Chapter 10

A Survey of Stream Clustering Algorithms

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
charu@us.ibm.com

10.1 Introduction

In recent years, advances in hardware technology have allowed us to automatically record transactions and other pieces of information of everyday life at a rapid rate. Such processes generate huge amounts of online data which grow at an unlimited rate. These kinds of online data are referred to

as *data streams*. The issues on management and analysis of data streams have been researched extensively in recent years because of its emerging, imminent, and broad applications [1].

Many important problems such as clustering and classification have been widely studied in the data mining community. The problem has been investigated classically in the context of a wide variety of methods such as the *k*-means, *k*-medians, density-based methods, probabilistic clustering methods etc. A detailed discussion of different kinds of clustering methods may be found in [42, 44]. Most of the classical methods in the literature are not necessarily designed in the context of very large data sets and data streams. The stream scenario brings a unique set of challenges with it, which cannot be addressed by most of the classical methods proposed in [42, 44].

The specific challenges in the context of stream scenario are as follows:

- Streams typically have *massive volume*, and it is often not possible to store the data explicitly on disk. Therefore, the data needs to be processed in a *single pass*, in which all the summary information required for the clustering process needs to be stored and maintained. The time needed to process each record must be small and constant. Otherwise, the model construction process would never be able to catch up with the stream.

- The patterns in the data stream may continuously evolve over time [13]. From a stream mining perspective, this implies that the underlying cluster models need to be continuously updated. A usable model must be available at any time, because the end of stream computation may never be reached, and an analyst may require results at any point in time.

- Different domains of data may pose different challenges to data stream clustering. For example, in a *massive domain* of discrete attributes, it may not be possible to store summary representations of the clusters effectively without increasing the computational complexity of the problem significantly. Therefore, space-efficient methods need to be designed for massive-domain clustering of data streams.

Clearly, the issue of scalability is a primary one from the perspective of stream processing, but by no means is it the only issue. In the context of stream processing, temporal locality is also quite important, because the underlying patterns in the data may evolve, and therefore, the clusters in the past history may no longer remain relevant to the future. The issue of scalability also arises in the context of data clustering of *very large data sets* [31, 37, 51, 63]. In such cases, the major constraint is that the algorithm should require no more than one (or at least a small constant number of) pass(es) over the data. This is because I/O operations are traditionally quite expensive in databases, as compared to main memory operations. Therefore, the optimum performance is often achieved, when the underlying algorithm is designed to minimize the number of passes over the data, rather than minimize the number of CPU operations. In the context of data streams, temporal locality issues are also very important, and should not be ignored in the clustering process. Therefore, a variety of stream clustering algorithms attempt to take such temporal issues into account with the use of snapshot-based methods, decay-based techniques, windowing etc. We will make an effort to point such techniques where they are used.

This chapter is organized as follows. In the next section, we will study clustering methods which are based on the *k*-means or the *k*-medians methodology. In section 10.3, we will study density-based methods for stream clustering. Section 10.4 discusses probabilistic algorithms for clustering data streams. High dimensional streaming algorithms are introduced in section 10.5. Methods for discrete and categorical stream clustering introduced in section 10.6. Methods for clustering text streams are discussed in section 10.7. Other scenarios for data stream clustering are discussed in section 10.8. The conclusions and summary are introduced in section 10.9.

10.2 Methods based on Partitioning Representatives

A number of methods for clustering are based on partitioning representatives. These include methods such as the *k*-means and *k*-medians based methods. In these techniques, the clusters are defined by a set of data points, which are drawn either directly from the data set or otherwise. The cluster membership of the remaining points are defined by assigning them to their closest representative. In the classical literature, such methods are iterative and require multiple passes over the data in order to estimate the representatives accurately. However, in the stream scenario, multiple passes are not required because a sufficient amount of data is available for estimating the representatives efficiently in even one pass of the data. In this section, we will first discuss the stream clustering methods, which are based on the partitioning methodology.

10.2.1 The STREAM Algorithm

The *STREAM* framework [36, 53] is based on the *k*-medians clustering methodology. The core idea is to break the stream into *chunks*, each of which is of manageable size and fits into main memory. Thus, for the original data stream D, we divide it into chunks $D_1 \ldots D_r \ldots$, each of which contains at most m data points. The value of m is defined on the basis of a pre-defined memory budget.

Since each chunk fits in main memory, a variety of more complex clustering algorithms can be used for each chunk. The methods in [36, 53] use a variety of different *k*-medians style algorithms for this purpose. The choice the subroutine used is crucial to the quality of the underlying algorithm, and will be discussed in detail below. In *k*-medians algorithms, we pick a set S of k representatives from each chunk D_i, so that each point in D_i is assigned to its closest representatives. The goal is to pick the representatives in such a way, so as to minimize the *Sum of Squared Error (SSQ)* of the assigned data points from these representatives. For a set of m data points $\overline{X_1} \ldots \overline{X_m}$ in S, and a set of k representatives $Y = \overline{Y_1} \ldots \overline{Y_r}$, the objective function is defined as follows:

$$Objective(S, Y) = \sum_{\overline{X_i} \in S, \overline{X_i} \Leftarrow \overline{Y_{j_i}}} dist(\overline{X_i}, \overline{Y_{j_i}}) \tag{10.1}$$

The assignment operator is denoted by $\Leftarrow$ in the above. The sum of squared distances between a pair of records is denoted by $dist(\cdot, \cdot)$.

After the first chunk has been processed, we now have a set of k medians, which are stored away. The number of points assigned to each representative is stored as a "weight" for that representative. Such representatives are considered *level-1* representatives. The next chunk is independently processed in order to find its k optimal median representatives. Thus, at the end of processing the second chunk, we will have $2 \cdot k$ such representatives. Thus, the memory requirement for storing the representatives also increases with time, and after processing r chunks, we will have a total of $r \cdot k$ representatives. When the number of representatives exceeds m, a second level of clustering is applied to these set of $r \cdot k$ points, except that the stored weights on the representatives are also used in the clustering process. The resulting representatives are stored as *level-2* representatives. In general, when the number of representatives of level-p reaches m, they are converted to k level-$(p+1)$ representatives. Thus, the process will result in increasing the number of representatives of all levels, though the number of representatives in higher levels will increase exponentially slower than the lower levels. At the end of processing the entire data stream (or when a specific need for the clustering result arises), all remaining representatives of different levels are clustered together in one final application of the *k*-medians subroutine.

Clearly, the choice of the particular algorithm which is used for the *k*-medians problem is critical in providing an effective solution. The other factor which impacts the quality of the final output is

the effect of the problem decomposition into chunks followed by hierarchical clustering. How does such a problem decomposition affect the final quality of the output? It has been shown in [53], that the final quality of the output cannot be arbitrarily worse than the particular subroutine which is used at the intermediate stage for *k*-medians clustering.

Lemma 10.2.1 *Let the subroutine used for k-medians clustering in the* STREAM *algorithm have an approximation factor of c. Then, the* STREAM *algorithm will have an approximation factor of no worse than* $5 \cdot c$.

The work in [53] extends the original work in [36] by designing a more effective subroutine for the *k*-medians problem. This solution is based on the problem of facility location. We note that the *Lagrangian relaxation of the* clustering problem can be modeled as a facility location problem, wherein we relax the constraint on the number of representatives, and incorporate it into the objective function with a Lagrangian multiplier. These representatives are analogous to facilities in the context of the facility location problem, where the cost associated with a representative, can be physically visualized as the cost of "building" a facility, in order to service its assigned clients (data points). Thus, we add a cost λ (Lagrangian parameter) for each facility included. The cost of assignments is the same as before. Thus, the new objective function in terms of the set of facilities Y may be expressed as follows:

$$Objective(S, Y) = \sum_{\overline{X_i} \in S, \overline{X_i} \Leftarrow \overline{Y_{j_i}}} dist(\overline{X_i}, \overline{Y_{j_i}}) + \lambda \cdot |Y| \tag{10.2}$$

Unlike the previous case, the cardinality of Y may not necessarily be k at the optimal value of the objective function, unless the value of the Lagrangian parameter (or facility cost) λ is picked appropriately. In order to use this approach to solve the *k*-medians subroutine, two steps are needed [53]:

- Given a particular value of λ, how do we determine the optimal objective function value?
- How do we determine the value of λ in such a way that the number of facilities in the optimal solution are *k*.

The latter part of the solution is easy. As long as the optimal number of facilities changes monotonically with λ, one can use binary search on λ in order to determine the appropriate value. It was shown in [53] that the optimal number of facilities indeed changes monotonically with *k*.

In order to solve the first part of the problem, a local search algorithm called *LSEARCH* is proposed. In this algorithm, we start off with a current set of open facilities *O*, and a particular assignment of data points to these facilities (which is not necessarily optimal). Then, we repeatedly improve the solution by examining all the facilities *x* not in *O* exactly once in random order, and computing $gain(x)$, of adding *x* to the current set of open facilities, while re-assigning some of the points to *x* and closing some other facilities. Certain rules of re-assignment of data points to facilities and closing facilities are used for this purpose, as discussed below.

Specifically, any data point is allowed to be re-assigned to *x*, when *x* is added to *O* (if there is an improvement), and it is also allowed to close a facility *y* and assign *all* of its points to *x*. There latter case would result in a gain, as long as the cost of the re-assignment together to *x* is no greater than the savings from closing *y*. The cost of opening *x* is always subtracted from the overall gains. Thus, the value of $gain(x)$ may be positive or negative. The facility *x* is added to *O*, only if $gain(x) > 0$. This process is repeated over all facilities in random order, and the final set of open facilities and re-assignments represents one possible locally optimal solution. This process would need to be repeated $\Omega(\log(m))$ times in order to provide guarantees on the underlying quality.

A major limitation of the *STREAM* algorithm is that it is not particularly sensitive to evolution in the underlying data stream. In many cases, the patterns in the underlying stream may evolve and

change significantly. Therefore, it is critical for the clustering process to be able to adapt to such changes and provide insights over different time horizons. In this sense, the *CluStream* algorithm is able to provide significantly better insights at differing levels of temporal granularity.

