Under consideration for publication in Knowledge and Information Systems

On Clustering Massive Text and Categorical Data Streams

Charu C. Aggarwal[1], Philip S. Yu[2]

[1]IBM T. J. Watson Research Center, Hawthorne, NY 10532, USA;
[2]University of Illinois at Chicago, Chicago, IL, USA

Abstract. In this paper, we will study the data stream clustering problem in the context of text and categorical data domains. While the clustering problem has been studied recently for *numeric* data streams, the problems of text and categorical data present different challenges because of the large and un-ordered nature of the corresponding attributes. Therefore, we will propose algorithms for text and categorical data stream clustering. We will propose a condensation based approach for stream clustering which summarizes the stream into a number of fine grained cluster droplets. These summarized droplets can be used in conjunction with a variety of user queries to construct the clusters for different input parameters. Thus, this provides an online analytical processing approach to stream clustering. We also study the problem of detecting noisy and outlier records in real time. We will test the approach for a number of real and synthetic data sets, and show the effectiveness of the method over the baseline OSKM algorithm for stream clustering.

Keywords: Stream clustering, Text clustering, text streams, text stream clustering, categorical data

1. Introduction

In this paper, we will study the problem of clustering text and categorical data streams. This problem is relevant in a number of web related applications such as news group segmentation, text crawling, and target marketing for electronic commerce. Some applications of text and categorical data stream clustering are as follows:

Received September 08, 2009
Revised May 31, 2009
Accepted June 20, 2009

– Many portals on the world wide web provide real time news and other articles which require quick summarization and filtering. Such methods often require effective and efficient methods for text segmentation.
– Many web crawlers continuously harvest thousands of web pages on the web, which is subsequently summarized by human effort. When the volume of such crawls is significant, it is not realistically possible to achieve this goal by human effort. In such applications, data stream clustering algorithms can be helpful in organizing the crawled resources into coherent sets of clusters.
– In many electronic commerce applications, large volumes of transactions are processed on the world wide web. Such transactions can take the form of categorical or market basket records. In such cases, it is often useful to perform real time clustering for target marketing.

The data stream problem has received increasing attention in recent years because of technological innovations which have facilitated the creation and maintenance of such data. A number of data mining problems recently have been studied in the context of data streams (Aggarwal, 2003; Babcock et al, 2002; Domingos et al, 2000; O'Callaghan et al, 2002). The clustering problem is formally defined as follows: for a given set of data points, we wish to partition them into one or more groups of similar objects. The similarity among the objects is typically defined with the use of some distance measure or objective function. In addition, data points which do not naturally fit into any particular cluster are referred to as *outliers*. Because of the applicability of the clustering and outlier detection problems, these problems have been well researched in the database and data mining communities (Bradley et al, 1998; Knorr et al, 1998; Ng et al, 1994; Rastogi et al, 2000; Zhang et al, 1996). The problems of text and categorical data clustering have also been recently studied (Cutting et al, 1992; Gibson et al, 1998; Guha et al, 1997; Silverstein et al, 1997) in the offline context. These methods cannot be easily extended to the data stream problem because of their high computational complexity. The data stream problem poses special challenges because of the large volume of incoming data. As a result, one cannot scan a data point more than once during the course of the computation. Furthermore, it is extremely important for the process to be computationally efficient, since the processing rate needs to be sufficiently fast to match the incoming rate of the data points.

The problem of stream clustering has been studied extensively for the case of continuous data (Aggarwal et al, 2003b; Aggarwal et al, 2008; Cao et al, 2006; Chen et al, 2007). However, such approaches are difficult to use directly for other data domains, since the cluster computation and tracking techniques may be completely different. A number of other techniques for the Topic Detection and Tracking (TDT) program have studied incremental clustering of text streams (Allan et al, 1998a; Allan et al, 1998b; Franz et al, 2001; Yang et al, 1998) to find new events and to track later articles related to new events. A recent piece of work in (Yang et al, 1998) is able to perform topic-conditioned novelty detection, though this approach requires supervision with class labels. Some recent techniques for streaming text clustering (Banerjee et al, 2004; He et al, 2007; Surendran et al, 2006; li et al, 2006; Zhang et al, 2005; Zhong, 2005) adapt a partition based approach to perform rapid online clustering of streaming data. While these techniques are effective for online clustering, they lack the flexibility to allow the user the ability to perform effective analysis of the clusters over different time horizons. A recent piece of work (Banerjee et al, 2007) studies the problem of online topic learning in text streams with the use of a

variety of similarity measures. The machine learning community has also studied the problem of incremental clustering for non-text data using methods such as COBWEB and CLASSIT (LFisher, 1987; Gennari et al, 1989). Our goal in this paper is to study the stream clustering problem for evolving data sets, so that a user has the maximum flexibility of examining the behavior of the clusters over different time horizons. In typical applications, users may wish to examine the clusters only over a particular region of the stream, or may wish to vary the parameters such as the number of clusters in the data. In particular, the ability to provide flexibility to the user in specifying different parameters (such as time horizon) is very challenging for a streaming application. This is because such parameters affect the clustering process from scratch, and it is difficult to maintain the results of clustering over multiple choices of parameters such as the user-specified horizon or the number of clusters. In order to achieve this goal, we will utilize a technique which leverages on intermediate summary statistics for the clustering task.

Most real applications present data streams which are highly evolving in nature. For example, an electronic commerce application containing customer transactions of market basket data is likely to create data which is highly skewed from the temporal perspective. Similarly, many newsfeed services receive large volumes of documents for text clustering and categorization. Thus, real applications often exhibit *temporal locality* which is not taken into account by most batch processing algorithms. The problems of clustering and outlier detection present a number of unique challenges in an evolving data stream environment. For example, the continuous evolution of clusters makes it essential to be able to quickly identify new clusters in the data. While the clustering process needs to be executed continuously in online fashion, it is also important to be able to provide end users with the ability to analyze the clusters in an offline fashion. In many cases, the users may wish to cluster the data based on specific temporal parameters such as the choice of a particular time horizon or specified number of clusters.

In order to achieve this goal, we will construct a framework in which carefully chosen statistical summary data is stored at regular intervals. We will introduce the concept of *cluster droplets* which provides an effective representation of the condensed characterization of the data. We will see that the summary data has specific characteristics such as *additivity* and *subtractivity* which are useful in computing the effectiveness over user-specified time horizons. The cluster droplets can be considered *intermediate summary data* which can be reused for a variety of purposes. Thus, this is an online analytical processing method in which efficient construction of the intermediate structures is essential for the success of the approach. While the data stream may have a very large volume, the condensed droplets are only a summary representation, and can be processed efficiently because of their compressed format. We will refer to our algorithm as ConStream which corresponds to *CONdensation based STREAM Clustering*. A related problem to clustering is that of *outlier detection*. We will also examine the outlier detection problem in the context of data stream clustering. Previous work has studied anomalies only in the context of either supervised models (Agrawal, 2007) or non-stream cases (Peterson et al, 2008). In the case of stream clustering, new patterns in the data often appear as outliers, which eventually become clusters in such cases. We will show how such outlier patterns can be extracted from the data stream in real time.

We note that the nature of the underlying data can affect the procedures for clustering data streams. For example, the sparsity and distribution of the different values of the categorical data can affect the methodology which is utilized for the clustering process. For example, numerical data allows for easy tracking of the underlying information in the form of quantitative moment information. This is not the case for text or categorical data in which the such moment information cannot be easily defined. Therefore, a different methodology is required for the case of categorical data streams. In this paper, we will discuss the problem of clustering different kinds of categorical data sets, including the closely related text domain.

In the presence of data evolution or temporal locality, the exploration of the stream over different time horizons may lead to different kinds of clusters. These different kinds of clusters may have applicability in understanding the behavior of the stream over different periods in time. For this purpose, one may desire to periodically store a certain amount of summary information which can be utilized later for better understanding and analysis of the data stream. This kind of methodology is similar to online analytical processing algorithms in which summary information is created for the purpose of repeated querying. In the context of a data stream, such a methodology seems quite convenient, since a fast data stream cannot be repeatedly processed in order to answer different kinds of queries.

The clustering algorithm of this paper pre-stores condensed statistical information at regular intervals. This condensed statistical data satisfies two requirements:

- It should be easy to update the statistical information for a fast data stream. In this paper, we specifically choose the nature of the statistical information in such a way that it is possible to perform updates which are linear in the number of data points.
- The statistical information stored should allow the computation of various analytical measures required by the user. Such measures could include clusters or outliers over a specific time horizon. It is also often desirable to find the nature of the evolution over a given time horizon.

In the next sections, we will discuss methods to store statistical information which satisfy both the conditions.

This paper is organized as follows. In the next section, we will discuss the classification of different kinds of clusters and outliers. In section 3, we will discuss the process of storing and maintaining the data structures necessary for the clustering algorithm. We will also discuss the differences which arise from using different kinds of data. In section 4, we will describe the process of utilizing different kinds of offline processing methods on the underlying summary data. Section 5 describes the empirical results. The conclusions and summary are discussed in section 6.

2. The Nature of Clusters and Outliers in Stream Processing

Data streams exhibit considerable variations in the underlying patterns over the course of their progression. Old clusters may become inactive, and eventually

get replaced by new clusters. Similarly, when newly arriving data points do not naturally fit in any particular cluster, these need to be initially classified as outliers. However, as time progresses, these new points may create a distinctive pattern of activity which can be recognized as a new cluster. The temporal locality of the data stream is manifested by these new clusters. For example, the first web page belonging to a particular category in a news stream of current events may be recognized as an outlier, but may later form a cluster of documents of its own. On the other hand, the new outliers may not necessarily result in the formation of new clusters. Such outliers are true short-term abnormalities in the data since they do not result in the emergence of sustainable patterns.

