtered data to be stored in the synopsis matrix, with an average of 2.2×10^{-4}%, as shown in Table 7. The processing times and memory usage for the synopsis matrix have been found to be insignificant.

[5] [5]Note that the error in recovered data is the complement of recovery accuracy.

Internet Traffic		Stock Information		Weather Information	
Original Values	**Recovered Values**	**Original Values**	**Recovered Values**	**Original Values**	**Recovered Values**
3.979	4.761	0.3	0.297	7.06	6.991
2.666	2.382	0.71	0.716	6.79	7.004
2.659	3.148	-0.04	-0.04	7.06	7.243
2.648	3	0.02	0.02	7.32	7.539
2.663	2.97	0.14	0.141	6.22	6.398
2.655	3.123	0.04	0.04	7.22	7.187
2.650	2.757	-0.1	-0.101	6.49	6.595
2.663	2.871	0.45	0.452	6.41	6.373
2.648	3.002	0.37	0.372	7.16	7.435
2.662	2.9	0.18	0.181	6.22	6.476
2.649	2.386	1.19	1.202	6.03	5.927
2.641	2.402	-0.06	-0.061	6.24	6.366

Table 6: Filtered data values that were not stored in the synopsis matrix for an Internet traffic (http://www.Internettrafficreport.com), a stock exchange (http://quotes.nasdaq.com/quote.dll?page=nasdaq100), and a weather (http://weather.yahoo.com/) data sources

Data Stream Source	**Data Size (GB)**	**Error Threshold Value**	**Data recovery Accuracy (%)**	**Error %**	**% of Shed data Stored**
http://www.internettrafficreport.com	2	20	90.17	9.83	1.71×10^{-4}
http://quotes.nasdaq.com/quote.dll?page=nasdaq100	3	1.3	99.63	0.37	1.73×10^{-4}
http://weather.yahoo.com/	2	4.5	97.12	2.88	3.2×10^{-4}
Average	**2.3**	**8.6**	**95.64**	**4.36**	**2.2×10^{-4}**

Table 7: Summary of the experiments conducted on the filtered data recovery approach

4.5 Experimental Comparisons of Our Data Filtering and Recovery Approach

In this section, we provide experimental comparisons of our data filtering approach with some of the closely related works. As discussed in Section 2, [BWL06] accomplishes redundant data removal by retaining only the first (or the earliest) record within a sliding window and eliminating all new records, which are treated as duplicates.

We compared the method proposed in [BWL06] with our data filtering approach using the Internet traffic test data, which come with eight attributes. The amount of data to be filtered was set at 25%, and thus the 'Minimum Delay' and 'Average Delay', which are the two least informative attributes out of the eight, are filtered out using our data filtering approach. For each original attribute value that was filtered, we computed an *estimated*

value used for measuring the data loss. (Estimated value is the most recent non-filtered value of the affected attribute with the same SID value.) Table 8 presents the data loss for some of the filtered values, along with the cumulative data loss, which is 0.3% using our data filtering approach. Furthermore, we applied the redundant data removal method proposed in [BWL06] on the same test data described above. According to [BWL06], in order to achieve a 25% data filtering, we should retain the first three records, i.e., tuples, and filter every fourth tuple within a data segment. Since complete tuples were filtered out, we measured the data loss for each eliminated attribute. Table 8 shows the data losses for each attribute, six in total of the Internet traffic application domain, along with the cumulative data loss, which is 38.75%, which is much higher than the cumulative data loss using our data filtering approach.

The reason behind the difference in the data loss is that our data filtering approach filters less-informative attributes, instead of complete tuples, which may include informative, as well as, less-informative attributes, as filtered by [BWL06].

Our Data Filtering Approach								
Minimum Delay			**Average Delay**					
Original Value	**Estimated Value**	**Data Loss (%)**	**Original Value**	**Estimated Value**	**Data Loss (%)**			
3.97	3.98	0.35	3.97	3.98	0.34			
2.66	2.67	0.51	2.66	2.67	0.51			
2.65	2.66	0.47	2.65	2.66	0.47			
2.64	2.65	0.26	2.64	2.65	0.26			
2.67	2.67	0.06	2.66	2.66	0.06			
2.67	2.66	0.36	2.68	2.66	0.36			
: :	: :	: :	: :	: :	: :			
		0.30			**0.30**			
Cumulative Data Loss = 0.30%								
[BWL06] Data Filtering Approach								
Current Index			**Response Time**			**Packet Loss**		
Original Value	**Estimated Value**	**Data Loss (%)**	**Original Value**	**Estimated Value**	**Data Loss (%)**	**Original Value**	**Estimated Value**	**Data Loss (%)**
87.49	87.40	0.1	123.23	123.67	0.35	0.02	0.93	7311.39
88.53	88.56	0.05	110.59	110.21	0.35	0.66	0.78	17.33
88.88	88.15	0.83	111.87	111.96	0.08	0.90	0.46	48.78
0.65	0.07	89.04	0.84	0.6	27.94	100.9	100.95	0.05
87.59	87.44	0.17	125.31	125.85	0.43	0.52	0.87	66.13
88.77	88.11	0.75	111.71	111.32	0.36	0.82	0.94	14.08