10.2.2 CluStream: The Micro-clustering Framework

Since stream data naturally imposes a one-pass constraint on the design of the algorithms, it becomes more difficult to provide such a flexibility in computing clusters over different kinds of time horizons using conventional algorithms. For example, a direct extension of the stream based *k*-means algorithm in [53] to such a case would require a simultaneous maintenance of the intermediate results of clustering algorithms over all possible time horizons. Such a computational burden increases with progression of the data stream and can rapidly become a bottleneck for online implementation. Furthermore, in many cases, an analyst may wish to determine the clusters at a previous moment in time, and compare them to the current clusters. This requires even greater book-keeping and can rapidly become unwieldy for fast data streams.

Therefore a natural design to stream clustering would be separate out the process into an online micro-clustering component and an offline macro-clustering component. The online micro-clustering component requires a very efficient process for storage of appropriate summary statistics in a fast data stream. The offline component uses these summary statistics in conjunction with other user input in order to provide the user with a quick understanding of the clusters whenever required.

It is assumed that the data stream consists of a set of multi-dimensional records $\overline{X}_1 \ldots \overline{X}_k \ldots$ arriving at time stamps $T_1 \ldots T_k \ldots$. Each $\overline{X_i}$ is a multi-dimensional record containing d dimensions which are denoted by $\overline{X_i} = (x_i^1 \ldots x_i^d)$.

The micro-clustering framework is designed to capture summary information about the data stream, in order to facilitate clustering and analysis over different time horizons. This summary information is defined by the following structures:

- **Micro-clusters:** We maintain statistical information about the data locality in terms of micro-clusters. These micro-clusters are defined as a temporal extension of the *cluster feature vector* [63]. The additivity property of the micro-clusters makes them a natural choice for the data stream problem.

- **Pyramidal Time Frame:** The micro-clusters are stored at snapshots in time which follow a pyramidal pattern. This pattern provides an effective trade-off between the storage requirements and the ability to recall summary statistics from different time horizons.

The summary information in the micro-clusters is used by an offline component which is dependent upon a wide variety of user inputs such as the time horizon or the granularity of clustering. We will first begin by defining the concept of micro-clusters and pyramidal time frame more precisely.

10.2.2.1 Micro-Cluster Definition

Micro-clusters are defined as follows.

Definition 10.2.1 *A micro-cluster for a set of d-dimensional points $X_{i_1} \ldots X_{i_n}$ with time stamps $T_{i_1} \ldots T_{i_n}$ is the $(2 \cdot d + 3)$ tuple $(\overline{CF2^x}, \overline{CF1^x}, CF2^t, CF1^t, n)$, wherein $\overline{CF2^x}$ and $\overline{CF1^x}$ each correspond to a vector of d entries. The definition of each of these entries is as follows:*

- *For each dimension, the sum of the squares of the data values is maintained in $\overline{CF2^x}$. Thus, $\overline{CF2^x}$ contains d values. The p-th entry of $\overline{CF2^x}$ is equal to $\sum_{j=1}^{n} (x_{i_j}^p)^2$.*

- *For each dimension, the sum of the data values is maintained in $\overline{CF1^x}$. Thus, $\overline{CF1^x}$ contains d values. The p-th entry of $\overline{CF1^x}$ is equal to $\sum_{j=1}^{n} x_{i_j}^p$.*

- *The sum of the squares of the time stamps $T_{i_1} \ldots T_{i_n}$ is maintained in $CF2^t$.*
- *The sum of the time stamps $T_{i_1} \ldots T_{i_n}$ is maintained in $CF1^t$.*
- *The number of data points is maintained in n.*

The data stream clustering algorithm proposed in [14] can generate approximate clusters in any user-specified length of history from the current instant. This is achieved by storing the micro-clusters at particular moments in the stream which are referred to as *snapshots*. At the same time, the current snapshot of micro-clusters is always maintained by the algorithm. Consider for example, the case when the current clock time is t_c and the user wishes to find clusters in the stream based on a history of length h. Then, the macro-clustering algorithm will use some of the additive properties of the micro-clusters stored at snapshots t_c and $(t_c - h)$ in order to find the higher level clusters in a history or *time horizon* of length h. Of course, since it is not possible to store the snapshots at each and every moment in time, it is important to choose particular instants of time at which it is possible to store the state of the micro-clusters so that clusters in any user specified time horizon $(t_c - h, t_c)$ can be approximated. This was achieved in [14] with the use of the concept of a pyramidal time frame.

10.2.2.2 Pyramidal Time Frame

In this technique, the snapshots are stored at differing levels of granularity depending upon the recency. Snapshots are classified into different *orders* which can vary from 1 to $\log(T)$, where T is the clock time elapsed since the beginning of the stream. The order of a particular class of snapshots define the level of granularity in time at which the snapshots are maintained. The snapshots of different order are maintained as follows:

- Snapshots of the i-th order occur at time intervals of α^i, where α is an integer and $\alpha \geq 1$. Specifically, each snapshot of the i-th order is taken at a moment in time when the clock value is exactly divisible by α^i.
- At any given moment in time, only the last $\alpha^l + 1$ snapshots of order i are stored.

The above definition allows for considerable redundancy in storage of snapshots. For example, the clock time of 8 is divisible by 2^0, 2^1, 2^2, and 2^3 (where $\alpha = 2$). Therefore, the state of the micro-clusters at a clock time of 8 simultaneously corresponds to order 0, order 1, order 2 and order 3 snapshots. From an implementation point of view, a snapshot needs to be maintained only once. The following observations are true:

- For a data stream, the maximum order of any snapshot stored at T time units since the beginning of the stream mining process is $\log_\alpha(T)$.
- For a data stream the maximum number of snapshots maintained at T time units since the beginning of the stream mining process is $(\alpha^l + 1) \cdot \log_\alpha(T)$.
- For any user specified time window of h, at least one stored snapshot can be found within $(1 + 1/\alpha^{l-1})$ units of the current time.

While the first two results are quite easy to see, the last one needs to be proven formally [14]. We summarize this result as follows:

Lemma 10.2.2 *Let h be a user specified time horizon, t_c be the current time, and t_s be the time of the last stored snapshot of any order just before the time $t_c - h$. Then $t_c - t_s \leq (1 + 1/\alpha^{l-1}) \cdot h$.*

Order of Snapshots	Clock Times (Last 5 Snapshots)
0	55 54 53 52 51
1	54 52 50 48 46
2	52 48 44 40 36
3	48 40 32 24 16
4	48 32 16
5	32

TABLE 10.1: An example of snapshots stored for $\alpha = 2$ and $l = 2$

Proof: See [14].

For larger values of l, the time horizon can be approximated as closely as desired. For example, by choosing $l = 10$, it is possible to approximate any time horizon within 0.2%, while a total of only $(2^{10} + 1) \cdot \log_2(100 * 365 * 24 * 60 * 60) \approx 32343$ snapshots are required for 100 years. Since historical snapshots can be stored on disk and only the current snapshot needs to be maintained in main memory, this requirement is quite feasible from a practical point of view. It is also possible to specify the pyramidal time window in accordance with user preferences corresponding to particular moments in time such as beginning of calendar years, months, and days.

In order to clarify the way in which snapshots are stored, let us consider the case when the stream has been running starting at a clock-time of 1, and a use of $\alpha = 2$ and $l = 2$. Therefore $2^2 + 1 = 5$ snapshots of each order are stored. Then, at a clock time of 55, snapshots at the clock times illustrated in Table 10.1 are stored.

We note that a large number of snapshots are common among different orders. From an implementation point of view, the states of the micro-clusters at times of 16, 24, 32, 36, 40, 44, 46, 48, 50, 51, 52, 53, 54, and 55 are stored. It is easy to see that for more recent clock times, there is less distance between successive snapshots (better granularity). We also note that the storage requirements estimated in this section do not take this redundancy into account. Therefore, the requirements which have been presented so far are actually worst-case requirements.

10.2.2.3 Online Clustering with CluStream

The micro-clustering phase is the online statistical data collection portion of the algorithm. The aim is to maintain statistics at a sufficiently high level of (temporal and spatial) granularity so that it can be effectively used by the offline components such as horizon-specific macro-clustering as well as evolution analysis. The algorithm works in an iterative fashion, by always maintaining a current set of micro-clusters. It is assumed that a total of q micro-clusters are stored at any moment by the algorithm. We will denote these micro-clusters by $M_1 \ldots M_q$. Associated with each micro-cluster i, we create a unique *id* whenever it is first created. If two micro-clusters are merged (as will become evident from the details of our maintenance algorithm), a *list* of *id*s is created in order to identify the constituent micro-clusters. The value of q is determined by the amount of main memory available in order to store the micro-clusters. Therefore, typical values of q are significantly larger than the natural number of clusters in the data but are also significantly smaller than the number of data points arriving in a long period of time for a massive data stream. These micro-clusters represent the current snapshot of clusters which change over the course of the stream as new points arrive. Their status is stored away on disk whenever the clock time is divisible by α^i for any integer i. At the same time any micro-clusters of order r which were stored at a time in the past more remote than α^{l+r} units are deleted by the algorithm.

Whenever a new data point $\overline{X_{i_k}}$ arrives, the micro-clusters are updated in order to reflect the

changes. Each data point either needs to be absorbed by a micro-cluster, or it needs to be put in a cluster of its own. The first preference is to absorb the data point into a currently existing micro-cluster. The distance of each data point to the micro-cluster centroids $M_1 \ldots M_q$ is determined . The distance value of the data point $\overline{X_{i_k}}$ to the centroid of the micro-cluster M_j is denoted by $dist(M_j, \overline{X_{i_k}})$. Since the centroid of the micro-cluster is available in the cluster feature vector, this value can be computed relatively easily. This distance is used to compute the distance of the cluster M_p to the data point $\overline{X_{i_k}}$. In many cases, the point $\overline{X_{i_k}}$ does not naturally belong to the cluster M_p. These cases are as follows:

- The data point $\overline{X_{i_k}}$ corresponds to an outlier.
- The data point $\overline{X_{i_k}}$ corresponds to the beginning of a new cluster because of evolution of the data stream.