In order to distinguish between the different kinds of clusters and abnormalities, we will introduce some notations and terminology. When a cluster is newly discovered during the arrival of the data stream, it is referred to as a *trend-setter*. From the point of view of the user, a trend-setter is an outlier, until the arrival of other points certify the fact that it is actually a cluster. If and when a sufficient number of new points have arrived in the cluster, it is referred to as a *mature* cluster. We will quantify these definitions more precisely slightly later. At a given moment in time, a mature cluster can either be *active* or *inactive*. A mature cluster is said to be *active* when it has continued to receive data points in the recent past. When a mature cluster is not active, it is said to be *inactive*. In some cases, a trend-setting cluster becomes inactive before it has had a chance to mature. Such a cluster typically contains a small number of transient data points. In a given application, this may typically be the result of a short-term abnormality.

Since the stream clustering process should provide a greater level of importance to recent clusters, we will provide a time-sensitive weightage to each data point. It is assumed that each data point has a time-dependent weight defined by the function $f(t)$. The function $f(t)$ is also referred to as the *fading function*. The fading function $f(t)$ is a non-monotonic decreasing function which decays uniformly with time t. In order to formalize this concept, we will define the *half-life* of a point in the data stream.

Definition 1. The half life t_0 of a point is defined as the time at which $f(t_0) = (1/2)f(0)$.

Conceptually, the aim of defining a half life is to quantify the rate of decay of the importance of each data point in the stream clustering process. The *decay-rate* is defined as the inverse of the half life of the data stream. We denote the decay rate by $\lambda = 1/t_0$. We denote the weight function of each point in the data stream by $f(t) = 2^{-\lambda \cdot t}$. From the perspective of the clustering process, the weight of each data point is $f(t)$. It is easy to see that this decay function creates a half life of $1/\lambda$. It is also evident that by changing the value of λ, it is possible to change the rate at which the importance of the historical information in the data stream decays. The higher the value of λ, the lower the importance of the historical information compared to more recent data. For more stable data streams, it is desirable to pick a smaller value of λ, whereas for rapidly evolving data streams, it is desirable to pick a larger value of λ.

When a cluster is created during the streaming process by a newly arriving data point, it is allowed to remain as a trend-setting outlier for at least one half-life. During that period, if at least one more data point arrives, then the cluster becomes an active and mature cluster. On the other hand, if no new

points arrive during a half-life, then the trend-setting outlier is recognized as a true anomaly in the data stream. At this point, this anomaly is removed from the list of current clusters. We refer to the process of removal as *cluster death*. Thus, a new cluster containing one data point dies when the (weighted) number of points in the cluster is 0.5. The same criterion is used to define the death of mature clusters. A necessary condition for this criterion to be met is that the inactivity period in the cluster has exceeded the half life $1/\lambda$. The greater the number of points in the cluster, the greater the level by which the inactivity period would need to exceed its half life in order to meet the criterion. This is a natural solution, since it is intuitively desirable to have stronger requirements (a longer inactivity period) for the death of a cluster containing a larger number of points.

In the next section, we will discuss the process for cluster droplet maintenance. The algorithm dynamically maintains a set of clusters by using an algorithm which scales effectively with data size. In order to achieve better scalability of the maintenance process, it is necessary to construct data structures which allow for additive operations on the data points.

3. Storage and Update of Cluster Statistics

The data stream consists of a set of multi-dimensional records of dimensionality denoted by d. The format of the attribute values for each data point is defined based on the domain at hand. For the case of the categorical stream domain, each of these d dimensions corresponds to a categorical attribute value. It is assumed that the ith categorical dimension contains v_i possible values. For the case of text records, each dimension corresponds to the frequency of a word in the document. Thus, the dimensionality is defined by the size of the lexicon. Only words which are included in a document have non-zero frequency. The corresponding attribute value is the numeric frequency of that word in the vector space representation.

For the purpose of achieving greater accuracy in the clustering process, it is necessary to maintain a high level of granularity in the underlying data structures. In order to achieve this goal, we will use a process in which condensed clusters of data points are maintained. We will refer to such groups as *cluster droplets*. We will discuss and define the cluster droplet differently for the case of text and categorical data streams respectively. First, we will define the cluster droplet for the categorical data domain:

Definition 2. A cluster droplet $\mathcal{D}(t, \mathcal{C})$ for a set of categorical data points $\mathcal{C}$ at time t is referred to as a tuple $(\overline{DF2}, \overline{DF1}, n, w(t), l)$, in which each tuple component is defined as follows:

–The vector $\overline{DF2}$ contains $\sum_{i \in \{1 \ldots d\}, j \in \{1, \ldots d\}, i \neq j} v_i \cdot v_j$ entries. For each pair of dimensions, we maintain $v_i \cdot v_j$ values. We note that v_i is number of possible categorical values of dimension i and v_j is the number of possible values of dimension j. Thus, for each of the $v_i \cdot v_j$ categorical value pairs i and j, we maintain the (weighted) counts of the number of points for each value pair which are included in cluster $\mathcal{C}$. In other words, for every possible pair of categorical values of dimensions i and j, we maintain the weighted number of points in the cluster in which these values co-occur.

–The vector $\overline{DF1}$ contains $\sum_{i=1}^{d} v_i$ entries. For each i, we maintain a weighted

count of each of the v_i possible values of categorical attribute i occurring in the cluster.

–The entry n contains the number of data points in the cluster.

–The entry $w(t)$ contains the sum of the weights of the data points at time t. We note that the value $w(t)$ is a function of the time t and decays with time unless new data points are added to the droplet $\mathcal{D}(t)$.

–The entry l contains the time stamp of the last time that a data point was added to the cluster.

The reason for keeping the pairwise information on attribute values will become clear in the section on offline analysis, in which we exploit this information to perform analysis of inter-attribute correlations within clusters. We note that the above definition of a droplet assumes a data set in which each categorical attribute takes on a small number of possible values. (Thus, the value of v_i for each dimension i is relatively small.) However, in many cases, the data might actually be somewhat sparse. In such cases, the values of v_i could be relatively large. In those instances, we use a sparse representation. Specifically, for each pair of dimensions i and j, we maintain a list of the categorical value pairs which have *non-zero* counts. In a second list, we store the actual counts of these pairs. In many cases, this results in considerable savings of storage space. For example, consider the dimension pairs i and j, which contain v_i and v_j possible categorical values. Also, let us consider the case when $b_i \leq v_i$ and $b_j \leq v_j$ of them have non-zero presence in the droplet. Thus, at most $b_i \cdot b_j$ categorical attribute pairs will co-occur in the points in the cluster. We maintain a list of these (at most) $b_{ij} < b_i \cdot b_j$ value pairs along with the corresponding counts. This requires a storage of $3 \cdot b_{ij}$ values. (Two entries are required for the identities of the value pairs and one is required for the count.) We note that if the number of distinct non-zero values b_i and b_j are substantially lower than the number of possible non-zero values v_i and v_j respectively, then it may be more economical to store $3 \cdot b_{ij}$ values instead of $v_i \cdot v_j$ entries. These correspond to the list of categorical values which have non-zero presence together with the corresponding weighted counts. This works best for correlated data in which there are even fewer non-zero entries within a cluster droplet. Similarly, for the case of $\overline{DF1}$, we only need to maintain $2 \cdot b_i$ entries for each dimension i.

Next, we consider the case of the text data set which is an example of a *sparse numeric* data set. This is because most documents contain only a small fraction of the vocabulary with non-zero frequency. The only difference with the categorical data domain is the way in which the underlying cluster droplets are maintained.

Definition 3. A cluster droplet $\mathcal{D}(t, \mathcal{C})$ for a set of text data points $\mathcal{C}$ at time t is defined to as a tuple $(\overline{DF2}, \overline{DF1}, n, w(t), l)$. Each tuple component is defined as follows:

–The vector $\overline{DF2}$ contains at most $3 \cdot wb \cdot (wb-1)/2$ entries. Here wb is the number of distinct words with non-zero presence in the cluster $\mathcal{C}$. For each pair of dimensions, we maintain a list of the pairs of word ids with non-zero counts. We also maintain the sum of the weighted counts for such word pairs. In practice, the number is substantially lower than $3 \cdot wb \cdot (wb-1)/2$, since we only need to keep those word pairs which co-occur in at least one document.

–The vector $\overline{DF1}$ contains $2 \cdot wb$ entries. We maintain the identities of the words

with non-zero counts. In addition, we maintain the sum of the weighted counts for each word occurring in the cluster.

–The entry n contains the number of data points in the cluster.

–The entry $w(t)$ contains the sum of the weights of the data points at time t. We note that the value $w(t)$ is a function of the time t and decays with time unless new data points are added to the droplet $\mathcal{D}(t)$.

–The entry l contains the time stamp of the last time that a data point was added to the cluster.

We note that the definition of the cluster droplet for the case of categorical and text data sets are quite similar, except that the text data sets are sparse, and the categorical data sets are dense. The representation for each case needs to take this into account. In addition, the categorical data set contains unordered values for each attribute, which is not the case for the text data set.

The concept of cluster droplet has some interesting properties that will be useful during the maintenance process. These properties relate to the additivity and decay behavior of the cluster droplet.

