Incremental and Adaptive Clustering Stream Data over Sliding Window

Xuan Hong Dang[1], Vincent C S Lee[1], Wee Keong Ng[2], and Kok Leong Ong[3]

[1] Monash University, Australia, xhdang;vincent.lee@infotech.monash.edu
[2] Nanyang Technological University, Singapore, awkng@ntu.edu.sg
[3] Deakin University, Australia, leong@deakin.edu.au

Abstract. Cluster analysis has played a key role in data understanding. When such an important data mining task is extended to the context of data streams, it becomes more challenging since system resources are bounded whilst the data arrive at the system unboundedly and in one-pass manner. The problem is even more difficult when the clustering task is considered in a sliding window model in which the requirement of outdated data elimination must be dealt with properly. We propose SWEM algorithm that is designed based on the Expectation Maximization technique to address these challenges. Equipped in SWEM is the capability to compute clusters incrementally using a small number of statistics summarized over the stream and the capability to adapt to the stream distribution's changes. The feasibility of SWEM has been verified via a number of experiments and we show that it is superior than Clustream algorithm, for both synthetic and real datasets.

1 Introduction

In recent years, we are seeing a new class of applications that changed the traditional view of databases as a static store of information. These applications are commonly characterized by the unbounded data streams they generate (or receive), and the need to analyze them in a continuous manner over limited computation resources [10, 3, 8]. Further, stream data can be lost under high speed conditions, become outdated in the analysis context, or intentionally dropped through techniques like sampling [4] or load shedding [16]. This makes it imperative to design algorithms that compute answers in a continuous fashion with only one scan over stream data whilst operating under the resource limitations.

Among various data mining tasks, clustering is one of the most important tasks that widely helps to analyze and uncover structures in the data. Research in data stream clustering reported so far has mostly focused on two mining models, the landmark window [12, 2, 18] and the forgetful window [7, 8]. In the former one, clustering results are computed based on the entire data elements generated so far in the stream. In the latter model, they are also discovered from the complete history of stream data; however, the weight (or importance) of each data point is decreased with time by using a fading function. While these two mining models are useful in some data stream applications, there is a strong demand to devise novel techniques that are able to cluster the data stream in a sliding window model. For example, in network monitoring, the changes of stream characteristics in the past several hours are much more valuable compared to those happened in days ago for detection of

network intrusions [11]. Moreover, it has also been shown that [19] performing a stream mining or querying task in the sliding window model is the most general and the most challenging work since it further deals with the problem of outdated data elimination.

In this paper, we propose SWEM algorithm (clustering data streams in a time-based Sliding Window with Expectation Maximization technique) to address the above challenges. Compared to other stream clustering algorithms relying on k-means [13, 12, 2, 1], an EM-based algorithm is a soft clustering method and thus, it has some properties that are desirable for stream environments such as robustness to noise or the ability to handle missing data (as clearly shown in [18]). Further, it has a strong statistical basis and theoretically guarantees optimal convergence. The SWEM algorithm consists of two stages which are strictly designed to address the problem of constrained memory usage and one-pass processing over data streams. In the first stage, SWEM scans data records arriving in the stream and summarizes them into a set of micro components where each one is characterized by only a small number of statistics. These small amount of statistics in turns are effectively used in the second stage of SWEM in order to approximate a set of global clusters. Most importantly, in order to address the problem of characteristics changing in stream environments, we develop a method to adaptively split and merge micro components. Such an approach provides a flexible technique to re-distribute micro components across the data space and thus efficiently approximates the evolving distribution in the stream. Experiments on various data streams have empirically shown that SWEM is very effective in finding clusters from a time-based sliding window. It is not only able to process data streams in an incremental fashion with confined memory space, its clustering qualities are high and close to those of a standard EM (working without any stream environment's constraints). Furthermore, the algorithm also outperforms the well-known CluStream algorithms [2] both in time performance and clustering qualities.

The rest of the paper is organized as follows. In the following section, we briefly review some related work. The problem of clustering data streams from a time-based sliding window is then formally formulated in Section 3. Section 4 describes our algorithm in details and Section 5 reports its experimental results. Our conclusions are finally presented in Section 6.