While the two cases above cannot be distinguished until more data points arrive, the data point $\overline{X_{i_k}}$ needs to be assigned a (new) micro-cluster of its own with a unique *id*. In order to make this decision, the cluster feature vector of M_p is used to decide if this data point falls within the *maximum boundary* of the micro-cluster M_p. If so, then the data point $\overline{X_{i_k}}$ is added to the micro-cluster M_p using the CF additivity property. The maximum boundary of the micro-cluster M_p is defined as a factor of t of the RMS deviation of the data points in M_p from the centroid. We define this as the *maximal boundary factor*. We note that the RMS deviation can only be defined for a cluster with more than one point. For a cluster with only one previous point, the maximum boundary is defined in a heuristic way. Specifically, it is chosen to be r times that of the next closest cluster.

If the data point does not lie within the maximum boundary of the nearest micro-cluster, then a new micro-cluster must be created containing the data point X_{i_k}. This newly created micro-cluster is assigned a new id which can identify it uniquely at any future stage of the data steam process. However, in order to create this new micro-cluster, the number of other clusters must be reduced by one in order to create memory space. This can be achieved by either deleting an old cluster or joining two of the old clusters. Our maintenance algorithm first determines if it is safe to delete any of the current micro-clusters as outliers. If not, then a merge of two micro-clusters is initiated.

The first step is to identify if any of the old micro-clusters are possibly outliers which can be safely deleted by the algorithm. While it might be tempting to simply pick the micro-cluster with the fewest number of points as the micro-cluster to be deleted, this may often lead to misleading results. In many cases, a given micro-cluster might correspond to a point of considerable cluster presence in the past history of the stream, but may no longer be an active cluster in the recent stream activity. Such a micro-cluster can be considered an outlier from the current point of view. An ideal goal would be to estimate the average timestamp of the last m arrivals in each micro-cluster, and delete the micro-cluster with the least recent timestamp. While the above estimation can be achieved by simply storing the last m points in each micro-cluster, this increases the memory requirements of a micro-cluster by a factor of m. Such a requirement reduces the number of micro-clusters that can be stored by the available memory and therefore reduces the effectiveness of the algorithm.

It is also needed to approximate the average time-stamp of the last m data points of the cluster M. This is achieved by using the data about the timestamps stored in the micro-cluster M. We note that the timestamp data allows the calculation of the mean and standard deviation[1] of the arrival times of points in a given micro-cluster M. Let these values be denoted by μM and σM respectively. Then, the time of arrival of the $m/(2 \cdot n)$-th percentile of the points in M is computed under the assumption that the time-stamps are normally distributed. This time-stamp is used as the approximate value of the recency. We shall call this value as the *relevance stamp* of cluster M. When the smallest such stamp of any micro-cluster is below a user-defined threshold δ, it can be

[1]The mean is equal to $CF1^t/n$. The standard deviation is equal to $\sqrt{CF2^t/n - (CF1^t/n)^2}$.

eliminated and a new micro-cluster can be created with a unique *id* corresponding to the newly arrived data point $\overline{X_{i_k}}$.

In some cases, none of the micro-clusters can be readily eliminated. This happens when all relevance stamps are sufficiently recent and lie above the user-defined threshold δ. In such a case, the closes micro-clusters are merged. The new micro-cluster no longer corresponds to one *id*. Instead, an *idlist* is created which is a union of the the *id*s in the individual micro-clusters. Thus, any micro-cluster which is result of one or more merging operations can be identified in terms of the individual micro-clusters merged into it.

While the above process of updating is executed at the arrival of each data point, an additional process is executed at each clock time which is divisible by α^i for any integer *i*. At each such time, the current set of micro-clusters are stored on disk, together with their id list, and indexed by their time of storage. The least recent snapshot of order *i* is deleted, if $\alpha^l + 1$ snapshots of such order had already been stored on disk, and if the clock time for this snapshot is not divisible by α^{i+1}. In the latter case, the snapshot continues to be a viable snapshot of order $(i+1)$. These micro-clusters can then be used to form higher level clusters or an evolution analysis of the data stream.

It should be pointed out that the micro-clustering model can be used in conjunction with fast indexing structures in order to allow *anytime* stream mining. This is particularly important in the context of data streams, since the stream speed is not known on an a-priori basis. A particular approach along this direction is the *ClusTree* method [46], which allows the adaptation of the granularity of the cluster model to the stream speed. The broader principle is that it is possible to follow the anytime paradigm to spend as much (or as little) time as dynamically available to digest new events.

While the use of snapshots is a natural way for examining the evolving stream at a variety of different granularities, other methods are possible for capturing the evolution of the data stream by incorporating decay into the micro-clusters [15], or by using sliding windows in conjunction with an exponential histogram of the temporal cluster feature vector [65]. These methods are generally preferable, if the level of evolution in the data stream is known in advance. These different methods have differing tradeoffs between memory requirements and flexibility, but their goals are similar in terms of capturing the evolution of the data stream.

10.3 Density-based Stream Clustering

Density-based methods [18, 30] construct a density profile of the data for clustering purposes. Typically, kernel density estimation methods [58] are used in order to construct a smooth density profile of the underlying data. Subsequently, the data is separated out into density-connected regions. These density connected regions may be of different shapes and sizes. One of the advantages of density-based algorithms is that an implicit shape is not assumed for the clusters. For example, when euclidian distance functions are used, it is always assumed that the clusters have spherical shapes. Similarly, the Manhattan metric assumes that the clusters are of a diamond shape. In density-based clustering, connected regions of high density may often have arbitrary shapes. Another aspect of density based clustering is that it does not pre-decide the number of clusters. Rather, a threshold on the density is used in order to determine the connected regions. Of course, this changes the nature of the parameter which needs to be presented to the algorithm (density threshold instead of number of clusters), but it does not necessarily make the approach parameter-free.

The main challenge in the stream scenario is to construct density-based algorithms which can be efficiently executed in a single pass of the data, since the process of density estimation may be computationally intensive. There are two broad classes of techniques:

- The first class of techniques extends the micro-clustering technique to this case, by relaxing the constraint on the number of micro-clusters, and imposing a constraint on the radius and "weight" of each micro-cluster. The dense regions are generated by connecting together the dense micro-clusters which satisfy a condition on connectivity similar to that in [30].
- The second class of techniques divides the data space into grids, and then determines the dense grids. The dense regions in the data are re-constructed by piecing together the connected dense grids.

We will discuss both these classes of techniques in this section.

10.3.1 DenStream: Density-based Micro-clustering

The *DenStream* algorithm [24] approach combines micro-clustering with a density-estimation process for effective clustering. The first step is to define a *core object*, which is defined as an object, in the ε-neighborhood of which the weight of the data points is at least μ. A *density area* is defined as the union of the ε neighborhoods of the core objects.

In the context of streaming data, it is difficult to determine these dense regions naturally. Therefore, dense regions of smaller granularity are defined in the form of core micro-clusters. A core micro-cluster is defined a set of data points with weight at least μ and for which the radius of the micro-cluster about its center is less than ε. We note that the weight of a data point is based on a decay weighted function of the time that it last arrived. Therefore, if δt be the time since a data point arrived, its weight is given by:

$$f(\delta t) = 2^{-\delta t} \tag{10.3}$$

Since the radius of the micro-cluster is constrained to be less than ε, it implies that the number of micro-clusters is much larger than the number of natural clusters in the data for small values of ε. At the same time, the number of core micro-clusters is much smaller than the number of points in the data stream, since each cluster contains a weight of at least μ. We note that the key difference from the standard micro-clustering definition is that the number of micro-cluster are not constrained, though the radius of each micro-cluster is constrained. Thus, this approach approaches a different parameter set to the underlying application. The core micro-cluster is also referred to as a c-micro-cluster.

One immediate observation is that when a micro-cluster is first formed by such an algorithm, it is unlikely to contain the requisite weight of data points required to be defined as a micro-cluster. Therefore, the concepts of *potential core micro-cluster* and *outlier micro-cluster* are defined in [24]. In the former case, the micro-cluster contains a weight of at least $\beta \cdot \mu$ (for some $\beta \in (0,1)$), and in the latter case, it contains a weight less than $\beta \cdot \mu$. Furthermore, since the weights of points decay over time, a cluster may also change from being a p-micro-cluster to an o-micro-cluster, if sufficient data points are not added to it in order to compensate for the decay. Thus, during its lifecycle, a micro-cluster may move from being an outlier micro-cluster to being a potential core micro-cluster, and finally to the stage of being a core micro-cluster. These two kinds of micro-clusters are referred to as p-micro-clusters and o-micro-clusters respectively.

When a new data point arrives, the following can be the possibilities in terms of what is done with it:

- The first step is to try to insert it into a p-micro-cluster, as long as it is possible to do so, without violating the radius constraint.
- If the first step is not possible, the next step is to try to insert it into an o-micro-cluster, as long as this can be done without violating the radius constraint.
- If the first and second steps are not possible, then a new o-micro-cluster is created containing this data point.

One challenge of this approach is that the number of o-micro-clusters will increase with time, as new o-micro-clusters are created, and some of the p-micro-clusters decay back to o-micro-clusters. Therefore, from time to time, we purge some of the o-micro-clusters, which will low potential for becoming p-micro-clusters. The larger the time δt that has elapsed since the creation of an o-micro-cluster, the larger is weight is expected to be. Therefore, every T_p time periods, we prune all those micro-clusters, whose weight is less than the threshold $\psi(\delta t)$, where:

$$\psi(\delta t) = \frac{2^{-\lambda \cdot (\delta t + T_p)} - 1}{2^{-\lambda \cdot (T_p)} - 1} \tag{10.4}$$

This process continues in order to maintain the micro-clusters dynamically. We note that the individual micro-clusters can be re-constructed into density-connected regions in order to create the final set of clusters of arbitrary shape. The overall approach for creating the clusters of arbitrary shape is discussed in detail in in [24].