Observation 1. Consider the cluster droplets $\mathcal{D}(t, \mathcal{C}_1) = (\overline{DF2_1}, \overline{DF1_1}, n_1, w(t)_1, l_1)$ and $\mathcal{D}(t, \mathcal{C}_2) = (\overline{DF2_2}, \overline{DF1_2}, n_2, w(t)_2, l_2)$. Then the cluster droplet $\mathcal{D}(t, \mathcal{C}_1 \cup \mathcal{C}_2)$ is defined by the tuple $(\overline{DF2_1} + \overline{DF2_2}, \overline{DF1_1} + \overline{DF1_2}, n_1 + n_2, w(t)_1 + w(t)_2, \max\{l_1, l_2\})$.

The cluster droplet for the union of two clusters is the sum of individual entries. The only exception is the last entry which is the maxima of the two last-update times. We note that the additivity property provides considerable convenience for data stream processing since the entries can be updated efficiently using simple additive and maximization operations.

The second observation relates to the rate of decay of the condensed droplets. Since the weights of each data point decay with the passage of time, the corresponding entries also decay at the same rate. Correspondingly, we make the following observation:

Observation 2. Consider the cluster droplet $\mathcal{D}(t, \mathcal{C}) = (\overline{DF2}, \overline{DF1}, n, w(t), l)$. Then the entries of the same cluster droplet $\mathcal{C}$ at a time $t' > t$ (without the addition of new data points) are given by $\mathcal{D}(t', \mathcal{C}) = (\overline{DF2} \cdot 2^{-\lambda \cdot (t'-t)}, \overline{DF1} \cdot 2^{-\lambda \cdot (t'-t)}, n, w(t) \cdot 2^{-\lambda \cdot (t'-t)}, l)$.

The above observation is important in regulating the rate at which the data points in the cluster decay over time. The combination of the two observations discussed above are essential in maintaining the clusters over time.

3.1. Cluster Droplet Maintenance

In this subsection, we will discuss the process of cluster droplet maintenance. The purpose of using cluster droplets is to create an effective intermediate representation from which a user may query using a variety of parameters. The maintenance algorithm continuously maintains the droplets $\mathcal{C}_1 \ldots \mathcal{C}_k$, which it updates as new data points arrive. For each cluster, the entire set of statistics in the droplet is maintained. The maximum number k of droplets maintained is dependent upon the amount of available main memory. Even if modest memory

Algorithm *MaintainStreamClusters*(Clusters: $\mathcal{C}_1 \ldots \mathcal{C}_k$,
Incoming Data Point: $\overline{X^q}$);
begin
{ Let the time at which $\overline{X^q}$ arrives be t^q;
for $i = 1$ to k $S(\overline{X^q}, \mathcal{C}_i) = ComputeSimilarity(\overline{X^q}, \mathcal{C}_i)$;
$mindex = \text{argmax}_{i \in \{1,\ldots k\}} S(\overline{X^q}, \mathcal{C}_i)$;
if $(S(\overline{X^q}, \mathcal{C}_{mindex}) > thresh)$ $AddPoints(\overline{X^q}, t^q, \mathcal{C}_{mindex})$;
else begin
Create new cluster with solitary point $\overline{X^q}$;
Create cluster droplet statistics with solitary point X^q;
Remove the least recently updated cluster to make room for new cluster;
end;
end;

Fig. 1. Maintenance of Cluster Droplets

Algorithm *AddPoints*(Data Point: X^q, Time stamp: t^q,
Cluster: $\mathcal{C}_j$);
begin
{ Decay factor needs to be updated since it was last
changed at the last update time of the cluster $\mathcal{C}_j$ }
Let l be the last update time of cluster $\mathcal{C}_j$;
Use Observation 2 to apply the temporal decay
factor $e^{-\lambda \cdot (t^q - l)}$ to update
statistics of cluster $\mathcal{C}_j$;
Use Observation 1 to add the data point $\overline{X^q}$ to $\mathcal{C}_j$ and update its statistics;
end

Fig. 2. Adding a new point to a cluster

resources are available, the statistics can be maintained at a relatively high level of granularity.

At the beginning of algorithmic execution, we start with an empty set of clusters. As new data points arrive, unit clusters containing individual data points are created. Once a maximum number k of such clusters have been created, we can begin the process of online cluster maintenance. Thus, we initially start off with a trivial set of k clusters. These clusters are updated over time with the arrival of new data points.

When a new data point $\overline{X}$ arrives, its similarity to each cluster droplet is computed. For the case of text data sets, the cosine similarity measure (Cutting et al, 1992; Silverstein et al, 1997) between $\overline{DF1}$ and $\overline{X}$ is used. Let $q_1 \ldots q_n$ be the frequencies of the words in $\overline{DF1}$, and $u_1 \ldots u_n$ be the frequencies of the words in $\overline{X}$. Then, the cosine measure between $\overline{DF1}$ and $\overline{X}$ is defined as follows:

$$cosine(\overline{DF1}, \overline{X}) = \frac{\sum_{r=1}^{n} q_r \cdot u_r}{\sqrt{\sum_{r=1}^{n} q_r^2} \sqrt{\sum_{r=1}^{n} u_r^2}} \quad (1)$$

While the cosine measure is used for the purpose of this paper, other coefficients such as dice and jaccard coefficients can also be supported. The following relationships may be used:

$$jaccard(\overline{DF1}, \overline{X}) = \frac{\sum_{r=1}^{n} q_r \cdot u_r}{\sum_{r=1}^{n} q_r^2 + \sum_{r=1}^{n} u_r^2 - \sum_{r=1}^{n} q_r \cdot u_r} \quad (2)$$

$$dice(\overline{DF1}, \overline{X}) = \frac{\sum_{r=1}^{n} 2 \cdot q_r \cdot u_r}{\sum_{r=1}^{n} q_r^2 + \sum_{r=1}^{n} u_r^2} \quad (3)$$

We note that all of these similarity functions simply use different combinations of the same basic computations (dot product and the L_2-modulus) as the cosine function, and can therefore be computed directly from the droplet statistics.

For the case of categorical data sets, the similarity measure is computed as follows: for each attribute i, we calculate the (weighted) fraction of records in the clusters which contain the same categorical value as the data point $\overline{X}$. We note that for the cluster $\mathcal{C}_j$, this percentage can be computed from the summary information contained in cluster droplet $\mathcal{D}(t, \mathcal{C}_j)$. This is because the vector $\overline{DF1}$ contains the weighted counts of each value for attribute i. The relevant weighted count is divided by the total weight $w(t)$ in order to calculate the corresponding fractional presence of a particular categorical value of attribute i. Let the weighted fraction for attribute i and cluster $\mathcal{C}_j$ be denoted by $wf_i(\overline{X}, \mathcal{C}_j)$. The average weighted fraction over all attributes for cluster $\mathcal{C}_j$ is given by the similarity value $S(\overline{X}, \mathcal{C}_j) = \sum_{i=1}^{d} wf_i(\overline{X}, \mathcal{C}_j)/d$. The overall process of stream cluster maintenance is illustrated in Figure 1.

As illustrated in Figure 1, the similarity value $S(\overline{X}, \mathcal{C}_j)$ is computed over all clusters. The cluster with the maximum value of $S(\overline{X}, \mathcal{C}_j)$ is chosen as the relevant cluster for data insertion. Let us assume that this cluster is $\mathcal{C}_{mindex}$. If the value of $S(\overline{X}, \mathcal{C}_{mindex})$ is larger than the threshold $thresh$, then the point $\overline{X}$ is assigned to the cluster $\mathcal{C}_{mindex}$. The process of adding a data point to a cluster is denoted by *AddPoints* and is illustrated in Figure 2. On the other hand, when the similarity value is less than the threshold $thresh$, a new cluster is created containing the solitary data point $\overline{X}$. This newly created cluster replaces the inactive cluster. This newly created cluster replaces the least recently updated cluster from the current set of clusters. We note that the newly cluster is a potential true outlier or the beginning of a new trend of data points. Further understanding of this new cluster may only be obtained with the progress of the data stream.

In the event that $\overline{X}$ is inserted into the cluster $\mathcal{C}_{mindex}$, we need to perform two steps:

- We update the statistics to reflect the decay of the data points at the current moment in time. This updating is performed using the computation discussed in Observation 2. Thus, the relevant updates are performed in a "lazy" fashion. In other words, the statistics for a cluster do not decay, until a new point is added to it. Let the corresponding time stamp of the moment of addition be t. The last update time l is available from the cluster droplet statistics. We multiply the entries in the vectors $\overline{DC2}$, $\overline{DC1}$ and $w(t)$ by the factor $2^{-\lambda \cdot (t-l)}$ in order to update the corresponding statistics. We note that the lazy update mechanism results in stale decay characteristics for most of the clusters. This does not however affect the afore-discussed computation of the similarity measures.
- In the second step, we add the statistics for each newly arriving data point to the statistics for $\mathcal{C}_{mindex}$ by using the computation discussed in Observation 2.

In the event that the newly arriving data point does not naturally fit in any of the cluster droplets and an inactive cluster does exist, then we replace

the most inactive cluster by a new cluster containing the solitary data point $\overline{X}$. In particular, the replaced cluster is the least recently updated cluster from the current set of clusters. We also associate an *id* with each cluster when it is created. This *id* is unique to each cluster and is helpful in book keeping while comparing the set of droplets at two different time periods. We will discuss more details on this issue in the next section.

We note that a similarity threshold *thresh* is used in order to determine when an incoming data point creates a new cluster. This threshold is *dynamically* determined by using the similarity computation of previous assignments. The mean μ_t and standard deviation σ_t of the minimum similarity values (or similarity value to assigned cluster) in the history of assignments are computed at the time of arrival of the tth data point. Note that this can be efficiently computed in the stream scenario by maintaining the first order and second order statistics (Zhang et al, 1996) of the history of computations. The threshold is set at $\mu + 3 \cdot \sigma$, which is consistent with a normal distribution assumption.