2 Related Work

Previous work on clustering data streams focused on developing space-efficient and one-pass algorithms. STREAM is one of the first algorithms that addresses the problem in a landmark window model [13]. It is a divide-and-conquer algorithm that builds clusters incrementally and hierarchically using bicriterion approximation algorithms. The stream is processed in a batch model where a set of data points fitting in the main memory are clustered into a smaller set of medians (using the K-means technique). The process is iterated for the next batches of data points until the set of medians fit the memory space and they are then clustered into a higher level of medians. In [6, 15], this approach is extended in that both the approximation factor and the storage memory are improved. It is also expanded for stream clustering in a sliding window model [5]. Unfortunately, in order to deal

with the issue of finding clusters in window sliding mode, their algorithm simply re-clusters all summarized data points. Hence, there is no incremental work for the process of computing global clusters. Also, there is no experimental work to report the accuracy and effectiveness of the algorithm. Furthermore, by adopting the exponential histogram [9] as the framework for summarizing data, their algorithm can only work with count-based sliding window where the number of data records arriving in the window must be fixed in advance.

Recently, CluStream [2], DenStream [7] and D-Stream [8] are representative algorithms focusing on evolving data stream clustering. In CluStream [2], it divides the clustering process into online and offline components. The online component is designed to quickly update raw data into a set of micro clusters (maintained so far in the stream) and the offline component is designed to compute clustering results based on snapshots captured at different times on the set of micro clusters. To address the issue of confined memory space, CluStream stores snapshots at various levels of granularity. Recently captured ones are stored at finer granularity while earlier snapshots are stored at coarser granularity. This technique is then later extended in HPStream [1] where the concept of projected clusters is introduced to cluster high dimensional data streams. DenStream [7] is another algorithm proposed to address the issue of clustering evolving data streams. The authors extended the DBSCAN method to the online-offline framework proposed in Clustream [2]. A fading function is also used to gradually reduce the weight of each micro cluster with time. The DBSCAN with the concept of *density connectivity* is then applied to the set of micro clusters to derive global clustering results. D-Stream [8] is another density-based algorithm for stream clustering. It divided the data space into a set of disjointed grids. Then, for each data point arriving in the stream, D-Stream maps it into a corresponding grid based on its dimensional values. Similar to DenStream, a fading function is used in D-Stream to reduce the weight of each grid over time. The final clustering results are then obtained based on the density and the connectivity of grids maintained in the data space. Since the number of grids is exponentially proportional to the number of dimensions, D-Stream has to frequently prune un-dense grids whose weight is below than a threshold. Motivated by the drawbacks of the hard clustering techniques (e.g., based on k-means) applied on various stream environments, CluDistream [18] has recently been proposed to cluster data streams based on the Expectation Maximization technique. Although this algorithm has been shown to provide significant results over other methods (especially in distributed stream environments where data could be noised or missed due to transmission), it only addressed the clustering problem in a *landmark* window model where each EM algorithm is simply implemented at each node of the distributed network.

3 Problem Formulation

We consider a data stream as a time ordered series of data points (or records) $\mathcal{DS} = \{x_1, x_2, ..., x_n, ...\}$ where each x_i has the form $x_i = \{x_i^1, x_i^2, ..., x_i^d\}$ in d-dimensional space.

We focus on the *time-based* sliding window model. In this model, let $TS_0, TS_1, \ldots, TS_{i-b+1}, \ldots, TS_i$ denote the time periods elapsed so far in the stream. Each time

period contains multiple data points arriving in that interval. Given an integer b, the time-based sliding window is defined as the set of data records arriving in the last b time periods and denoted by $\mathcal{SW} = \{TS_{i-b+1}, \ldots, TS_{i-1}, TS_i\}$. TS_i is called the latest time slot and TS_{i-b} is called the expiring one in the sliding window. When the time shifts to the new slot TS_i, the effect of all data points in TS_{i-b} will be eliminated from the clustering model.