10.3.2 Grid-based Streaming Algorithms

Grid-based methods are a class of density-based streaming algorithms, in which a grid structure is used in order to quantify the density at each point in the data. The core idea is that the data is discretized into ranges along each dimension, and this also results in dividing the data into cells along different dimensions. The number of points in each cell defines the density of that cell. Then, the dense cells in the data can be aggregated in order to determine the dense regions for the clustering.

10.3.2.1 D-Stream Algorithm

A method called D-Stream for real-time density based clustering of streaming data was proposed in [25]. The algorithm has many similarities with [30] in terms of trying to determine fine grained regions of high density. The main difference at the conceptual level is that this is done with the use of grids rather than micro-clusters. As in the previous case, a decay function $f(\delta t(\overline{X}))$ is used to denote the weight of a point $\overline{X}$ since the time of its arrival $\delta t(\overline{X}, t_c)$ units ago from the current time t_c:

$$f(\delta t(\overline{X}, t_c)) = \mu^{-\delta t(\overline{X}, t_c)} \tag{10.5}$$

Here, we assume that $\mu > 1$, which is slightly different from the notations in [25]. We note that this decay function is identical to that proposed in [15, 30], by defining the relationship with respect to the parameter λ and assuming that $\mu = 2^{\lambda}$. Under the assumption of [25] that exactly one record arrives at each time-stamp, it can be shown that the sum of the weights of all data points is no larger than $\mu/(\mu - 1)$.

The grids are defined by discretizing these ranges along each dimension. For the i-th dimension, the discretization is performed into p_i different ranges along the ith dimension. This dicretization naturally defines a total of $\eta = \prod_i p_i$ d-dimensional cells. For each cell S, its weight $W(S, t_c)$ is defined as the current time t_c as follows:

$$W(S, t_c) = \sum_{\overline{X} \in S} f(\delta t(\overline{X}), t_c) \tag{10.6}$$

We note that the grid is essentially analogous to the radius constrained micro-cluster defined in the *DenStream* algorithm. Thus, as in the case of *DenStream*, the density of a grid is constantly changing over time. However, it is never necessary to update any decay-based statistics either in grids or micro-clusters at each time instant [15, 25]. This is because all grids decay at the same proportional rate, and the update can be lazily performed only when the density value in the grid is updated with the addition of a new data point.

The next step is to define what it means for a grid to be dense. Since the total density over

all grids is no larger than $\mu/(\mu-1)$, it follows that the *average* density of each grid is no larger than $\mu/(\eta \cdot (\mu-1))$. Therefore, a grid is defined as *dense*, when its density is a constant times larger than this factor. This is essentially analogous to the concept of a *c*-micro-cluster defined in [30]. Analogous to the concept of an o-micro-cluster, and a p-micro-cluster, the work in [25] divides the non-dense grid-cells into *sparse grid cells* and *transitional grid cells*, with the use of a smaller threshold on the density. Grid cells can change between the different states of being sparse, transitional or dense, both because of the addition of new data points, and also because of decay.

One observation is that the number of grid-cells $\eta = \prod_{i=1}^{d} p_i$ is exponentially dependent upon the dimensionality d. However, in practice, most of these grid-cells are empty, and the information for empty grids need not be stored. This may not be sufficient in many real applications, where many outlier points may sporadically appear in the data. The work in [25] designs methods for identifying and removing such sporadic grids from the data. The maintained dense grids can then be consolidated into larger dense regions in the data. As in the case of [30], this is defined in the form of density-connected grids, where adjacency of two dense grids is treated as a density-connection. For the precise details of the dense region construction, we refer the reader to [25]. Thus, it is evident that grid-based and micro-cluster based density clusters share a number of conceptual similarities ar various stages of the algorithm.

One weakness of the approach is that a significant number of non-empty grid cells need to be discarded in order to keep the memory requirements in check. In many cases, such grid-cells occur at the borders of the clusters. The discarding of such cells may lead to degradation in cluster quality. Therefore, a method has been proposed in [43] to design a variation of the *D-Stream* method (known as *DD-Stream*), which includes the data points at the borders into adjacent denser grids, which are retained by the algorithm. It has been shown that such an approach leads to some improvements in cluster quality.

10.3.2.2 Other Grid-based Algorithms

The method in [35] updates a full-dimensional grid of the incoming data stream. The clusters are discovered from the updated density values in this grid. At any given time, the density values in the grid can be used in order to determine the updated and most effective clusters.

An important point to be kept in mind is that grid-based algorithms require the discretization of the data along the different dimensions in order to create the grid cells. The granularity of the discretization is a critical design choice at the very beginning of the algorithm. Unfortunately, at the beginning of the stream arrival, very little may be known about how the underlying data points are distributed, and therefore it is difficult to choose the level of discretization along each dimension in an optimal way. Furthermore, the appropriate level of discretization for each data locality may be a different, and a single global level of discretization may be suboptimal over different data localities.

Therefore, the work in [55] uses a dynamic approach called *Statsgrid* for the discretization process, wherein the data cells are recursively partitioned based on their local density. The algorithm starts off with cells of equal size. As data points are added to the cells, and the number of points in a cell becomes sufficiently large, the algorithm partitions the cell into two along one of the dimensions. This process can of course be repeated recursively each time any of the children cells become dense. At some point, the maximum level of allowed granularity is reached, and such a cell is called a *unit cell*, which cannot be further divided. We note that this approach naturally leads to a hierarchical clustering of the data, which can be very useful in many applications. Nevertheless, the work does not use temporal decay, and therefore does not adjust very well to an evolving data stream.

This method has therefore been extended to the *CellTree* method [56], which allows decay in the statistics. It explicitly uses a *CellTree* data structure in order to maintain the hierarchical relationships among the grid cells. Furthermore, when the data cells decay with time, it may be possible

to merge adjacent cells. Therefore, the method in [56] provides greater flexibility, than discussed in the original algorithm.

10.4 Probabilistic Streaming Algorithms

One of the common methods for probabilistic clustering is that of *mixture modeling*, in which the data is assumed to be generated by a mixture of known distributions such as the Gaussian distribution. The parameters of this distribution are then typically learned with an EM algorithm from the actual data records [59]. The main argument in [59] is that probability density-based clustering algorithms are much more efficient than applying EM on the entire data set. On the other hand, it has been shown that it is possible to use the EM-algorithm in order to provide an efficient update process for newly arriving data. Nevertheless, since the EM algorithm requires one to learn a large number of parameters, such an approach is unlikely to be effective, when the underlying data is evolving rapidly.

Another area in which probabilistic models are commonly used for clustering, are for the case of text. A common technique which is used to create a soft clustering of the data for the case of text, is that of *topic modeling* [41]. In this technique, soft clusters are associated with the data in which words and documents are probabilistically assigned to the different partitions. Since the topic modeling approach uses an EM-algorithm, it can sometimes be slow in practice. Therefore, a method has been proposed in [20] for topic modeling over text streams. The work in [20] proposes online variants of three common batch algorithms for topic modeling. These correspond to the Latent Dirichlet Allocation (LDA) [22], Dirichlet Compound Multinomial (DCM) mixtures [29] and von-Mises Fisher (vMF) mixture models [21]. It was shown in [20] that the online variant of the vMF approach provides the best results. A detailed study of these topic modeling algorithms is beyond the scope of this survey. Interested readers are referred to [20].

10.5 Clustering High Dimensional Streams

In many circumstances, data stream are very high-dimensional, because a large number of features are available for the mining process. The high-dimensional case presents a special challenge to clustering algorithms even in the traditional domain of static data sets. This is because of the sparsity of the data in the high-dimensional case. In high-dimensional space, all pairs of points tend to be almost equidistant from one another. As a result, it is often unrealistic to define distance-based clusters in a meaningful way. Some recent work on high-dimensional data uses techniques for *projected clustering* which can determine clusters for a specific subset of dimensions [3, 17]. In these methods, the definitions of the clusters are such that each cluster is specific to a particular group of dimensions. This alleviates the sparsity problem in high-dimensional space to some extent. Even though a cluster may not be meaningfully defined on all the dimensions because of the sparsity of the data, some subset of the dimensions can always be found on which particular subsets of points form high quality and meaningful clusters. Of course, these subsets of dimensions may vary over the different clusters. Such clusters are referred to as *projected clusters* [3].

10.5.1 The HPSTREAM Method

The micro-clustering method can also be extended to the case of high dimensional projected stream clustering . The algorithms is referred to as *HPSTREAM*. In [15, 16], methods have been proposed for high dimensional projected clustering of data streams. The basic idea is to use an (incremental) algorithm in which we associate a set of dimensions with each cluster. This corresponds to the standard micro-clustering method as discussed in [15], with the main difference that the distances from data points to clusters are computed on the basis of dimension-specific clusters. Therefore, additional information needs to be associated with a cluster in order to capture the set of dimensions associated with them. The set of dimensions is represented as a d-dimensional bit vector $B(C_i)$ for each cluster structure in FCS. This bit vector contains a 1 bit for each dimension which is included in cluster C_i. In addition, the maximum number of clusters k and the average cluster dimensionality l is used as an input parameter. The average cluster dimensionality l represents the average number of dimensions used in the cluster projection. An iterative approach is used in which the dimensions are used to update the clusters and vice-versa. The structure in FCS uses a decay-based mechanism in order to adjust for evolution in the underlying data stream. For a data point, which arrived δt units ago, its weight is assumed to be $2^{-\lambda \cdot \delta t}$, where λ is the decay rate.