At a given moment in time, we maintain only the current snapshot of clusters in main memory. We also periodically store the statistical information about the clusters on disk. The main reason behind the storage of the snapshots of clusters at different moments in time is to be able to create and analyze the clusters over different time horizons. This can be done at regular time intervals, or it can be done by using a pyramidal time frame concept discussed in (Aggarwal, 2003). In the pyramidal time frame concept, the cluster droplets are stored at intervals which decrease exponentially with greater level of recency. Conceptually, the uniform time interval and the pyramidal time frame concepts provide no difference in terms of functionality. The only difference is in terms of a better level of approximation of a user specified time horizon using the pyramidal time frame. Therefore, we will refer the discussion of the pyramidal time frame to (Aggarwal, 2003), and proceed with a (simpler) description using regular intervals of storage.

4. Offline Analysis of Clusters

The methods discussed in the previous section are useful for online maintenance of the cluster droplets. In many cases, it may be valuable to provide the user with the ability to perform offline analysis of the cluster droplets. For example, even though the cluster droplets are condensed representations of the data with relatively high granularity, it may often be desirable to find higher level clusters which provide a broader overview.

4.1. Horizon Specific Analysis of Clusters

In many applications, it may be desirable to perform a horizon-specific analysis of the underlying clusters in order to obtain better insight. Some examples of queries which may be useful to the end-user are as follows:

- For a given time window (t_1, t_2), find the l higher level clusters in the data.
- For a given time window (t_1, t_2), find the new trend setting outliers.
- For a given time window (t_1, t_2), find all the newly created true outliers.
- For a given time window (t_1, t_2) find all clusters which became inactive.

– For a given time window (t_1, t_2) find the most correlated sets of categorical/text attribute pairs.

We note that the individual cluster droplets are constructed using the data received from the beginning of the stream arrival. Therefore, these droplets cannot be used directly in order to construct the clusters. We assume that the cluster droplets are stored at pre-decided intervals in the data stream. If the cluster droplets are stored at an interval with high granularity, then the cluster droplets at times t_1 and t_2 can be closely approximated by the droplets at the closest storage times $t_1' \approx= t_1$ and $t_2' \approx= t_2$. Let $\mathcal{R}(t_1)$ and $\mathcal{R}(t_2)$ be the set of (approximated) cluster droplets at times t_1 and t_2. Then, it is desirable to find a new set of cluster droplets which correspond to the activity occurring in the interval (t_1, t_2). In order to construct this new set of cluster droplets $\mathcal{M}(t_1, t_2)$, we use the following procedure:

– For all cluster droplets which are contained in $\mathcal{R}(t_2)$, we classify them into two types: (a) Droplets which were created after time t_1 and are therefore not present in $\mathcal{R}(t_1)$ (b) Droplets which were created before time t_1, and are therefore present in $\mathcal{R}(t_1)$. We note that the *ids* associated with each droplet can be utilized in order to perform the matching at the two times.
– Those droplets which are not present in $\mathcal{R}(t_1)$ are directly added to $\mathcal{M}(t_1, t_2)$. For each such droplet, we multiply each of the statistical values $\overline{DF2}$, $\overline{DF1}$, and $w(t)$ by $e^{-\lambda \cdot (t_2 - l_u)}$. The purpose of performing this additional multiplicative operation is to ensure that the decay characteristics of each droplet have been properly updated.
– For those droplets which are present in both $\mathcal{R}(t_1)$ and $\mathcal{R}(t_2)$, an additional droplet subtraction operation needs to be performed for each cluster in $\mathcal{R}(t_2)$. After performing the subtraction operation, each newly modified cluster is added to $\mathcal{M}(t_1, t_2)$. We will discuss this subtraction operation in some detail below.

We note that the time-discounting of the data points make the subtraction operation somewhat tricky. The time decay component of each cluster droplet contained in $\mathcal{R}(t_1)$ assumes a clock value of at most t_1. In practice, since the time decay process is applied in a lazy fashion, the individual clusters are decayed to an even lower extent than the time stamp t_1. In order to meaningfully compare $\mathcal{R}(t_1)$ and $\mathcal{R}(t_2)$, we need to apply the same decay function to each corresponding cluster droplet. Therefore, we need to modify the statistics of each cluster droplet in $\mathcal{R}(t_1)$ and $\mathcal{R}(t_2)$ so that the decay function is applied assuming a time stamp of t_2. We also note that the decay component of a cluster droplet is updated only when a data point is added to the cluster. Therefore, for a cluster with last update time l_u, we multiply each of the statistical values $\overline{DF2}$, $\overline{DF1}$, and $w(t)$ by $e^{-\lambda \cdot (t_2 - l_u)}$. This process is applied to each of the cluster droplets in $\mathcal{R}(t_1)$ and $\mathcal{R}(t_2)$. Next,we match pairs of cluster droplets by using their *ids*. Thus, a cluster droplet at time t_2 is an evolved version of its matched version at time t_1. Let $(\overline{DF2}_1, \overline{DF1}_1, n_1, w(t)_1, l_1)$ and $(\overline{DF2}_2, \overline{DF1}_2, n_2, w(t)_2, l_2)$ be two corresponding droplets from $\mathcal{R}(t_1)$ and $\mathcal{R}(t_2)$. Next, we perform a subtraction operation to form the droplet $(\overline{DF2}_2 - \overline{DF2}_1, \overline{DF1}_2 - \overline{DF1}_1, n_2 - n_1, w(t)_2 - w(t)_1, l_2)$. This modified droplet is added to $\mathcal{M}(t_1, t_2)$.

Once the set of droplets in $\mathcal{M}(t_1, t_2)$ have been constructed, we apply a higher level clustering operation on these objects. For this purpose, we treat

each droplet in $\mathcal{M}(t_1, t_2)$ as a pseudo-point. These pseudo-points are then re-clustered by using a K-means type algorithm. In this K-means type algorithm, we apply the following iterative process:

- We first sample K points from $\mathcal{M}(t_1, t_2)$.
- We assign each point in $\mathcal{M}(t_1, t_2)$ to one of these K points. This assignment is done using the same distance measure which was discussed in the section on cluster droplet maintenance.
- We re-center each of these K clusters so as to create a new centroid from each of these K newly created clusters. This process of re-centering simply creates a new droplet which is the union of the all the pseudo-points (droplets) assigned to that particular seed.

This iterative process is repeated until the centroids of the cluster droplets converge to a stable set. As in most real applications, the use of 4 or 5 iterations suffices. As a practical matter, one can set the number of iterations at 10. At this point, the higher level cluster droplets can be reported as the final clusters.

4.2. Outlier Creation

We define *newly created outliers* in (t_1, t_2) as the data points in those clusters which were created in (t_1, t_2), and which contain only singleton points. We have chosen this definition, since the addition of one more point to that cluster could be interpreted as a trend rather than an outlier by time t_2. The process of finding all the newly created outliers in the interval (t_1, t_2) is quite simple. In this case, we need to find all those droplets in $\mathcal{R}(t_2)$ which contain only one data point, and for which the last update time is larger than t_1. Note that the last update time is the same as the creation time in this case. We note that the cluster droplet statistics contain both the number of points and the last update time. This is utilized in order to determine the relevant droplets which correspond to the newly created outliers. A similar process may be used in order to find all the newly created outliers in the time window (t_1, t_2). The only difference is that in this case, we need to ensure that the corresponding cluster has become inactive at time t_2. This is because we are not looking for a new pattern of activity but a newly created cluster which did not receive a sufficient number of new data points. This can be determined by using the weight of the corresponding cluster droplet at time t_2. If the weight $w(t)$ of the droplet is less than 0.5, then the cluster is deemed to be inactive. Such a cluster is considered inactive at time t_2. Similarly, all those clusters with weight larger than 0.5 at time t_1, but with weight less than 0.5 at time t_2 are said to be clusters which have become inactive in the interval (t_1, t_2).

4.3. Correlation Analysis

Finally, we discuss the process of finding pairwise correlations between attribute values. In order to achieve this goal, the second order cluster statistics $\overline{DF2}$ need to be used. Since these cluster statistics contain the pairwise count between different attribute-value pairs, they can be used for the purpose of determining all pairs of attribute values which satisfy a certain support threshold. We note that in this model, the pairs of attribute values satisfying the support threshold

are *cluster specific*. Such cluster-specific pairs of attribute values are local to a particular part of the data. Thus, these correspond to the prominent localized correlations in the cluster within a given temporal locality. Such local correlations are often quite different from global correlations, and provide unique insights which cannot be obtained otherwise (Aggarwal et al, 2002).

5. Empirical Results

We tested the clustering system for quality and performance over text and market basket data sets. We also wanted to test the behavior of the stream clustering algorithm under different levels of temporal locality. Clearly, it is more desirable to have a data stream clustering algorithm which can quickly adapt to the evolution in the stream in such a way that the clusters represent the behavior within a given temporal locality.

For the text data sets, we utilized documents obtained from a sample of the 1996 scan of the $Yahoo!$ taxonomy. This stream contained $1.63 * 10^5$ documents. A stream was synthetically created from this scan by creating an order which matched a depth-first traversal of the $Yahoo!$ hierarchy. Thus, the resulting stream identically simulates a depth first crawl of the $Yahoo!$ taxonomy. Since web pages at a given node in the hierarchy are crawled at one time, the web pages are also contiguous by their particular class as defined by the $Yahoo!$ labels.