In our sliding window model, once data points in a time slot are processed, they cannot be retrieved for further computation at a later time (unless their summarized statistics are explicitly stored in memory). The amount of memory available is assumed to be limited. In particular, it is sub-linear in the size of the sliding window. As such, approaches that require storing the entire data points in the sliding window are not acceptable in this model. We also assume that streaming data evolve with time and data records are generated as a result of a dynamic statistical process that consists of k mixture models (or components). Each model corresponds to a cluster that follows a multivariate normal distribution. Consequently, any cluster $C_h, 1 \leq h \leq k$, is characterized by a parameter: $\phi_h = \{\alpha_h, \mu_h, \Sigma_h\}$ where α_h is the cluster weight, μ_h is its vector mean of d-dimension (determining the cluster center), and Σ_h is its $d \times d$ covariance matrix (determining the cluster shape).

Accordingly, we define the data stream clustering problem in a sliding window model as follows: Given the number of time slots b, our goal is to cluster the data stream incrementally by identifying a set of parameters $\Phi_G = \{\phi_1, ..., \phi_k\}$ that optimally fit the current set of data points arriving in the last b time periods of the stream.

4 Algorithm Description

In this section, we describe each phase of SWEM algorithm in details. The initial phase is applied at the very beginning of the stream where the sliding window includes only one time slot. The incremental phase is then performed when data in subsequent time slots arrive. Once the window in the sliding mode, the expiring phase is applied to remove the effect of out-dated data (in the last time slot) from the clustering results.

4.1 Initial Phase

We compute m distributions (also called micro components) modelling the data within each time slot of the sliding window. Let Φ_L be the set of parameters of these local components, i.e., $\Phi_L = \{\phi_1, ..., \phi_m\}$, where each micro component MC_ℓ, $1 \leq \ell \leq m$, is also assumed to follow a multivariate Gaussian distribution characterized by three parameters $\phi_\ell = \{\alpha_\ell, \mu_\ell, \Sigma_\ell\}$. For the initial phase where $SW = \{TS_0\}$, the initial values for these parameters will be randomly chosen. We clarify the following concepts and assumptions.

In our model, each data point belongs to all components yet with different probability. Given x, its probability in component ℓ^{th} is computed based on the Bayes rule: $p(\phi_\ell|x) = \alpha_\ell \times p_\ell(x|\phi_\ell)/p(x) = \alpha_\ell \times p_\ell(x|\phi_\ell)/\sum_{i=1}^{m} \alpha_i \times p_i(x|\phi_i)$, in which

$$p_\ell(x|\phi_\ell) = \frac{1}{(2\pi)^{d/2}|\Sigma_\ell|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_\ell)^T \Sigma_\ell^{-1}(x-\mu_\ell)\right] \quad (1)$$

where μ_ℓ is the d-dimensional mean vector and Σ_ℓ is the covariance matrix of size $d \times d$. $|\Sigma_\ell|$ and $(\Sigma_\ell)^{-1}$ are respectively the determinant and inverse matrix of Σ_h. Since we assume each attribute is independent of one another, Σ_ℓ is a diagonal variance matrix. Thus, its determinant and inverse matrix can be easily computed: $|\Sigma_\ell| = \prod_{i=1}^{d}(\sigma_\ell^i)^2$ and $\Sigma_\ell^{-1} = \begin{pmatrix} 1/(\sigma_\ell^1)^2 & 1/(\sigma_\ell^2)^2 & \dots & 1/(\sigma_\ell^d)^2 \end{pmatrix} \mathbf{I}$, where σ_ℓ^i is the variance at dimension i of MC_ℓ and $\mathbf{I}$ is the identity matrix.

As we assume data points in the stream are generated independently, the probability of n data records arriving in TS_0 is computed by the product:

$$p(TS_0|\Phi_L) = \prod_{x_i \in TS_0} p(x_i|\Phi_L) = \prod_{i=1}^{n}\sum_{\ell=1}^{m} \alpha_\ell \times p_\ell(x_i|\phi_\ell) \tag{2}$$

Since this probability is small, we typically work with its log likelihood form. We define the average log likelihood measure which is used to evaluate how well the set of micro components approximating the data stream within this time period by:

$$Q(\Phi_L) = \frac{1}{|TS_0|} \log \prod_{x \in TS_0} \sum_{h=1}^{m} \alpha_\ell \times p_\ell(x_i|\phi_\ell) \tag{3}$$