Therefore, the micro-clusters for the *HPSTREAM* algorithm contain decay-based statistics, wherein the micro-clusters are similar to the *CluStream* algorithm. Furthermore, the overall framework of the *HPSTREAM* algorithm is quite similar, because data points are assigned to micro-clusters on the basis of their projected distances. The main difference is that each component of the additive micro-cluster statistics is a decay-based weight. In addition, the bit vector corresponding to the choice of dimensions is stored with the micro-cluster statistics. Clearly, a number of changes need to be incorporated into the *CluStream* approach in order to account for these changes:

- We note that the decay-based statistics ensure that the weights of the micro-clusters change in each time stamp. However, it is not necessary to update the statistics at each time-stamp, since all the micro-clusters decay at the same rate. Rather, we perform the update only when a new point is inserted into the data. When a new point is inserted, and δx is the time interval since the last time, then all micro-cluster statistics are multiplied by $2^{-\lambda \cdot \delta x}$ before adding a data point to the micro-cluster statistics.

- The average distance to each cluster is now computed on the basis of the projected dimensions specific to that cluster. The bit vector in the micro-cluster statistics is used in order to decide on the exact set of dimensions to use.

- The projected dimensions in the different clusters are updated periodically, so that the most compact dimensions are retained for each cluster. The corresponding bit vector in the micro-cluster statistics is updated.

It has been shown in [15], that the incorporation of projections can significantly improve the effectiveness of the approach. Details are discussed in [15].

10.5.2 Other High-Dimensional Streaming Algorithms

A variety of other high dimensional streaming clustering algorithms have been proposed after the first *HPSTREAM* framework. In particular, a grid-based algorithm was proposed in [49] for high-dimensional projected clustering of data streams. High-dimensional projected clustering has also been applied to other domains such as uncertain data. For example, methods for high dimensional projected clustering of uncertain data streams have been proposed in [8, 60].

Most of the methods discussed in the literature use a k-means type approach, which fixes the number of clusters in the data. Furthermore, a k-means type approach also makes implicit assumptions about the *shapes* of the underlying clusters. The work in [52] proposes a density-based method

for high-dimensional stream clustering. Such an approach has the virtue of recognizing the fact that the number and shape of the clusters in the stream may vary over time. The work in [52] proposes HDDSTREAM, which is a method for high-dimensional stream clustering. It generalizes the density-based approach proposed in [30] in order to incorporate subspace analysis in the clustering process. Since the method in [30] was originally designed to handle variations in the number of clusters in the stream, as well as different shapes of clusters, these virtues are inherited by the HDDSTREAM method of [52] as well.

A number of methods have also been proposed for high dimensional projected clustering of dynamic data [47, 62]. While these methods can be effectively used for dynamic data, they do require access to the raw data for clustering purposes. Therefore, these methods are not streaming techniques in the strict sense.

10.6 Clustering Discrete and Categorical Streams

Many data streams are defined on a domain of *discrete values* in which the attributes are unordered and take on one of a very large number of possible discrete values. In such cases, the cluster feature vector of the micro-clustering approach does not represent the values in the underlying data well. Therefore, methods are required in order to perform stream clustering algorithms in such scenarios. The simplest case of categorical stream clustering is that of *binary data* in which the data takes on values from $\{0,1\}$. Binary data can be considered both quantitative and symbolic, and therefore almost all stream algorithms can be used directly for this case.

10.6.1 Clustering Binary Data Streams with k-Means

The simplest case of categorical data is binary data, in which the discrete attributes may take on only one of two possible values. Binary data is also a special case of quantitative data, because an ordering can always be assigned to discrete values as long as there are only two of them. Therefore, virtually all of the streaming algorithms can be used for binary data. Nevertheless, it can sometimes be useful to leverage a method which is specifically designed for the case of binary data.

An algorithm for utilizing optimizations of k-means algorithms for data streams is proposed in [54]. The main observation in [54] is that the binary transactions are often *sparse*. This can be used in order to greatly speed up the distance computations. Since distance computations form the bottleneck operation for such algorithms, a speed up of the distance computations also greatly speeds up the underlying algorithm. From a practical point of view, this means that only a small fraction of the features in the transaction take on the 1-value. Therefore, the general approach used in [54] is that first the distance of the null transaction to each of the centroids is computed. Subsequently, for each position in the transaction, the effect of that position on the distances is computed. Since many distance functions are separable functions across different dimensions, this can be achieved quite effectively. It has been shown in [54] that these speedups can be implemented very effectively at no reduction in quality of the underlying results.

10.6.2 The StreamCluCD Algorithm

The *Squeezer* algorithm was a one-pass algorithm for clustering categorical data [39]. The *StreamCluCD* approach [40] is the streaming extension of this framework. The essential idea behind the algorithm is that when a data point comes in, it is placed into a cluster of its own. For subsequent incoming points, we compute their similarity to the existing clusters in the data. If the

incoming points are sufficiently similar to one of the existing clusters, then they are placed in that cluster. Otherwise, the incoming data point is placed in a cluster of its own. A key operation in the *StreamCluCD* algorithm is to maintain the frequency counts of the attribute values in each cluster. The lossy counting approach introduced in [50] is used for this purpose. The motivation for this is to reduce the memory footprint for maintaining the frequency counts. The issue of memory requirements becomes especially important, when the number of possible discrete values of each attribute increases. This is referred to as the *massive domain* scenario, and will be discussed in the next section.

10.6.3 Massive-Domain Clustering

Massive-domains are those data domains in which the number of possible values for one or more attributes is very large. Examples of such domains are as follows:

- In network applications, many attributes such as IP-addresses are drawn over millions of possibilities. In a multi-dimensional application, this problem is further magnified because of the multiplication of possibilities over different attributes.
- Typical credit-card transactions can be drawn from a universe of millions of different possibilities depending upon the nature of the transactions.
- Supermarket transactions are often drawn from a universe of millions of possibilities. In such cases, the determination of patterns which indicate different kinds of classification behavior may become infeasible from a space- and computational efficiency perspective.

The massive domain size of the underlying data restricts the *computational approach* which may be used for discriminatory analysis. Thus, this problem is significantly more difficult than the standard clustering problem in data streams. Space-efficiency is a special concern in the case of data streams, because it is desirable to hold most of the data structures in main memory in order to maximize the processing rate of the underlying data. Smaller space requirements ensure that it may be possible to hold most of the intermediate data in fast caches, which can further improve the efficiency of the approach. Furthermore, it may often be desirable to implement stream clustering algorithms in a wide variety of space-constrained architectures such as mobile devices, sensor hardware, or cell processors. Such architectures present special challenges to the massive-domain case, if the underlying algorithms are not space-efficient.

The problem of clustering can be extremely challenging from a space and time perspective in the massive-domain case. This is because one needs to retain the discriminatory characteristics of the most relevant clusters in the data. In the massive-domain case, this may entail storing the frequency statistics of a large number of possible attribute values. While this may be difficult to do explicitly, the problem is further intensified by the large volume of the data stream which prevents easy determination of the importance of different attribute-values. The work in [4] proposes a sketch-based approach in order to keep track of the intermediate statistics of the underlying clusters. These statistics are used in order to make approximate determinations of the assignment of data points to clusters. A number of probabilistic results are provided in [4], which indicate that these approximations are sufficiently accurate to provide similar results to an infinite-space clustering algorithm with high probability.

The data stream D contains d-dimensional records denoted by $\overline{X_1} \ldots \overline{X_N} \ldots$. The attributes of record $\overline{X_i}$ are denoted by $(x_i^1 \ldots x_i^d)$. It is assumed that the attribute value x_i^k is drawn from the unordered domain set $J_k = \{v_1^k \ldots v_{M^k}^k\}$. The value of M^k denotes the domain size for the kth attribute. The value of M^k can be very large, and may range in the order of millions or billions. From the point of view of a clustering application, this creates a number of challenges, since it is no longer possible to hold the cluster statistics in a space-limited scenario.

```
Algorithm CSketch(Labeled Data Stream: D,
                  NumClusters: k)
begin
  Create k sketch tables of size w·h each;
  Initialize k sketch tables to null counts;
  repeat
    Receive next data point X from D;
    Compute the approximate dot product of incoming data
     point with each cluster-centroid with the use of the
     sketch tables;
    Pick the centroid with the largest approximate
      dot product to incoming point;
    Increment the sketch counts in the chosen table
     for all d dimensional value strings;
  until(all points in D have been processed);
end
```

FIGURE 10.1: The Sketch-based Clustering Algorithm (CSketch Algorithm)

The work in [4] uses the count-min sketch [26] for the problem of clustering massive-domain data streams. In the count-min sketch, a hashing approach is utilized in order to keep track of the attribute-value statistics in the underlying data. We use $w = \lceil \ln(1/\delta) \rceil$ pairwise independent hash functions, each of which map onto uniformly random integers in the range $h = [0, e/\varepsilon]$, where e is the base of the natural logarithm. The data structure itself consists of a two dimensional array with $w \cdot h$ cells with a length of h and width of w. Each hash function corresponds to one of w 1-dimensional arrays with h cells each. In standard applications of the count-min sketch, the hash functions are used in order to update the counts of the different cells in this 2-dimensional data structure. For example, consider a 1-dimensional data stream with elements drawn from a massive set of domain values. When a new element of the data stream is received, we apply each of the w hash functions to map onto a number in $[0 \ldots h-1]$. The count of each of the set of w cells is incremented by 1. In order to *estimate* the count of an item, we determine the set of w cells to which each of the w hash-functions map, and compute the minimum value among all these cells. Let c_t be the true value of the count being estimated. We note that the estimated count is at least equal to c_t, since we are dealing with non-negative counts only, and there may be an over-estimation because of collisions among hash cells. As it turns out, a probabilistic upper bound to the estimate may also be determined. It has been shown in [26], that for a data stream with T arrivals, the estimate is at most $c_t + \varepsilon \cdot T$ with probability at least $1 - \delta$.

The *CSketch* algorithm uses the number of clusters k and the data stream D as input to the algorithm. The clustering algorithm is partition-based, and assigns incoming data points to the most similar cluster centroid. The frequency counts for the different attribute values in the cluster centroids are incremented with the use of the sketch table. These frequency counts can be maintained only approximately because of the massive domain size of the underlying attributes in the data stream. Similarity is measured with the computation of the dot-product function between the incoming point and the centroid of the different clusters. This computation can be performed only approximately in the massive-domain case, since the frequency counts for the values in the different dimensions cannot be maintained explicitly. For each cluster, we maintain the frequency sketches of the records which are assigned to it. Specifically, for each cluster, the algorithm maintains a separate sketch table containing the counts of the values in the incoming records. The same hash function is used for each table. The algorithm starts off by initializing the counts in each sketch table to 0.