:	:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:	:
		7.74			**17.14**			**206.68**
Minimum Delay			**Average Delay**			**Maximum Delay**		
Original Value	**Estimated Value**	**Data Loss (%)**	**Original Value**	**Estimated Value**	**Data Loss (%)**	**Original Value**	**Estimated Value**	**Data Loss (%)**
2.64	2.67	0.92	2.64	2.67	0.91	2.65	2.68	0.92
2.66	2.66	0.11	2.66	2.66	0.11	2.67	2.67	0.11
2.67	2.66	0.44	2.67	2.66	0.44	2.68	2.66	0.49
5.31	5.31	0.03	5.31	5.31	0.03	5.33	5.33	0.03
2.66	2.66	0.34	2.66	2.66	0.34	2.67	2.68	0.34
2.67	2.67	0.11	2.66	2.67	0.11	2.67	2.68	0.12
:	:	:	:	:	:	:	:	:
		0.31			**0.31**			**0.31**
Cumulative Data Loss = 38.75%								

Table 8: Comparisons of our data filtering approach and the data filtering approach presented in [BWL06] using Set 1 of the Internet traffic test data in Table 2.

In [BDM04], the authors propose the usage of aggregate values to compensate for data lost caused by load shedding on stream data. We compared the method proposed in [BDM04] with our data recovery scheme using the Internet traffic and the Yahoo Weather information test data in Table 2 for recovering the 'Minimum Delay' and 'Sun Set' attribute values, respectively. For each original attribute value that is recovered, we compute the error in the recovered value using (i) our data recovery scheme and (ii) the data recovery approach described in [BDM04]. Table 9 shows the error rates in the recovered data, which is 4.54% using our approach, and 7.55% using the approach in [BDM04]. The error rate in recovered attribute values using our approach is computed as $(V - R_1)/V$, and the error rate using the approach described in [BDM04] is computed as $(V - R_2)/V$, where V is the original attribute value before filtering, R_1 is the recovered attribute value using our filtered data recovery approach, and R_2 is the recovered attribute value using the approach described in [BDM04].

The reason behind the difference in the recovered data accuracy is that our data recovery approach uses the intelligent synopsis based approach, which stores the actual or nearly actual filtered value, whereas the approach employed in [BDM04] uses an aggregate value to compensate for data lost by filtering.

Web data source	Error percentage using our data recovery approach	Error percentage using the data recovery approach in [BDM04]
http://www.internettrafficreport.com	6.83	8.5
http://weather.yahoo.com/	2.25	6.6
Average	**4.54**	**7.55**

Table 9: Comparisons of our data recovery approach and the data recovery approach presented in [BDM06] using Internet traffic Set 1 and weather Set 1 test data as shown in Table 2.

5 Conclusions

In this paper we have proposed a dynamic data filtering and recovery approach for solving the excessive amount of data transmitted on the Internet. Our data filtering and recovery approach (i) detects and filters less-informative attribute(s) from a source Website, which reduces the information loss by retaining more-informative data from the source Website, and (ii) includes a unique data recovery method with low storage overhead and high accuracy in recovering filtered Web data.

We have conducted experiments to verify (i) the correctness of our less-informative attribute data filtering approach, with a 100% accuracy in choosing the less-informative attributes of a source Website to be filtered, (ii) the correctness of our data filtering scheme generation and reevaluation, with a 94.3% accuracy rate in generating and reevaluating a data filtering scheme, and (iii) the accuracy of our data recovery approach, with 90.2%, 99.6%, and 97.1% success rates in recovering data in Internet traffic, stock exchange, and weather Web data, respectively, with an average data recovery accuracy of 95.6%.

Our data filtering approach (i) is dynamic in nature, since it is reevaluated in real-time, and (ii) is applicable to any kind of Web data with a static schema. In addition, the criticality of predefined category of data in a source Website S is computed automatically in defining the error threshold value of S, which facilitates automatic detection of the error threshold value of S. Furthermore, our data recovery approach is also adaptive, since it stores filtered data in a synopsis matrix of S whenever the change in the value to be filtered and the previous value stored in the synopsis matrix is greater than the error threshold value.

Although our data filtering and recovery approach is capable of handling numerical attribute data while detecting the less-informative attributes with high accuracy, the *SD* based approach for determining less-informative attributes and *SID*s can be further enhanced so that it can handle non-numerical data by converting them into their ASCII equivalences. In addition, if the predefined lists of attributes for the various criticality categories used to detect the criticality category to which the Web data belongs is incomplete, this can affect our ability to detect the category recovery error rate accurately. Use of adaptive predefined lists should solve this problem, where the predefined lists are updated every time Web data are detected to belong to their category, by including similar or new attributes contained in the Web data.

Acknowledgements

We are thankful to the referees for their constructive comments, based on which revisions were made to further enhance the content, presentation, and quality of the paper.