In its first stage, SWEM employs the EM technique to maximize this quantity. Specifically, the algorithm begins with an initial estimation of Φ_L and iteratively updates it until $|Q(\Phi_L^{t+1}) - Q(\Phi_L^t)| \leq \epsilon$. Given Φ_L^t, SWEM updates it at iteration $t+1$ as follows:

In the E-step: $$p(\phi_\ell|x) = \frac{\alpha_\ell^{(t)} \times p_\ell(x|\mu_\ell^{(t)}, \Sigma_\ell^{(t)})}{\sum_i \alpha_i^{(t)} \times p_i(x|\mu_i^{(t)}, \Sigma_i^{(t)})}$$

In the M-step: $$\alpha_\ell^{(t+1)} = \frac{1}{n}\sum_{x \in TS_0} p(\phi_\ell|x); \quad \mu_\ell^{(t+1)} = \sum_{x \in TS_0} \frac{p(\phi_\ell|x)}{n_\ell} \times x;$$
$$\Sigma_\ell^{(t+1)} = \sum_{x \in TS_0} \frac{p(\phi_\ell|x)}{n_\ell} \times (x - \mu_\ell^{(t+1)})(x - \mu_\ell^{(t+1)})^T$$

where $n_\ell = \sum_{x \in TS_0} p(\phi_\ell|x)$.

When the algorithm converges, these micro components are approximated by only keeping a triple $T_\ell = \{N_\ell, \theta_\ell, \Gamma_\ell\}$ for each MC_ℓ. Essentially, let S_ℓ be the set of data points assigned to MC_ℓ (to which they have the highest probability), then:

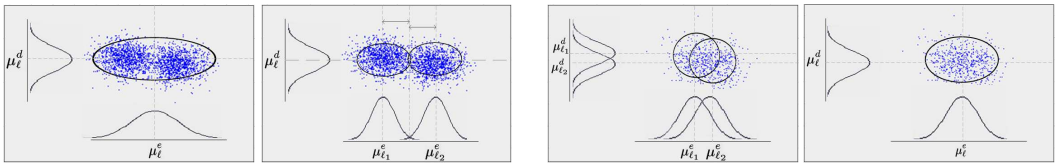
- $N_\ell = |S_\ell|$ is the cardinality of the set S_ℓ;
- $\theta_\ell = \Sigma_{x_i \in S_\ell} x_i$ is the linear summation of the data points in the set S_ℓ; and
- $\Gamma_\ell = \Sigma_{x_i \in S_\ell} x_i x_i^T$ is the squared summation of these points.

The important property of T_ℓ is that it is sufficient to compute the mean and covariance of MC_ℓ. Concretely, $\mu_\ell = N_\ell^{-1}\theta_\ell$ and $\Sigma_\ell = N_\ell^{-1}\Gamma_\ell - N_\ell^{-2}\theta_\ell \times \theta_\ell^T$. Furthermore, its additive property ensures that the mean and covariance matrix of a merged component can be computed from the values of each member component: $T_\ell = \{N_{\ell_1} + N_{\ell_2}, \theta_{\ell_1} + \theta_{\ell_2}, \Gamma_{\ell_1} + \Gamma_{\ell_2}\}$ given two member components $T_{\ell_1} = \{N_{\ell_1}, \theta_{\ell_1}, \Gamma_{\ell_1}\}$ and $T_{\ell_2} = \{N_{\ell_2}, \theta_{\ell_2}, \Gamma_{\ell_2}\}$.

Therefore, SWEM treats these triples as the sufficient statistics summarizing the information regarding the data points within the first time slot of the stream. They are then used in the second stage of the algorithm to compute the k global clusters. In specific, SWEM updates parameters for each global model $\phi_h \in \Phi_G$ as follows:

E-step: $$p(\phi_h|T_\ell) = \frac{\alpha_h^{(t)} \times p_h(\frac{1}{N_\ell}\theta_\ell|\mu_h^{(t)}, \Sigma_h^{(t)})}{\sum_{i=1}^{k} \alpha_i^{(t)} \times p_i(\frac{1}{N_\ell}\theta_\ell|\mu_i^{(t)}, \Sigma_i^{(t)})}$$

M-step: $$\alpha_h^{(t+1)} = \frac{1}{n}\sum_{\ell=1}^{m} N_\ell \times p(\phi_h|T_\ell); \quad \mu_h^{(t+1)} = \frac{1}{n_h}\sum_{\ell=1}^{m} p(\phi_h|T_\ell) \times \theta_\ell;$$