Subsequently, for each incoming record, we will update the counts of each of the cluster-specific hash tables. In order to determine the assignments of data points to clusters, the dot products of the d-dimensional incoming records to the frequency statistics of the values in the different clusters are computed. This is sometimes not possible to do explicitly, since the precise frequency statistics are not maintained. Let $q_r^j(x_i^r)$ represents the frequency of the value x_i^r in the j-th cluster. Let m_i be the number of data points assigned to the j-th cluster. Then, the d-dimensional statistics of the record $(x_i^1 \ldots x_i^d)$ for the j-th cluster is given by $(q_1^j(x_i^1) \ldots q_d^j(x_i^d))$. Then, the frequency-based dot-product $D^j(\overline{X_i})$ of the incoming record with statistics of cluster j is given by the dot product of the fractional frequencies $(q_1^j(x_i^1)/m_j \ldots q_d^j(x_i^d)/m_j)$ of the attribute values $(x_i^1 \ldots x_i^d)$ with the frequencies of these same attribute values within record $\overline{X_i}$. We note that the frequencies of the attribute values with the record $\overline{X_i}$ are unit values corresponding to $(1, \ldots 1)$. Therefore, the corresponding dot product is the following:

$$D^j(\overline{X_i}) = \sum_{r=1}^{d} q_r^j(x_i^r)/m_j \tag{10.7}$$

The incoming record is assigned to the cluster for which the estimated dot product is the largest. We note that the value of $q_r^j(x_i^r)$ cannot be known exactly, but can only be estimated approximately because of the massive-domain constraint. There are two key steps which use the sketch table during the clustering process:

- Updating the sketch-table and other required statistics for the corresponding cluster for each incoming record.
- Comparing the similarity of the incoming record to the different clusters with the use of the corresponding sketch tables.

First, we discuss the process of updating the sketch table, once a particular cluster has been identified for assignment. For each record, the sketch-table entries corresponding to the attribute values on the different dimensions are incremented. For each incoming record $\overline{X_i}$, the w hash functions are applied to the strings corresponding to the attribute values in it. Let m be the index of the cluster to which the data point is assigned. Then, exactly $d \cdot w$ entries in the sketch table for cluster m are updated by applying the w hash functions to each of the d strings which are denoted by $x_i^1 \oplus 1 \ldots x_i^d \oplus d$. The corresponding entries are incremented by one unit each.

In order to pick a cluster for assignment, the approximate dot-products across different clusters need to be computed. The record is converted to its sketch representation by applying the w hash functions to each of these d different attribute values. We retrieve the corresponding d sketch-table entries for each of the w hash functions and each cluster. For each of the w hash-functions for the sketch table for cluster j, the d counts are simply estimates of the value of $q_1^j(x_i^1) \ldots q_d^j(x_i^d)$. Specifically, let the count for the entry picked by the lth hash function corresponding to the rth dimension of record $\overline{X_i}$ in the sketch table for cluster j be denoted by c_{ijlr}. Then, we estimate the dot product D_{ij} between the record $\overline{X_i}$ and the frequency statistics for cluster j as follows:

$$D_{ij} = \min_l \sum_{r=1}^{d} c_{ijlr}/m_j \tag{10.8}$$

The value of D_{ij} is computed over all clusters j, and the cluster with the largest dot product to the record $\overline{X_i}$ is picked for assignment.

It has been shown in [4] that this assignment process approximates an infinite space clustering algorithm quite well. This is characterized in terms of *assignment errors* because of the approximation process. An important point to be kept in mind is that in many cases an incoming data point may match well with many of the clusters. Since similarity functions such as the dot product are heuristically defined, small errors in ordering (when b_q is small) are not very significant for

the clustering process. Similarly, some of the clusters correspond to outlier points and are sparsely populated. Therefore, it is more important to ensure that assignments to "significant clusters" (for which f_p is above a given threshold) are correct. Therefore, the work in [4] defines the concept of an (f,b)-significant assignment error as follows:

Definition 10.6.1 *An assignment error which results from the estimation of dot products is said to be (f,b)-significant, if a data point is incorrectly assigned to cluster index p which contains at least a fraction f of the points, and the correct cluster for assignment has index q which satisfies the following relationship:*

$$D_{ip} \geq D_{iq} + b \tag{10.9}$$

It has been shown in [4] that it is possible to bound the probability of an (f,b)-significant error in a given assignment.

Lemma 10.6.1 *The probability of an (f,b)-significant error in an assignment is at most equal to $k \cdot (d^2/(b \cdot f \cdot h))^w$.*

This can be used to further bound the probability that there is no (f,b)-significant error over the entire course of the clustering process of N data points.

Lemma 10.6.2 *Let us assume that $N \cdot k \cdot (d^2/(b \cdot f \cdot h))^w < 1$. The probability that there is at least one (f,b)-significant error in the clustering process of N data points is given by at most $\frac{N \cdot k}{(b \cdot f \cdot h/d^2)^w}$.*

In addition, the experimental results in [4] show that the clustering process can be replicated almost exactly in practice with the use of this approximation process. Thus, the work in [4] proposes a fast and space-efficient method for clustering massive-domain data streams.

10.7 Text Stream Clustering

The problem of streaming text clustering is particularly challenging in the context of text data because of the fact that the clusters need to be continuously maintained in real time. One of the earliest methods for streaming text clustering was proposed in [64]. This technique is referred to as the *Online Spherical k-Means Algorithm (OSKM)*, which reflects the broad approach used by the methodology. This technique divides up the incoming stream into small segments, each of which can be processed effectively in main memory. A set of *k*-means iterations are applied to each such data segment in order to cluster them. The advantage of using a segment-wise approach for clustering is that since each segment can be held in main memory, we can process each data point multiple times as long as it is held in main memory. In addition, the centroids from the previous segment are used in the next iteration for clustering purposes. A decay factor is introduced in order to age-out the old documents, so that the new documents are considered more important from a clustering perspective. This approach has been shown to be extremely effective in clustering massive text streams in [64].

A different method for clustering massive text and categorical data streams is discussed in [6]. The method discussed in [6] uses an approach which examines the relationship between outliers, emerging trends, and clusters in the underlying data. 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 crawl 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. The approach discussed in [6] recognizes new clusters by first recognizing them as outliers. This approach works with the use of a summarization methodology, in which we use the concept of *condensed droplets* [6] in order to create concise representations of the underlying clusters.

As in the case of the OSKM algorithm, we ensure that recent data points are given greater importance than older data points. This is achieved by creating a time-sensitive weight for 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. 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.

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.

The statistics of the data points are captured in summary statistics, which are referred to as *condensed droplets*. These represent the word distributions within a cluster, and can be used in order to compute the similarity of an incoming data point to the cluster. The overall algorithm proceeds as follows. 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. In the case of text data sets, the cosine similarity measure between $\overline{DF1}$ and $\overline{X}$ is used. The similarity value $S(\overline{X}, C_j)$ is computed from the incoming document $\overline{X}$ to every cluster C_j. The cluster with the maximum value of $S(\overline{X}, C_j)$ is chosen as the relevant cluster for data insertion. Let us assume that this cluster is C_{mindex}. We use a threshold denoted by *thresh* in order to determine whether the incoming data point is an outlier. If the value of $S(\overline{X}, C_{mindex})$ is larger than the threshold *thresh*, then the point $\overline{X}$ is assigned to the cluster C_{mindex}. Otherwise, we check if some inactive cluster exists in the current set of cluster droplets. If no such inactive cluster exists, then the data point $\overline{X}$ is added to C_{mindex}. On the other hand, when an inactive cluster does exist, a new cluster is created containing the solitary data point $\overline{X}$. This newly created cluster replaces the inactive cluster. We note that this new 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 C_{mindex}, we update the statistics of the cluster in order to reflect the insertion of the data point and temporal decay statistics. Otherwise, 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 among all inactive clusters. This process is continuously performed over the life of the data stream, as new documents arrive over time. The work in [6] also presents a variety of other applications of the stream clustering technique such as evolution and correlation analysis.

A different way of utilizing the temporal evolution of text documents in the clustering process is described in [38]. Specifically, the work in [38] uses *bursty features* as markers of new topic occurrences in the data stream. This is because the semantics of an up-and-coming topic are often reflected in the frequent presence of a few distinctive words in the text stream. At a given period in time, the nature of relevant topics could lead to bursts in specific features of the data stream. Clearly, such features are extremely important from a clustering perspective. Therefore, the method discussed in [38] uses a new representation, which is referred to as the *bursty feature representation* for mining text streams. In this representation, a time-varying weight is associated with the features depending upon its burstiness. This also reflects the varying importance of the feature to the clustering process. Thus, it is important to remember that a particular document representation is dependent upon the particular instant in time at which it is constructed.

Another issue which is handled effectively in this approach is an implicit reduction in dimensionality of the underlying collection. Text is inherently a high dimensional data domain, and the pre-selection of some of the features on the basis of their burstiness can be a natural way to reduce the dimensionality of document representation. This can help in both the effectiveness and efficiency of the underlying algorithm.

The first step in the process is to identify the bursty features in the data stream. In order to achieve this goal, the approach uses Kleinberg's 2-state finite automaton model [45]. Once these features have been identified, the bursty features are associated with weights which depend upon their level of burstiness. Subsequently, a bursty feature representation is defined in order to reflect the underlying weight of the feature. Both the identification and the weight of the bursty feature are dependent upon its underlying frequency. A standard *k*-means approach is applied to the new representation in order to construct the clustering. It was shown in [38] that the approach of using burstiness improves the cluster quality. Once criticism of the work in [38] is that it is mostly focused on the issue of improving effectiveness with the use of temporal characteristics of the data stream, and does not address the issue of efficient clustering of the underlying data stream.