References

[ACG02] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating Fuzzy Duplicates in Data Warehouses. In *Proceedings of the 28th International Conference on Very Large Databases*, pages 586–597, 2002.

[AD91] H. Almuallim and T. Dietterich. Learning with Many Irrelevant Features. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 547–552, 1991.

[AHU06] A. Ahuja. A Dynamic Attribute-Based Load-Shedding and Data Recovery Scheme for Data Stream Management Systems. Masters Thesis, Computer Science Dept., Brigham Young University, 2006.

[BDM04] B. Babcock, M. Datar, and R. Motwani. Load Shedding for Aggregation Queries over Data Streams. In *Proceedings of the 20th International Conference on Data Engineering (ICDE'04)*, pages 350-361, 2004.

[BG04] I. Bhattacharya and L. Getoor. Iterative Record Linkage for Cleaning and

Integration. In *Proceedings of the 9th ACM SIGMOD workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 11–18, 2004.

[BK05] C. Bouras and A. Konidaris. Estimating and Eliminating Redundant Data Transfers over the Web: A Fragment Based Approach. *International Journal of Communication Systems*, 18(2):119–142, 2005.

[BM03] M. Bilenko and R.J. Mooney. Adaptive Duplicate Detection Using Learnable String Similarity Measures. In *Proceedings of the International Conference on Knowledge Discover and Data Mining (KDD)*, pages 39–48, 2003.

[BWL06] Y. Bai, F. Wang, and P. Liu. Efficiently Filtering RFID Data Streams. In *Proceedings of the First International VLDB Workshop on Clean Databases (CleanDB'06)*, 2006.

[CGM05] S. Chaudhuri, V. Ganti, and R. Motwani. Robust Identification of Fuzzy Duplicates. In *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, pages 865–876, 2005.

[CK07] V. Chandola and V. Kumar. Summarization – Compressing Data into an Informative Representation. *Knowledge and Information Systems:* 12(3), 355-378, August 2007.

[FRPG01] A. Floratos, I. Rigoutsos, L. Parida, and Y. Gao. DELPHI: A Pattern-Based Method for Detecting Sequence Similarity. *IBM Journal of Research and Development*, 45(3/4):455–474, May/July 2001.

[GN06] I. Garcia and Y.-K. Ng. Eliminating Redundant and Less-Informative RSS News Articles Based on Word Similarity and a Fuzzy Equivalence Relation. In *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence* (*ICTAI*-2006), pages 465–473, November 2006.

[Ham50] R.W. Hamming. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 29:147–160, 1950.

[HS95] M.A. Hernandez and S.J. Stolfo. The Merge/Purge Problem for Large Databases. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 127–138, 1995.

[KR92] K. Kira and L.A. Rendell. A Practical Approach to Feature Selection. In *Proceedings of the Ninth International Workshop on Machine Learning* (*ML*92), pages 249–256, 1992.

[ME97] A.E. Monge and C. Elkan. An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records. In *Proceedings of the SIGMOD 1997 Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 23–29, 1997.

[PPH05] P. Pantel, A. Philpot, and E. Hovy. An Information Theoretic Model for Database Alignment. In *Proceedings of the 17th International Conference on Scientific and Statistical Database Management*, pages 14–23, 2005.

[RSM94] R. Richardson, A.F. Smeaton, and J. Murphy. Using WordNet as a Knowledge Base for Measuring Semantic Similarity between Words. In *Proceedings of AICS conference*, pages 179–192, 1994.

[SB02] S. Sarawagi and A. Bhamidipaty. Interactive Deduplication Using Active Learning. In *Proceedings of the 8th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 269–278, 2002.

[Sha49] C.E. Shannon. *The Mathematical Theory of Information*. University of Illinois Press, 1949. (Reprinted 1998).

[SSS00] M.R. Spiegel, J.J. Schiller, and R.A. Srinivasan. *Schaum's Outline of Probability and Statistics*. McGraw-Hill, 2000.

[THA03] M. Tobita, K. Horiuchi, and K. Araki. BirdsAnts: Bringing Informative Rules from a Database System, Aimed at Novel Targets Search. In *Genome Informatics 14*, pages 286–287, 2003.

[TL03] T.-H. Tsai and S.-Y. Lee. Simsearcher: A Local Similarity Search Engine for Biological Sequence Databases. In *Proceedings of the International Symposium on Multimedia Software Engineering (ISMSE'03)*, pages 305–312, 2003.

[VKH07] J.D. Van Hulse, T.M. Khoshgoftaar, and H. Huang. The Pairwise Attribute Noise Detection Algorithm. *Knowledge and Information Systems*: 11(2), 171-190, February 2007.

[WVMA07]L. Wei, E. Keogh, H. Van Herle, A. Mafra-Neto, and R.J. Abbott. Efficient Query Filtering for Streaming Time Series with Applications to Semisupervised Learning of Time Series Classifiers. *Knowledge and Information Systems*: 11(3), 313-344, April 2007.