(a) Before and after splitting (b) Before and after merging

Fig. 1. Split and merge components

$$\Sigma_h^{(t+1)} = \frac{1}{n_h}\left[\sum_{\ell=1}^{m} p(\phi_h|T_\ell)\Gamma_\ell - \frac{1}{n_h}\sum_{\ell=1}^{m}\left(p(\phi_h|T_\ell)\theta_\ell\right)\left(p(\phi_h|T_\ell)\theta_\ell\right)^T\right]$$

where $n_h = \sum_{\ell=1}^{m} N_\ell \times p(\phi_h|T_\ell)$.

4.2 Incremental Phase

This phase is incrementally applied when data in a new time slot arrive at the system. Nonetheless, different from the initial phase where SWEM has to randomly choose initial parameters for micro components, in this phase it utilizes the converged parameters in the previous time slot as the initial values for the mixture models' parameters. The rationale is that we may expect the characteristics of the stream only slightly changed between two consecutive time slots. Thus, inheriting the converged parameters of the previous time slot can minimize the number of iterations in the next one.

Essentially, after the first iteration on the data points arriving in the new time slot, SWEM compares the quantity $Q(\Phi_L)$ with the converged one in the previous time slot. If the two values are not much different, it is safe to say that the distribution in the new time interval is similar to the one in the previous time interval. Otherwise, the stream distribution has changed and it is necessary to adapt the set of micro components accordingly in order to avoid the local maximal problem [17]. We therefore develop in SWEM two split and merge operations to *discretely* redistribute components across the entire data space.

Figure 1(a) illustrates the idea of SWEM's split operation. A micro component is chosen for dividing if it is large enough and has the highest variance sum (i.e., its data are mostly spread). An MC_ℓ is considered large if its weight $\alpha_\ell > \frac{2}{m}\sum_i \alpha_i$ and the dimension for the split is the one having the maximum variance. Let e be such dimension, then the new means and variances for e in two new micro components MC_{ℓ_1} and MC_{ℓ_2} are approximated as follows:

$$\mu_{\ell_1}^e = \int_{\mu_\ell^e - 3\sigma_\ell^e}^{\mu_\ell^e} x \times p_{\ell,e}(x|\phi_\ell)dx; \qquad \mu_{\ell_2}^e = \int_{\mu_\ell^e}^{\mu_\ell^e + 3\sigma_\ell^e} x \times p_{\ell,e}(x|\phi_\ell)dx$$

$$(\sigma_{\ell_1}^e)^2 = \int_{\mu_\ell^e - 3\sigma_\ell^e}^{\mu_\ell^e} x^2 \times p_{\ell,e}(x|\phi_\ell)dx - (\mu_{\ell_1}^e)^2; \; (\sigma_{\ell_2}^e)^2 = \int_{\mu_\ell^e}^{\mu_\ell^e + 3\sigma_\ell^e} x^2 \times p_{\ell,e}(x|\phi_\ell)dx - (\mu_{\ell_2}^e)^2$$

For other dimensions, their means and variances are kept unchanged. The above computations are derived given the observation that each dimension is to follow

a Gaussian distribution in which 99.7% of data are within $\mu \pm 3\sigma$. Therefore, the integrals with lower and upper bounds chosen in this range can approximately cover all component's data points projecting on that dimension.

On the other hand, as illustrated in Figure 1 (b), two components are selected for merging if they are small and close enough. An MC_ℓ is considered small if its weight $\alpha_\ell < \frac{1}{2m}\sum_i \alpha_i$ and the distance, which is measured based on the Mahalanobis distance, between two components are closed enough. Notice that compared to the Euclidean distance, the Mahalanobis works more effectively in SWEM since it takes into account the covariance matrix (actually describing the spread of component). We define the average squared Mahalanobis distance between two components as follows: $Avg(D_{i,j}) = \frac{1}{2}[(\mu_i - \mu_j)^T \Sigma_j^{-1}(\mu_i - \mu_j) + (\mu_j - \mu_i)^T \Sigma_i^{-1}(\mu_j - \mu_i)]$.