For the purpose of categorical data streams we used market basket data sets. In order to generate the base market basket data sets, we used the procedure discussed in (Agrawal et al, 1994). A transaction generated from this model is denoted by Tx.Iy.Dz, where x, y, and z are parameters of the data model. In this technique, a correlation model is used to construct base *maximum potential itemsets* from which all transactions are generated. The maximum potential itemsets are generated from a base of 1000 items by randomly selecting items with an average length selected from a poisson distribution with parameter y. The transaction length is chosen from a poisson distribution with parameter x. The transactions are constructed by combining randomly chosen potential itemsets in order to achieve an average length of x. The bias of choosing a maximal potential itemset is decided by an exponential distribution. When the potential itemsets are added to a transaction, they are typically perturbed slightly (as described in (Agrawal et al, 1994)), by adding or deleting a few items. A total of z items are then generated which results in a data set which is denoted by Tx.Iy.Dz. For example, the data set T20.I10.D10K corresponds to a data set with transaction length 20, maximal potential itemset size 10, and with 10,000 records. These data sets were converted into a stream for the testing process. The conversion process was performed as follows:

- A total of $n' = 10$ different random settings of the data set T20.I10.D10K of (Agrawal et al, 1994) were generated. Each such "instance" of the data set was generated by using a different random seed.
- A continuous stream of records was created by concatenating the different instances of the data sets with one another. Since each data set contained $b = 10,000$ records, the corresponding stream consisted of $100,000$ records. The market basket data stream was referred to as MStream1(S). We note that this stream has a very high level of temporal locality in its behavior.

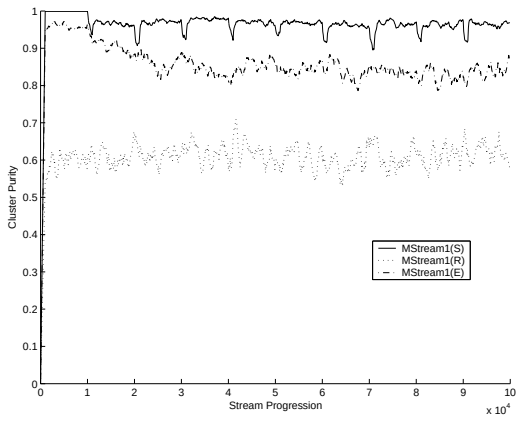

Fig. 3. Cluster Purity with Stream Progression (Market1 Data Stream)

– A second stream was generated from the same set of records, but in this case, the order of the records from the different data sets was randomized. Thus, a data point at a given stage of the stream could be generated from any of the sets of data. We refer to this stream as MStream1(R). This stream has almost no temporal locality.
– A third stream was created which continuously evolves over time. In order to create this smoothly evolving data stream, we applied a block mixing procedure in a sequential fashion. In the first step, the first $2 \cdot b$ records were randomized. In the next step, the block of records in the range $(b, 3 \cdot b)$ were randomized. This process was repeated sequentially for each contiguous block of $2 \cdot b$ records, at intervals of b records. The result was a data stream in which the evolution was more continuous than the original data. This stream exhibits a medium level of temporal locality. We refer to this data set as MStream1(E).

A second data stream was generated using the same methodology, except that the base data generation procedure used the data set T20.I8.D10K. We note that the change in the size of the underlying maximal potential itemset is useful in determining the behavior of the data sets with somewhat different correlation characteristics. The corresponding data streams are referred to as MStream2(R), MStream2(S), and MStream2(E) respectively. For the case of the text data streams, the definition of a class was more straightforward, since the original *Yahoo*! class labels could be directly used. However, in this case, the *Yahoo*! hierarchy was truncated in order to create 251 classes. In the Text(S) stream, the documents are contiguous by class and are arranged as if the *Yahoo*! taxonomy was crawled in depth-first order. As in the case of the market basket data sets, we used the same methodology to generate the other two data streams corresponding to Text(R) and Text(E). Finally, we constructed a data stream from the 20 newsgroups data set (Newsgroups, Data). The approach for constructing the data stream was exactly the same as the previous case, and three streams were constructed corresponding to the random, slowly evolving and rapidly evolving cases. The corresponding data sets are referred to as NG20(S), NG20(E), and NG20(R) respectively.

Thus, the data generation process created the following three types of data streams:

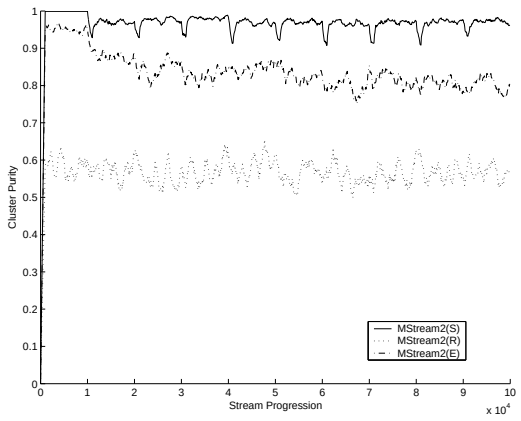

Fig. 4. Cluster Purity with Stream Progression (Market2 Data Stream)

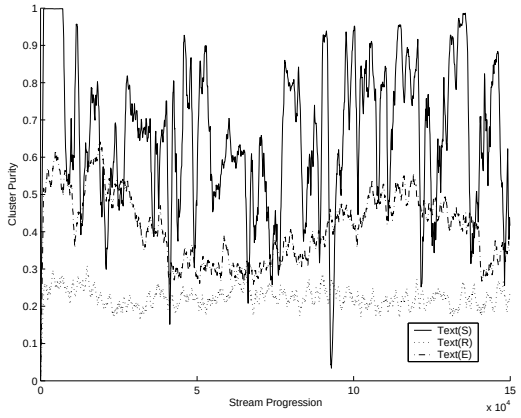

Fig. 5. Cluster Purity with Stream Progression (Text Data Stream)

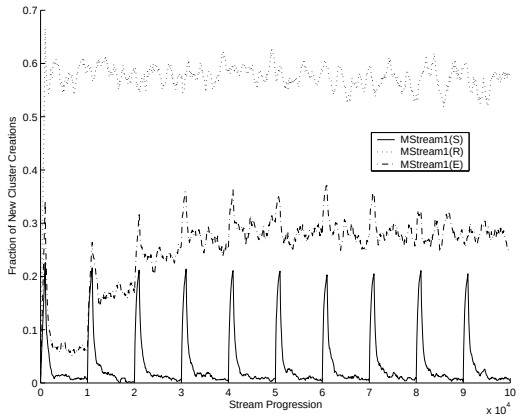

Fig. 6. Fraction of New Cluster Creations with Stream Progression (Market1 Data Stream)

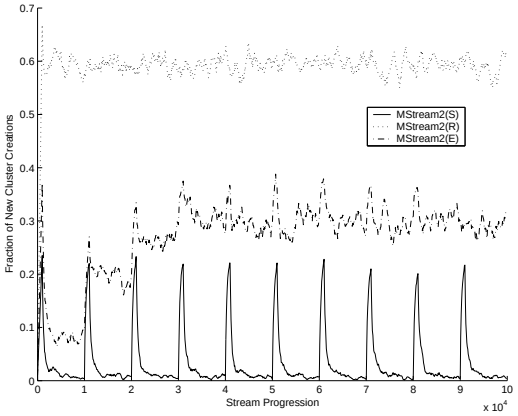

Fig. 7. Fraction of New Cluster Creations with Stream Progression (Market2 Data Stream)

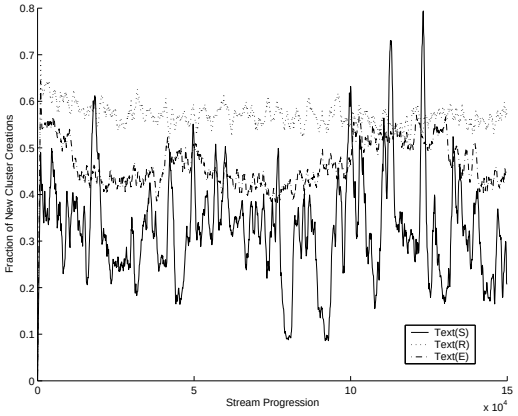

Fig. 8. Fraction of New Cluster Creations with Stream Progression (Text Data Stream)

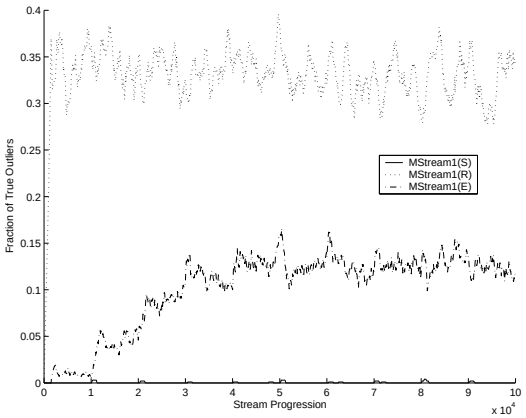

Fig. 9. Fraction of True Outliers with Stream Progression (Market1 Data Stream): One curve for MStream1(S) closely overlaps with X-axis