In general, it is evident that feature extraction is important for all clustering algorithms. While the work in [38] focuses on using temporal characteristics of the stream for feature extraction, the work in [48] focuses on using *phrase extraction* for effective feature selection. This work is also related to the concept of topic-modeling, which will be discussed in detail in the next section. This is because the different topics in a collection can be related to the clusters in a collection. The work in [48] uses topic-modeling techniques for clustering. The core idea in the work of [48] is that individual words are not very effective for a clustering algorithm because they miss the context in which the word is used. For example, the word "star" may either refer to a celestial body or to an entertainer. On the other hand, when the phrase "fixed star" is used, it becomes evident that the word "star" refers to a celestial body. The phrases which are extracted from the collection are also referred to as *topic signatures*.

The use of such phrasal clarification for improving the quality of the clustering is referred to as *semantic smoothing* because it reduces the noise which is associated with semantic ambiguity. Therefore, a key part of the approach is to extract phrases from the underlying data stream. After phrase extraction, the training process determines a translation probability of the phrase to terms in the vocabulary. For example, the word "planet" may have high probability of association with the phrase "fixed star", because both refer to celestial bodies. Therefore, for a given document, a

rational probability count may also be assigned to all terms. For each document, it is assumed that all terms in it are generated either by a topic-signature model, or a background collection model.

The approach in [48] works by modeling the soft probability $p(w|C_j)$ for word w and cluster C_j. The probability $p(w|C_j)$ is modeled as a linear combination of two factors; (a) A maximum likelihood model which computes the probabilities of generating specific words for each cluster (b) An indirect (translated) word-membership probability which first determines the maximum likelihood probability for each topic-signature, and then multiplying with the conditional probability of each word, given the topic-signature. We note that we can use $p(w|C_j)$ in order to estimate $p(d|C_j)$ by using the product of the constituent words in the document. For this purpose, we use the frequency $f(w,d)$ of word w in document d.

$$p(d|C_j) = \prod_{w \in d} p(w|C_j)^{f(w,d)} \tag{10.10}$$

We note that in the static case, it is also possible to add a background model in order to improve the robustness of the estimation process. This is however not possible in a data stream because of the fact that the background collection model may require multiple passes in order to build effectively. The work in [48] maintains these probabilities in online fashion with the use of a *cluster profile*, that weights the probabilities with the use of a fading function. We note that the concept of cluster profile is analogous to the concept of condensed droplet introduced in [6]. The key algorithm (denoted by OCTS) is to maintain a dynamic set of clusters into which documents are progressively assigned with the use of similarity computations. It has been shown in [48] how the cluster profile can be used in order to efficiently compute $p(d|C_j)$ for each incoming document. This value is then used in order to determine the similarity of the documents to the different clusters. This is used in order to assign the documents to their closest cluster. We note that the methods in [6, 48] share a number of similarities in terms of (a) maintenance of cluster profiles, (b) use of cluster profiles (or condensed droplets) to compute similarity and assignment of documents to most similar clusters, and (c) the rules used to decide when a new singleton cluster should be created, or one of the older clusters should be replaced.

The main difference between the two algorithms is the technique which is used in order to compute cluster similarity. The OCTS algorithm uses the probabilistic computation $p(d|C_j)$ to compute cluster similarity, which takes the phrasal information into account during the computation process. One observation about OCTS is that it may allow for very similar clusters to co-exist in the current set. This reduces the space available for distinct cluster profiles. A second algorithm called OCTSM is also proposed in [48], which allows for merging of very similar clusters. Before each assignment, it checks whether pairs of similar clusters can be merged on the basis of similarity. If this is the case, then we allow the merging of the similar clusters and their corresponding cluster profiles. Detailed experimental results on the different clustering algorithms and their effectiveness are presented in [48]. A comprehensive review of text stream clustering algorithms may be found in the chapters on data clustering and streaming algorithms in [2].

10.8 Other Scenarios for Stream Clustering

Data Stream clustering is a fundamental problem, and numerous other domains arise, in which the data occurs naturally in the streaming context. In this section, we provide a brief introduction to these different data domains, along with pointers to the relevant literature.

10.8.1 Clustering Uncertain Data Streams

Uncertain data streams arise quite often in the context of sensor data or other hardware collection technology such as RFID in which there are significant errors in the data collection process. In many of these cases, the errors in the data can be approximated either in terms of statistical parameters such as the standard deviation, or in terms of probability density functions (pdfs). Such statistical information increases the richness of the noisy data because it provides information about the parts of data which are less reliable, and should therefore be emphasized less in the mining process.

In this context, a method called *UMicro* for clustering uncertain data streams was proposed in [7]. This method enhances the micro-clusters with additional information about the uncertainty of the data points in the clusters. This information is used in order to improve the quality of the distance functions for the assignments. This approach was further improved for the case of projected clustering of uncertain data streams [8, 60]. We are not providing a detailed discussion of these methods, since they are discussed in detail in the chapter on uncertain data clustering.

10.8.2 Clustering Graph Streams

Graph streams are created by edge-centered activity in numerous social and information networks. Many different kinds of streaming and clustering models are possible, depending upon the application scenario. These different models are as follows:

- The stream is a sequence of *objects*, each of which is a small graph containing a subset of nodes and edges. We wish to determine similar *objects* in the stream based on the similarities between the nodes and edges. For example, the DBLP bibliographic network, has an incoming stream of graph objects corresponding to the co-authored papers. It may be desirable to determine clusters of objects with similar structure. An algorithm for this problem, known as *Gmicro* was proposed in [9]. The micro-clustering representation is extended to handle edges, and a sketch based compression is used on the edges in order to reduce the impact of the massive domain of the edges.

- The stream is a sequence of *objects*, each of which is a small graph containing a subset of nodes and edges. We wish to determine node sets which co-occur frequently in these objects, and are also densely connected together. A method was proposed in [11] with the use of min-hash based graph stream summarization.

- The stream is a sequence of either *objects* or *edges*. It is desirable to determine dense node clusters in the graph.

The last problem is particularly general and is also related to general problem of dynamic community detection. In this context, a method was proposed for creating dynamic partitions of graphs with the use of structural reservoir sampling of edge streams [10]. While the work in [10] is targeted to outlier detection, a key intermediate step in the process is the dynamic generation of node clusters from the edge stream. The work in [28] has further refined the structural reservoir sampling techniques of [10] in order to provide more effective methods for node clustering in graph streams.

In many cases, such as social networks, content information is associated with the structure in the network. For example, the tweets in a *Twitter* stream have both structure and content. Such streams are referred to as *social streams*. The clustering of such streams requires the use of both structural information and content in the process. The work in [12] has designed methods for clustering social streams. Sketch-based methods are used in order to summarize the content in the social stream and use it for the clustering process. In addition, it has been shown in [12], how this approach may be used for event detection.

In the conventional streaming model, it is assumed that only one pass is allowed over the data, and the amount of memory available to the application is constant, irrespective of stream length. A

technique for determining the densest subgraph has been proposed in [19] in a *weakly streaming model*, where a limited number of passes (more than one) are allowed over the data, and the amount of available memory is sub-linear in the size of the input. This approach is able to determine the densest subgraph in passes which grow logarithmically with the graph size.

10.8.3 Distributed Clustering of Data Streams

In the context of sensor networks, the stream data is often available only in a *distributed setting*, in which large volumes of data are collected separately at the different sensors. A natural approach for clustering such data is to transmit all of the data to a centralized server. The clustering can then be performed at the centralized server in order to determine the final results. Unfortunately, such an approach is extremely expensive in terms of its communication costs, because of the large volume of the data which must be transmitted to the centralized server. Therefore, it is important to design a method which can reduce the communication costs among the different processors. A method proposed in [27] performs local clustering at each node, and merges these different clusters into a single global clustering at low communication cost. Two different methods are proposed in this work. The first method determines the cluster centers by using a *furthest point algorithm*, on the current set of data points at the local site. In the furthest point algorithm, the center of a cluster is picked as a furthest point to the current set of centers. For any incoming data point, it is assigned to its closest center, as long the distance is within a certain factor of an optimally computed radius. Otherwise, a re-clustering is forced by applying the furthest point algorithm on current set of points. After the application of the furthest point algorithm, the centers are transmitted to the central server, which then computes a global clustering from these local centers over the different nodes. These global centers can ten be transmitted to the local nodes if desired. One attractive feature of the method is that an approximation bound is proposed on the quality of the clustering. A second method for distributed clustering proposed in [27] is the *parallel guessing algorithm*.

A variety of other methods have also been proposed for distributed clustering of data streams. For example, techniques for distributed data stream clustering with the use of the k-medians approach were proposed in [61]. Another method for distributed sensor stream clustering which reduces the dimensionality and communication cost by maintaining an online discretization may be found in [57]. Finally, a method for distributed clustering of data streams with the use of the EM approach is proposed in [66].

10.9 Discussion and Conclusions

The problem of clustering is fundamental to a wide variety of streaming applications, because of its natural applicability to summarizing and representing the data in very small space. A wide variety of techniques have been proposed in the literature for stream clustering, such as partitioning methods, density-based methods, probabilistic methods etc. The stream clustering method has also been extended to other data domains such as categorical, text, and uncertain data. Finally, methods have also been proposed for clustering graph streams.

Many further directions of research are likely for this problem. These are as follows:

- Heterogeneous data streams are becoming extremely common because of applications such as social networks, in which large amounts of heterogeneous content are posted over time. It is desirable to determine clusters among these groups of heterogeneous objects.
- It is desirable to use a combination of links and content in order to determine clusters from

heterogeneous activity streams. This direction of research has also found increasing interest in the community. This is a further layer of complexity over the graph streaming scenario.

Furthermore, it would also be interesting to perform clustering streams from multiple sources, while simultaneously learning the nature of the dynamic relationships between the different streams.