Given MC_{ℓ_1} and MC_{ℓ_2} to be merged, SWEM computes parameters for their merging component MC_ℓ based on the additive property:

$$\alpha_\ell = \alpha_{\ell_1} + \alpha_{\ell_2}; \quad \mu_\ell = \frac{\alpha_{\ell_1}}{\alpha_{\ell_1}+\alpha_{\ell_2}} \times \mu_{\ell_1} + \frac{\alpha_{\ell_2}}{\alpha_{\ell_1}+\alpha_{\ell_2}} \times \mu_{\ell_2}; \quad \Sigma_\ell = \frac{\Gamma_\ell}{n(\alpha_{\ell_1}+\alpha_{\ell_2})} - \frac{\theta_\ell \times \theta_\ell^T}{(n(\alpha_{\ell_1}+\alpha_{\ell_2}))^2}$$

In that, $\theta_\ell = n \times (\alpha_{\ell_1} \times \mu_{\ell_1} + \alpha_{\ell_2} \times \mu_{\ell_2}); \quad \Gamma_\ell = n[\alpha_{\ell_1}(\Sigma_{\ell_1} + \mu_{\ell_1}\mu_{\ell_1}^T) + \alpha_{\ell_2}(\Sigma_{\ell_2} + \mu_{\ell_2}\mu_{\ell_2}^T)]$.

When SWEM converges to the optimal solution, m micro components are again summarized into m triples. Then, k global clusters are updated with these new statistics. This second stage of the algorithm is analogous to the second stage of the initial phase. Nonetheless, SWEM derives the k global models from only $(m + k)$ components in this incremental phase.

4.3 Expiring Phase

We apply this phase of the SWEM algorithm when the window slides and the oldest time slot is expiring from the mining model. As such, it is necessary to update $\Phi_G = \{\phi_1, \ldots, \phi_k\}$ by subtracting the statistics summarized in the local model $\Phi_L = \{\phi_1, \phi_2, ..., \phi_m\}$ of the expiring time slot. SWEM controls this process by using a fading factor λ $(0 < \lambda < 1)$ to gradually remove these statistics. The closer to 1 the λ is, the smaller the amount of the reducing weights being eliminated at each iteration of the algorithm. Thus, this factor provides us a method to prevent the effect of each expiring component from reducing too fast or too slow, which would cause the local optimal convergence in SWEM. Essentially, at each iteration t of the algorithm, SWEM reduces the weight of each expiring MC_ℓ by $N_\ell^{(t)} = \lambda^{(t)} N_\ell$. This also means that the reducing amount, denoted by $r_\ell^{(t)}$, is:

$$r_\ell^{(t)} = (1 - \lambda)\lambda^{(t-1)} N_\ell \tag{4}$$

The following theorem guarantees that the number of iterations t can be any arbitrary integer while the total reducing weights on each expiring component approaches (but never exceeds) its original value.

Theorem 1. *Let t be an arbitrary number of iterations used by the SWEM algorithm. Then for each expiring micro component MC_ℓ:*

$$\lim_{t \to \infty} \sum_t r_\ell^{(t)} = N_\ell$$

Proof. At the first iteration, $N_\ell^{(1)} = \lambda N_\ell$. Thus the reducing amount is $r_\ell^{(1)} = N_\ell - \lambda N_\ell = (1 - \lambda)N_\ell$. At the second iteration, $N_\ell^{(2)} = \lambda N_\ell^{(1)} = \lambda^2 N_\ell$ and $r_\ell^{(2)} = N_\ell^{(1)} - \lambda N_\ell^{(1)} = (1 - \lambda)\lambda N_\ell$.