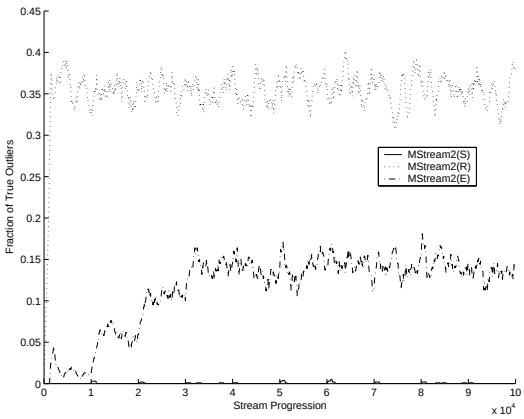

Fig. 10. Fraction of True Outliers with Stream Progression (Market2 Data Stream): One curve for MStream2(S) closely overlaps with X-axis

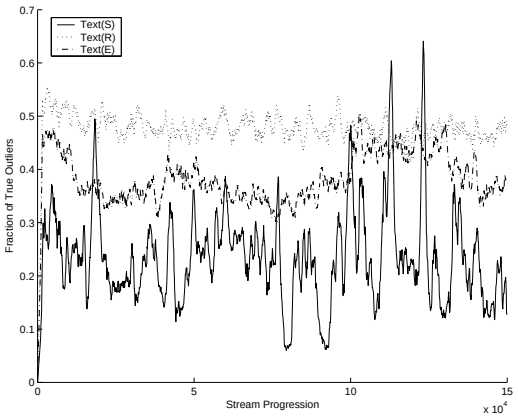

Fig. 11. Fraction of True Outliers with Stream Progression (Text Data Stream)

- The S-Class Streams correspond to Text(S), MStream1(S) and MStream2(S). These were the data streams with continuous segments of records belonging to a particular class. The evolution in this class is abrupt, when records move from one class to the other. The temporal locality was very high in this case.
- The E-class streams correspond to Text(E), MStream1(E), and MStream2(E). These were the smoothly evolving data streams with intermediate level of temporal locality.
- The R-class streams correspond to Text(R), MStream1(R) and MStream2(R). These were the non-evolving data streams with low temporal locality.

We tested how well the algorithm grouped records corresponding to a given class or category in a cluster. For the case of the market basket data sets, a class was defined as the set of records created by a particular instantiation of the random seed. Therefore, there were 10 "classes" in the case of the market basket

Data Set	S-Class	E-class	R-class
MStream1	96.86%	86.3%	61.0%
MStream2	97.1%	84.35%	56.8%
Text	65.28%	41.07%	22.47%
NG20	83.47%	68.26%	41.35%

Table 1. Overall Cluster Purity (Entire Stream)

data set. For each cluster, we defined the *birthing* class as the class label of the record which created the cluster. The fraction of the records in the cluster which belong to the birthing class is defined as the *class purity* of the cluster. We note that the value of the class purity can be computed only for those clusters which have more than one data point. This is because the first data point in a given cluster is not used in order to compute the cluster purity. At any given moment in time, the birthing accuracy was computed over only the last 1000 points in the stream. Thus, among the last 1000 data points, if n_1 be the number of trend-setting outliers, and n_2 be the number of non-trend setting points which match their birthing class label, the cluster purity is defined by $n_2/(1000 - n_1)$.

We wanted to test the effect of the evolution process on the clusters and outliers detected by the algorithm. For this purpose, we tested the number of new outliers created at each stage of the stream generation process. In general, outliers were created by substantial changes in the stream behavior. When a completely new and sustainable trend in the data appears, then the trend-setting outlier turns into a mature cluster. On the other hand, in cases when the cluster dies before the arrival of a second data point, the outlier is recognized to be a true anomaly. We tracked both the number of trend setting clusters as well as the number of true anomalies in the data, and plotted these values over the course of the data stream progression.

For all experiments, we assumed a slowly decaying stream in which the half-life was set to half the length of the stream. For the case of the initial qualitative statistics, we used $k = 1000$ droplets for the clustering process. In some of the later cases, we used the second phase of the clustering process to compare against a baseline approach, when we used the natural number of clusters as an input parameter. For the insertion of points into a cluster we used a similarity threshold which was 3 standard deviations less than the average similarity of the data points to the centroid of that cluster.

We have illustrated the cluster purity results using the different data sets in Figures 3, 4 and 5 respectively. In Figures 3 and 4, the S-Class stream is on top with the highest level of cluster purity, whereas the R-class stream is on the bottom with the lowest level of cluster purity. We note that in the case of the MStream1(S) and MStream2(S) data sets, the effect of the evolution of the data stream are quite pronounced and periodic. This is because the different patterns in the data are located at regular intervals. It is interesting to see that there is a considerable reduction in the class purity at each interval of $b = 10,000$ records for the MStream1(S) and MStream2(S) data sets. In the case of the Text(S) data set, the cluster purity behavior was much more irregular. This is because some of the classes in the original *Yahoo*! taxonomy do not

correspond to coherent sets of documents. For example, some of the documents from the @*Business_And_Economy* category of *Yahoo*! correspond to disparate businesses with few keywords in common. In such cases, the classification accuracy can vary rapidly with progression of the data stream. This can be seen in some cases of Figure 5, since the classification accuracy bounces back and forth at some regions of the stream.

In some other cases, the number of documents in a class was less than 10. In such cases, it was not possible to easily put a document in the cluster with the correct birthing class. On the other hand, for classes containing a larger number of documents, it was easy for the clustering process to adapt to the new class of documents. In such cases, the corresponding classification accuracy increased considerably. However, the overall level of cluster purity was higher in the Text(S) data set because of the greater level of temporal locality in the document data stream. As illustrated in Table 1, the text and newsgroup data sets had a higher level of cluster purity for the S-class and E-class data sets, as compared to the R-class data sets. This was also the case with the two market basket data sets. In the case of the Text(R), MStream1(R) and MStream2(R) data sets, there was no significant change in the cluster purity over the course of the data stream computation. This was also the case for the smoothly evolving data streams Text(E), MStream1(E), and MStream2(E) in which there was no significant change in the cluster purity during stream progression. In each case, the S-class streams had the highest cluster purity, whereas the R-class streams had the lowest cluster purity. The reason for this was that the R-class streams had a larger number of classes running in parallel. This resulted in a greater number of available possibilities in the class labels of the current clusters. On the other hand, the S-Class streams were pure within a given local segment. As a result, the corresponding cluster purity was much higher in this case.

In Figures 6, 7 and 8, we have illustrated the fraction of new cluster creations with stream progression for the different data sets. Again, the R-class streams had the highest fraction of new cluster creations, whereas the S-class streams had the smallest fraction of new cluster creations. At each interval of $b = 10,000$ records, the MStream1(S) and MStream2(S) data streams showed a higher fraction of new cluster creations. The text data stream was somewhat more irregular, but the new clusters were created whenever a new class of records arrived. We note that this coincides with the precise periods at which the patterns changed in the underlying data sets. For the case of the E-class and R-class data streams, the fraction of new cluster creations was more uniform. However, the average number of new cluster creations was higher in this case because of the greater non-uniformity of the data stream in the R-class and E-class data sets. More details of the quality of the underlying results are illustrated in Tables 5 and 6.

Not all the new cluster creations were true outliers. In Figures 9, 10 and 11, we have illustrated the fraction of true outliers with progression of the data stream. As in the previous case, the S-class and E-class streams had fewer outliers because of the lower level of heterogeneity in the data stream. In fact, in the case of the MStream1(S) and MStream2(S) data sets, no true outliers were created throughout the entire cluster maintenance process. Consequently, the number of true outliers for the MStream1(S) and MStream2(S) data sets in Figures 9 and 10 cannot be distinguished from the X-axis. A natural observation from these results is that the number of outliers in a data stream is highly dependent upon the ordering and skew of the underlying data stream. Clearly, when there are bursts of homogeneous behavior within a given temporal locality of the data

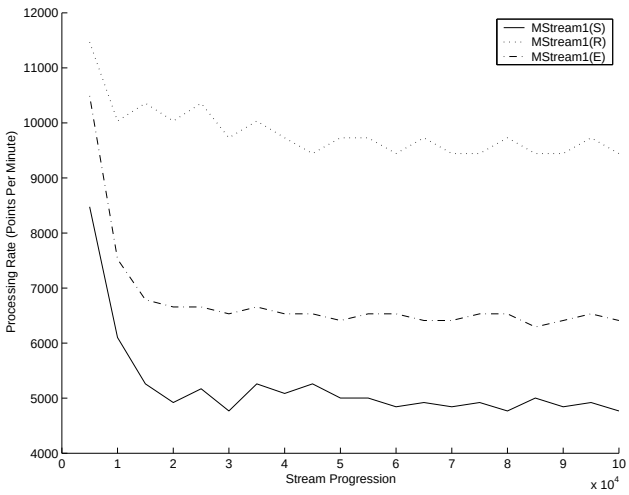

Fig. 12. Processing Rate with Stream Progression

Data Set	Outlier Precision	Inactive Cluster Precision
MStream1(E)	89%	98.3%
MStream2(E)	87.1%	95.4%
Text(E)	82.3%	89.1%

Table 2. Horizon-specific and Outlier Behavior

stream (as in the case of the S-class data sets), very few outliers are observed. The rate of outlier generation is thus an indirect indicator of the level of temporal locality of the underlying data stream. Data streams with lower temporal locality are likely to have a greater number of outliers. This is because there are likely to be fewer clusters corresponding to consistent patterns of activity in the data stream.