Bibliography

[1] C. Aggarwal. *Data Streams: Models and Algorithms*, Springer, 2007.

[2] C. Aggarwal, and C. X. Zhai, *Mining Text Data*, Springer, 2012.

[3] C. Aggarwal, C. Procopiuc, J. Wolf, P. Yu, and J.-S. Park. Fast algorithms for projected clustering. In *ACM SIGMOD Conference*, 1999.

[4] C. Aggarwal. A Framework for Clustering Massive-Domain Data Streams. In *ICDE Conference*, 2009.

[5] C. Aggarwal, and P. Yu. Finding Generalized Projected Clusters in High Dimensional Spaces. In *ACM SIGMOD Conference*, 2000.

[6] C. Aggarwal, and P. Yu. A Framework for Clustering Massive Text and Categorical Data Streams. In *SIAM Conference on Data Mining*, 2006.

[7] C. Aggarwal, and P. Yu. A Framework for Clustering Uncertain Data Streams, *IEEE ICDE Conference*, 2008.

[8] C. Aggarwal. On High-Dimensional Projected Clustering of Uncertain Data Streams, *IEEE ICDE Conference*, 2009.

[9] C. Aggarwal, and P. Yu. On Clustering Graph Streams, *SDM Conference*, 2010.

[10] C. Aggarwal, Y. Zhao, and P. Yu. Outlier Detection in Graph Streams, *ICDE Conference*, 2011.

[11] C. Aggarwal, Y. Li, P. Yu, and R. Jin. On Dense Pattern Mining in Graph Streams, *VLDB Conference*, 2010.

[12] C. Aggarwal, and K. Subbian. Event Detection in Social Streams, *SIAM Conference on Data Mining*, 2012.

[13] C. Aggarwal. On Change Dignosis in Evolving Data Streams. In *IEEE TKDE*, 17(5), 2005.

[14] C. Aggarwal, J. Han, J. Wang, and P. Yu. A Framework for Clustering Evolving Data Streams. In *VLDB Conference*, 2003.

[15] C. Aggarwal, J. Han, J. Wang, and P. Yu. A Framework for Projected Clustering of High Dimensional Data Streams. In *VLDB Conference*, 2004.

[16] C. Aggarwal, J. Han, J. Wang, and P. Yu. On High Dimensional Projected Clustering of Data Streams, *Data Mining and Knowledge Discovery Journal*, 10(3), pp. 251–273, 2005.

[17] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. In *ACM SIGMOD Conference*, 1998.

[18] M. Ankerst, M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering Points To Identify the Clustering Structure. In *ACM SIGMOD Conference*, 1999.

[19] B. Bahamani, R. Kumar, V. Vassilvitskii. Densest Subgraph Mining in Streaming and MapReduce, *VLDB Conference*, 2012.

[20] A. Banerjee, and S. Basu. Topic Models over Text Streams: A Study of Batch and Online Unsupervised Learning, *SIAM Conference*, 2007.

[21] A. Banerjee, I. Dhillon, J. Ghosh, and S. Sra. Clustering on the unit hypersphere using von Mises-Fisher distributions. In *Journal of Machine Learning Research*, 6: pages 1345–1382, 2005.

[22] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation, *Journal of Machine Learning Research*, 3: pp. 993–1022, 2003.

[23] P. Bradley, U. Fayyad, and C. Reina. Scaling Clustering Algorithms to Large Databases. In *ACM KDD Conference*, 1998.

[24] F. Cao, M. Ester, W. Qian, A. Zhou. Density-based Clustering over an Evolving Data Stream with Noise, In *SDM Conference*, 2006.

[25] Y. Chen, and L. Tu. Density-based clustering for real time stream data, *ACM KDD Conference*, 2007.

[26] G. Cormode and S. Muthukrishnan. An Improved Data-Stream Summary: The Count-min Sketch and its Applications. In *Journal of Algorithms*, 55(1), 2005.

[27] G. Cormode, S. Muthukrishnan, and W. Zhuang. Conquering the divide: Continuous clustering of distributed data streams. *ICDE Conference*, 2007.

[28] A. Eldawy, R. Khandekar, and K. L. Wu. On clustering Streaming Graphs, *ICDCS Conference*, 2012.

[29] C. Elkan. Clustering documents with an exponential family approximation of the Dirichlet compund multinomial distribution. *ICML Conference*, 2006.

[30] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *ACM KDD Conference*, 1996.

[31] F. Farnstrom, J. Lewis, and C. Elkan. Scalability for Clustering Algorithms Revisited. *ACM SIGKDD Explorations*, 2(1):pp. 51–57, 2000.

[32] D. Fisher. Knowledge Acquisition via incremental conceptual clustering. *Machine Learning*, 2: pp. 139–172, 1987.

[33] M. Franz, T. Ward, J. McCarley, and W.-J. Zhu. Unsupervised and supervised clustering for topic tracking. *ACM SIGIR Conference*, 2001.

[34] G. P. C. Fung, J. X. Yu, P. Yu, and H. Lu. Parameter Free Bursty Events Detection in Text Streams, *VLDB Conference*, 2005.

[35] J. Gao, J. Li, Z. Zhang, and P.-N. Tan. An incremental data stream clustering algorithm based on dense units detection. In *PAKDD Conference*, 2005.

[36] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams. In *IEEE FOCS Conference*, 2000.

[37] S. Guha, R. Rastogi, and K. Shim. CURE: An Efficient Clustering Algorithm for Large Databases. In *ACM SIGMOD Conference*, 1998.

[38] Q. He, K. Chang, E.-P. Lim, and J. Zhang. Bursty feature representation for clustering text streams. *SDM Conference*, 2007.

[39] Z. He, X. Xu, and S. Deng. Squeezer: An Efficient Algorithm for Clustering Categorical Data, *Journal of Computer Science and Technology (JCST)*, 17(5), pp. 611–624, 2002.

[40] Z. He, X. Xu, S. Deng, and J. Huang. Clustering Categorical Data Streams, *Published Online on Arxiv*, December, 2004.
`http://arxiv.org/ftp/cs/papers/0412/0412058.pdf`

[41] T. Hoffman. Probabilistic Latent Semantic Indexing, *ACM SIGIR Conference*, 1999.

[42] A. Jain, R. Dubes. Algorithms for Clustering Data, *Prentice Hall*, New Jersey, 1998.

[43] C. Jia, C. Tan, and A. Yong. A Grid and Density-based Clustering Algorithm for Processing Data Stream, *International Conference on Genetic and Evolutionary Computing*, 2008.

[44] L. Kaufman, and P. Rousseuw. *Finding Groups in Data- An Introduction to Cluster Analysis*. Wiley Series in Probability and Math. Sciences, 1990.

[45] J. Kleinberg, Bursty and hierarchical structure in streams, *ACM KDD Conference*, pp. 91–101, 2002.

[46] P. Kranen, I. Assent, C. Baldauf, and T. Seidl. The ClusTree: Indexing micro-clusters for anytime stream mining. *Knowledge and Information Systems*, 29(2), pp. 249–272, 2011.
http://link.springer.com/article/10.1007

[47] H.-P. Kriegel, P. Kroger, I. Ntoutsi, and A. Zimek. Density based subspace clustering over dynamic data. *SSDBM Conference*, 2011.

[48] Y.-B. Liu, J.-R. Cai, J. Yin, and A. W.-C. Fu. Clustering Text Data Streams, *Journal of Computer Science and Technology*, Vol. 23(1), pp. 112–128, 2008.

[49] Y. Lu, Y. Sun, G. Xu, and G. Liu. A Grid-based Clustering Approach for High-Dimensional Data Streams, *Advanced Data Mining and Applications*, 2005.

[50] G. Manku, and R. Motwani. Approximate Frequency Counts over Data Streams, *VLDB Conference*, 2002.

[51] R. Ng, and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. In *Very Large Data Bases Conference*, 1994.

[52] I. Ntoutsi, A. Zimek, T. Palpanas, H.-P. Kriegel, and A. Zimek. Density-based Projected Clustering over High Dimensional Data Streams, In *SDM Conference*, 2012.

[53] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-Data Algorithms For High-Quality Clustering. In *ICDE Conference*, 2002.

[54] C. Ordonez. Clustering Binary Data Streams with *K*-means. In *Data Mining and Knowledge Discovery Workshop*, 2003.

[55] N. H. Park, and W. S. Lee. Statistical Grid-based Clustering over Data Streams, *ACN SIGMOD Record*, 33(1), March, 2004.
`http://www09.sigmod.org/sigmod/record/issues/0403/A15.park-lee.pdf`

[56] N. H. Park, and W. S. Lee. Cell trees: An adaptive synopsis structure for clustering multi-dimensional on-line data streams. *Data and Knowledge Engineering*, 63(2), pp. 528–549, 2007.

[57] P. Rodrigues, J. Gama, and L. Lopes. Clustering Distributed Sensor Data Streams, *PKDD Conference*, 2008.

[58] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.

[59] M. Song, and H. Wang. Highly efficient incremental estimation of Gaussian Mixture Models for online data stream clustering, *SPIE*, 2005.

[60] C. Zhang, M. Gao, and A. Zhou. Tracking High Quality Clusters over Uncertain Data Streams, *ICDE Conference*, 2009.

[61] Q. Zhang, J. Liu, and W. Wang. Approximate clustering of distributed data streams, *ICDE Conference*, 2008.

[62] Q. Zhang, J. Liu, and W. Wang. Incremental subspace clustering over multiple data streams. *ICDM Conference*, 2007.

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

[64] S. Zhong. Efficient Streaming Text Clustering. *Neural Networks*, Volume 18, Issue 5–6, 2005.

[65] A. Zhou, F. Cao, W. Qian, and C. Jin. Tracking Clusters in Evolving Data Streams over Sliding Windows, *Knowledge and Information Systems*, 15(2), pp. 181–214, 2008.

[66] A. Zhou, F. Cao, Y. Yan, C. Sha, and X. He. Distributed Clustering of Data Streams: A Fast EM-based Approach, *ICDE Conference*, 2007.