5.1. Horizon Specific Applications and Outlier Detection

In this section, we will test the behavior of the different horizon-specific applications discussed in this paper. The horizon-specific applications are very useful in determining the evolution behavior over specific time horizons. In order to test this aspect, we used the continuously evolving E-class streams. For example in the case of the market basket data stream, the positions b, $2 \cdot b$, $3 \cdot b$, ... $9 \cdot b$ formed the change points at which a new pattern was introduced within the stream. These were also the positions at which some of old patterns in the stream expired. Therefore, we computed two metrics:

– For each horizon $[r \cdot b, (r+1) \cdot b]$, we computed the fraction of trend setting out-

Data Set	E-class ConStream	E-class OSKM	R-class ConStream	R-class OSKM
MStream1	90.6%	81.7%	73.2%	65.8%
MStream2	89.5%	80.3%	71.7%	64.3%
Text	51.07%	44.7%	33.56%	21.32%
NG20	87.54%	81.44%	46.35%	38.27%

Table 3. Overall Cluster Precision (Entire Stream)

Data Set	E-class ConStream	E-class OSKM	R-class ConStream	R-class OSKM
MStream1	88.3%	81.8%	71.5%	64.3%
MStream2	87.4%	80.0%	70.3%	63.2%
Text	49.05%	45.1%	31.4%	20.5%
NG20	83.32%	80.7%	45.1%	36.3%

Table 4. Overall Cluster Recall (Entire Stream)

liers which correspond to new patterns in the underlying data. In other words, the first data point in the cluster corresponds to a new pattern of activity. We refer to this metric as the *outlier precision*. This metric was averaged over all horizons.

– For each horizon $[r \cdot b, (r+1) \cdot b]$, we determined the percentage of replaced clusters which correspond to expired patterns. In this case, the majority of points in the cluster need to be drawn from an expired pattern. We refer to this metric as the *inactive cluster precision*. This metric was averaged over all horizons. The outlier precision and the inactive cluster precision for the data sets are illustrated in Table 2. We note that the precision values are typically much higher than 80% in most cases. The inactive cluster precisions are typically higher since inactivity behavior is determined by the behavior of the clustering over a period of time, and therefore more robust.

5.2. Baseline Comparisons

We compared the method to the OSKM algorithm discussed in (Zhong, 2005), which is the most recent method for streaming text clustering. The algorithm in (Zhong, 2005) was adapted to the market basket data set by using the similarity measure discussed in (Aggarwal et al, 1999) for the partitioning step. In the case of the OSKM algorithm, we used $M = 20$ iterations, segment length $S = 2000$ and $\gamma = 0.2$ (using the notations of (Zhong, 2005)). The learning rate schedule of OSKM was set by $\eta_0 = 1.0$ and $\eta_f = 0.01$. We further note that both algorithms require an input parameter k which defines the number of clusters. In order to test the the relative effectiveness of the two methods, we used an input parameter k which was equal to the number of natural clusters in the data

Data Set	E-class ConStream	E-class OSKM	R-class ConStream	R-class OSKM
MStream1	89.6%	75.3%	72.4%	58.5%
MStream2	88.4%	72.1%	70.5%	59.2%
Text	52.1%	41.2%	32.1%	16.1%
NG20	83.1%	74.2%	44.7%	31.9%

Table 5. Overall Cluster Precision (Horizon Specific)

Data Set	E-class ConStream	E-class OSKM	R-class ConStream	R-class OSKM
MStream1	87.1%	74.8%	71.5%	58.6%
MStream2	86.95%	70.3%	68.7%	58.3%
Text	49.4%	39.5%	31.4%	15.5%
NG20	81.2%	73.3%	51.5%	32.0%

Table 6. Overall Cluster Recall (Horizon Specific)

set. Since this parameter is set to the same value for both algorithms, the results should provide a good idea of the overall effectiveness of the approach. In the case of the ConStream approach, the number of droplets was still set at 1000, though the second phase of the clustering process reduced the number of droplets to the number of natural clusters in the data set. For the purpose of testing, we utilized two approaches:

- The entire data stream was clustered in one pass, and the precision and recall of the two methods were measured.
- We calculated the accuracy of clustering over the middle third of the stream. In the case of the *ConStream* method, this means that the online clustering for constructing the droplets was performed on the entire stream, but the second phase of offline phase of clustering the droplets was performed only over the horizon-specific portion after applying the subtractivity property. The precision and recall were measured over the final result of the clustering. On the other hand, in the case of the OSKM algorithm, the precision and recall were measured after discarding the irrelevant documents from the first and last third of the stream in the clusters. This ensures a fair comparison between the two methods.

In Tables 3 and 4 we have illustrated the precision and recall behavior of the *ConStream* method with respect to the OSKM approach. The precision and recall were computed against the true clusters which were either known from the base data in the case of market basket data sets or were available in the form of categories in the case of the two text data sets. Since the results on the S-class data sets are near perfect for all methods, we have illustrated the results only on the E-class and R-class data sets for all methods. In each case,

Data Set	Peak Consumption/droplet	Total Consumption
MStream1(E)	127.2KB	127 MB
MStream2(E)	131.5KB	131 MB
Text(E)	863.2KB	863 MB

Table 7. Memory Consumption Requirements

it is clear that the *ConStream* method substantially outperforms the OSKM technique in terms of both precision and recall. In the particular case of the *ConStream* method, the precision is slightly higher than the recall, since some of the droplets were dropped by the cluster maintenance algorithm. However, even in terms of the recall, the *ConStream* method was substantially superior to the OSKM algorithm. This is because the use of fine grained cluster droplets in combination with a parameter-specific user phase provides a much higher level of accuracy than the OSKM algorithm. We note that the complexity of the second phase is not proportional to the number of data stream points, but only to the number of cluster droplets.

The advantages of the *ConStream* method over the OSKM algorithm become even more apparent when the precision and recall are computed only over a user-specified horizon. As discussed above, only the middle third of the stream was used for the purpose of computing the precision and recall. In this case, the difference between the *ConStream* method and the OSKM method was quite significant. This is because the *ConStream* method was able to use the additive property in order to restrict the second phase to a particular user-specified horizon. This gave it a clear advantage over the OSKM method. This also illustrates the advantage of the *ConStream* method over the OSKM method over a variety of applications in which the cluster-specific parameters may be utilized at query-time. In such cases, the droplet construction provides a summary of the data which can be efficiently aggregated at query time in order to construct the final set of high-level clusters.

5.3. Memory Consumption

We also tested the memory consumption of the ConStream approach. The memory consumption requirement is defined by the space required to hold the cluster droplets. Therefore, we tested the memory requirement of the approach. We computed the average memory requirement of the approach *over all cluster droplets*, and then computed the *peak memory requirement per droplet over the entire data stream*. The results are presented for the MStream1(E), MStream2(E) and Text(E) data sets in Table 7. We note that the memory requirements are quite modest and typically in the range of a few hundred kilobytes per droplet. This is essentially because of the sparsity of the underlying data set in which the co-occurence of a pair of data values is quite rare in a given record. Since the droplets use a sparsity-friendly representation, this greatly reduces the memory requirements when the number of possible values is large. With current main memory availability in desktop machines of the order of gigabytes, this effectively means that it is possible to run the stream clustering algorithm with

Data Set	Average Proc. Rate (ConStream) (pts per min.)	Average Proc. Rate (OSKM) (pts per min.)
MStream1(S)	5314.4	2664.7
MStream1(E)	6586.1	2682.3
MStream1(R)	9538.9	2673.6
MStream2(S)	5123.8	2601.5
MStream2(E)	6629.8	2631.6
MStream2(R)	9950.2	4622.4
Text(S)	4231.9	2435.7
Text(E)	4580.9	2501.6
Text(R)	4863.2	2522.7
NG20(S)	4361.2	2541.6
NG20(E)	4781.3	2561.3
NG20(R)	5134.5	2556.7

Table 8. Running Times

thousands of cluster droplets. In this particular case, we used $k = 1000$ droplets, and the corresponding total peak consumption may be obtained by multiplying the per-droplet consumption by 1000. The total peak requirement is also illustrated in Table 7. The results suggest that the approach is quite scalable in terms of memory requirements. Next, we will test the scalability in terms of efficiency.

5.4. Efficiency

We have also compared the efficiency of our text clustering method with the OSKM algorithm. For the case of the OSKM algorithm, we used the same set of parameters as discussed in section 5.1, except that we used $k = 1000$ consistently for both algorithms. In Table 8, we have illustrated the stream processing rate for each of the data sets and the two methods. In each case, our method is significantly faster than the OSKM algorithm. It is clear that in each case, several thousand data points per minute could be processed. The other observation is that the market basket data sets exhibit much better performance than the text data sets. This is because the text data sets had a significantly larger number of attributes (text words) in each record. As a result, the corresponding processing times were faster in the case of the market basket data. We also note that the running time was not uniform across the entire execution. In Figure 12, we have illustrated the processing rate with progression of the data stream the market basket data set MStream1(E). The numbers on the Y-axis illustrate the rate at which the last 5000 data points were processed. The behavior for the other data sets was very similar. As evident from Figure 12, the initial processing rate is somewhat higher than the steady state processing rate. This is because the number of cluster droplets at the initial stage of the algorithm execution

is significantly lower. In addition, at the earlier stages of the stream, most of the attribute values/text words in the sparse data are not represented in the cluster. Over the course of progression of the data stream, the number of values represented increases. As this number increases and reaches a steady state, the corresponding rate of processing stabilizes over time. Another observation was that the processing rates of the data sets with lower temporal locality were much higher. This is because in such data streams, the clusters get discarded more quickly as outliers. This also means that the average life of a cluster is relatively low in such streams. In such cases, not much statistics is maintained for the different attributes in the underlying sparse data. This results in faster similarity computations. While there was variation across the different data streams in terms of the processing speed, the overall speed was over a few thousand data points per minute in each case. This suggests that the approach is extremely efficient and robust over different kinds of data sets.

6. Conclusions and Summary

In this paper, we discussed a new method for clustering and outlier detection of text and categorical data streams. In order to achieve this goal, we used a compact representation of the clusters which was utilized to construct an additive data stream mining algorithm. The resulting algorithm was applicable over both the text and categorical data mining domain with minor modifications. In addition, the technique can be applied to study the nature of the outliers and the evolution in the underlying stream. In addition, since the approach stores summary data about the clusters, it can be used in conjunction with a second level of clustering based on user-specified parameters. This is possible because of the additivity property of the droplets, which make it feasible for the clustering to be performed on user-specified horizons with the use of a second offline phase. In most real applications, a stream may continue to evolve over time, but a user may query for clusters over different time horizons. For such cases, the additivity property is particularly useful at the cost of some offline processing.

The algorithm was tested on a number of real and synthetic data sets. We found the algorithm to be highly effective in being able to quickly adapt to variations in the data stream and recognize the underlying temporal locality. We also tested the method against a recent stream clustering method known as OSKM using traditional measures such as precision and recall. In such cases, our method turns out to be much more effective, and the advantage was greater when the query was restricted to a particular user-specified horizon. We also tested the method for scalability, and it turns out to be highly efficient over a variety of data sets.

References

D. Agrawal, Detecting anomalies in cross-classified streams: a Bayesian approach, *KAIS Journal*, 11(1), pp. 29–44, 2007.

C. C. Aggarwal, P. S. Yu. A Framework for Clustering Uncertain Data Streams. *ICDE Conference*, 2008.

C. C. Aggarwal, P. S. Yu. Finding Localized Associations in Market Basket Data, *IEEE Transactions on Knowledge and Data Engineering*, 2002.

C. C. Aggarwal. A Framework for Diagnosing Changes in Evolving Data Streams. *ACM SIGMOD Conference*, 2003.
C. C. Aggarwal, J. Han, J. Wang, P. Yu. A Framework for Clustering Evolving Data Streams. VLDB Conference, 2003.
C. C. Aggarwal, P. S. Yu. A Framework for Clustering Massive Text and Categorical Data Streams. *ACM SIAM Data Mining Conference*, 2006.
C. C. Aggarwal, C. Procopiuc, P. S. Yu. Finding Localized Associations in Market Basket Data. *IEEE TKDE Journal*, 2002.
R. Agrawal, R. Srikant. Fast Algorithms for Mining Association Rules. *VLDB Conference*, 1994.
J. Allan, J. Carbonell, G. Doddington, J. Yamron, Y. Yang. Topic Detecting and Tracking Pilot Study Final Report. *Proceedings of the Broadcast News Understanding and Transcription Workshop*, 1998.
J. Allan, R. Papka, V. Lavrenko. On-line new event detection and tracking. *ACM SIGIR Conference*, pp. 37–45, 1998.
P. Andritsos, P. Tsaparas, R. Miller, K. Sevcik. LIMBO: Scalable Clustering of Categorical Data. *EDBT Conference*, 2004.
B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom. Models and Issues in Data Stream Systems, *ACM PODS Conference*, 2002.
A. Banerjee, J. Ghosh. Frequency-sensitive competitive learning for scalable balanced clustering on high-dimensional hyperspheres, *IEEE Transactions on Neural Networks*, 15, pp. 702–719, 2004.
A. Banerjee, S. Basu. Topic Models over Text Streams: A Study of Batch and Online Unsupervised Learning, *ICML Conference*, 2007.
V. Barnett, T. Lewis. Outliers in Statistical Data. *John Wiley and Sons*, NY 1994.
M. M. Breunig, H.-P. Kriegel, R. T. Ng, J. Sander. LOF: Identifying Density-Based Local Outliers. *Proceedings of the ACM SIGMOD Conference*, 2000.
P. Bradley, U. Fayyad, C. Reina. Scaling Clustering Algorithms to Large Databases. *SIGKDD Conference*, 1998.
B. Cao, M. Ester, W.Qian, A. Zhou. Density Based Clustering of Evolving Data Stream with Noise. *SIAM Data Mining Conference*, 2006.
Y. Chen, L. Tu. Density-based clustering for real-time stream data. *KDD Conference*, 2007.
C. Cortes et al. Hancock: A Language for Extracting Signatures from Data Streams. *ACM SIGKDD Conference*, 2000.
D. Cutting, D. Karger, J. Pedersen, J. Tukey. Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections. *Proceedings of the SIGIR*, 1992, pages 318-329.
P. Domingos, G. Hulten. Mining High-Speed Data Streams. *ACM SIGKDD Conference*, 2000.
D. Fisher. Knowledge Acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.
M. Franz, T. Ward, J. Scott McCarley, W.-J. Zhu. Unsupervised and supervised clustering for topic tracking. *SIGIR Conference*, 2001.
J. H. Gennari, P. Langley, D. Fisher. Models of incremental concept formation. *Journal of Artificial Intelligence*, 40:11-61, 1989.
D. Gibson, J. Kleinberg, P. Raghavan. Clustering Categorical Data: An Approach Based on Dynamical Systems, *Proceedings of the VLDB Conference*, 1998.
S. Guha, R. Rastogi, K. Shim. ROCK: a robust clustering algorithm for categorical attributes, *Proceedings of the International Conference on Data Engineering*, 1999.
Q. He, K. Chang, E. P. Lim, J. Zhang. Bursty Feature Representation for Clustering Text Streams. *SDM Conference*, 2007.
E. Knorr, R. Ng. Algorithms for Mining Distance-based Outliers in Large Data Sets. *Proceedings of the VLDB Conference*, September 1998.
M. Li, G. Holmes, and B. Pfahringer. Clustering large datasets using cobweb and k-means in tandem. In G. I. Webb and X. Yu, (editors), *Proc 17th Australian Joint Conference on Artificial Intelligence*, volume 3339 of LNAI, pages 368-379, Cairns, Australia, 2004. Springer.
Y. Li, R. Gopalan. Clustering Transactional Data Streams. *Advances in Artificial Intelligence*, 2006.
R. Ng, J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. *Very Large Data Bases Conference*, 1994.
L. O'Callaghan et al. Streaming-Data Algorithms For High-Quality Clustering. *ICDE Conference*, 2002.

G. L. Peterson, B. T. McBride, The importance of generalizability for anomaly detection, *KAIS Journal*, Vol 14, No. 3, pp. 377–392, 2008.

S. Ramaswamy, R. Rastogi, K. Shim. Efficient Algorithms for Mining Outliers from Large Data Sets. *Proceedings of the ACM SIGMOD Conference*, 2000.

C. Silverstein, J. Pedersen. Almost-constant time clustering of arbitrary corpus sets. *Proceedings of the ACM SIGIR*, pages 60-66, 1997.

A. Surendran, S. Sra. Incremental Aspect Models for Mining Document Streams. *Principles of Knowledge Discovery and Data Mining (PKDD)*, 2006.

Y. Yang, T. Pierce, J. Carbonell. A Study on Retrospective and On-line Event Detection. *Proceedings of the SIGIR Conference*, 1998.

Y.Yang, J. Carbonell, C. Jin. Topic-conditioned Novelty Detection. *ACM KDD Conference*, 2002.

T. Zhang, R. Ramakrishnan, M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. *ACM SIGMOD Conference,* 1996.

J. Zhang, Z. Ghahramani, and Y. Yang. A probabilistic model for online document clustering with application to novelty detection. In *Saul L., Weiss Y., Bottou L. (eds) Advances in Neural Information Processing Letters*, 17, 2005.

S. Zhong. Efficient Streaming Text Clustering. *Neural Networks*, Volume 18, Issue 5-6, 2005.

http://people.csail.mit.edu/jrennie/20Newsgroups/

Author Biographies

insert photo

Charu C. Aggarwal is a Research Staff member at the IBM T. J. Watson Research Center in Yorktown Heights, New York. He completed his B.S. from IIT Kanpur in 1993 and his Ph.D. from MIT in 1996. He has published over 130 papers, and has been granted over 45 patents. He has thrice been designated a Master Inventor at IBM for his patents. He is a recipient of an IBM Corporate Award (2003), a recipient of the IBM Outstanding Innovation Award (2008), and a recipient of an IBM Research Division Award (2008) for various contributions to stream and privacy research. He has served as program vice-chairs of the SIAM Conference on Data Mining, 2007, the IEEE ICDM Conference, 2007, the WWW Conference 2009, and the IEEE ICDM Conference, 2009. He served as an associate editor of the IEEE TKDE from 2004 to 2008. He is an action editor of the DMKD Journal, an associate editor of the ACM SIGKDD Explorations, and an associate editor of the KAIS Journal.

insert photo

Philip S. Yu received his Ph.D. degree in E.E. from Stanford University. He is a Professor in Computer Science at the University of Illinois at Chicago and also holds the Wexler Chair in Information Technology. Dr. Yu spent most of his career at IBM, where he was manager of the Software Tools and Techniques group at the Watson Research Center. His research interests include data mining, database and privacy. He has published more than 530 papers in refereed journals and conferences. He holds or has applied for more than 350 US patents.

Dr. Yu is a Fellow of the ACM and the IEEE. He is associate editors of ACM Transactions on the Internet Technology and ACM Transactions on Knowledge Discovery from Data. He was the Editor-in-Chief of IEEE Transactions on Knowledge and Data Engineering (2001-2004). He received a Research Contributions Award from IEEE Intl. Conference on Data Mining (2003).

Correspondence and offprint requests to: Charu C. Aggarwal, IBM T. J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532. Email: charu@us.ibm.com