By induction, at the iteration t, the reducing weight is $r_\ell^{(t)} = (1-\lambda)\lambda^{(t-1)}N_\ell$. Therefore, the total reducing amount so far is:

$$\begin{aligned}\sum_t r_\ell^{(t)} &= (1-\lambda)N_\ell + (1-\lambda)\lambda N_\ell + ... + (1-\lambda)\lambda^{(t-1)}N_\ell \\ &= (1-\lambda)N_\ell[1+\lambda+....+\lambda^{t-1}]\end{aligned}$$

It is clear that:

$\lim_{t\to\infty}(1-\lambda)[1+\lambda+....+\lambda^{t-1}] = \lim_{t\to\infty}(1-\lambda^t) = 1$ since $\lambda < 1$. □

Given the factor λ to progressively remove out-dated statistics, the E-step computes the posterior probability for each expiring component by:

$$p(\phi_h|T_\ell) = \frac{\alpha_h^{(t)} \times p_h(\frac{1}{N_\ell}\theta_\ell|\mu_h^{(t)}, \Sigma_h^{(t)})}{\sum_{i=1}^k \alpha_i^{(t)} \times p_i(\frac{1}{N_\ell}\theta_\ell|\mu_i^{(t)}, \Sigma_i^{(t)})}$$

and at the M-step, these posterior probabilities is subtracted from the global models. Specifically, the number of data points in the sliding window n_G is updated by: $n_G^{(t+1)} = n_G^{(t)} - \sum_{\ell=1}^m r_\ell^{(t+1)}$, in that the amount $\sum_{\ell=1}^m r_\ell^{(t+1)}$ is the total number of data points removed when applying Equation 4 at $t+1$. Subsequently, the weight of each global model ϕ_h is simply updated by:

$$\alpha_h^{(t+1)} = \frac{n_h^{(t+1)}}{n_G^{(t+1)}} \quad \text{where} \quad n_h^{(t+1)} = \alpha_h^{(t)} \times n_G^{(t)} - \sum_{\ell=1}^m p(\phi_h|T_\ell) \times r_\ell^{(t+1)}$$

The first factor $\alpha_h^{(t)} \times n_G^{(t)}$ is the estimated number of data points belonging to the global model ϕ_h while the second factor is its total number of data points (weighed by the posterior probability) now expiring at $t+1$.

To update new value for μ_h and Σ_h of each global model ϕ_h, SWEM first computes: $\theta_h^{(t+1)} = \theta_h^{(t)} - \sum_{\ell=1}^m p(\phi_h|T_\ell) \times r_\ell^{(t+1)} \times \theta_\ell/N_\ell$, in that θ_ℓ/N_ℓ actually is the vector mean of the expiring micro component (which does not change during this process). Then $\mu_h^{(t+1)}$ and $\Sigma_h^{(t+1)}$ are computed by:

$$\mu_h^{(t+1)} = \frac{\theta_h^{(t+1)}}{n_h^{(t+1)}}; \qquad \Sigma_h^{(t+1)} = \frac{1}{n_h^{(t+1)}}\left[\Gamma_h^{(t+1)} - \frac{1}{n_h^{(t+1)}}\theta_h^{(t+1)}\theta_h^{(t+1)^T}\right]$$

where $\Gamma_h^{(t+1)} = \Gamma_h^{(t)} - \sum_{\ell=1}^m \left[p(\phi_h|T_\ell) \times r_\ell^{(t+1)} \times \frac{\theta_\ell}{N_\ell}\right]\left[p(\phi_h|T_\ell) \times r_\ell^{(t+1)} \times \frac{\theta_\ell}{N_\ell}\right]^T$

and $\Gamma_h^{(t)} = n_h^{(t)}\left[\Sigma_h^{(t)} + \mu_h^{(t)}\mu_h^{(t)^T}\right]$

During this process, whenever any global model has weight becoming too small, it is safe to remove it from the mining results. This happens if a global model was formed by only the data points arriving in the expiring time slot. Thus, when this time interval is beyond the window, such a global model is eliminated as well. When a global model is deleted, another one which has the highest summation on variances should be split in order to keep the number of global clusters unchanged. The computation for a global model splitting is similar to the case with splitting micro components presented in Section 4.2.

Finally, the memory space of SWEM is guaranteed by the following theorem.

Theorem 2. *The memory space consumption of SWEM is* $O(n+(d^2+d+1)(mb+k))$*, where* n *is the maximal number of data points arriving in a time slot and* b *is the number of time slots within a sliding window.*

Proof. The memory consumption includes two parts. The first part is the data buffer storing the new data points. We assume that n is the maximal number of data points that can arrive in one time slot. The second part is the space for recording the parameters of the mixture Gaussian models at each time slot as well as the entire sliding window. For a Gaussian mixture model of k global components, the memory consumption includes k weights α_h, k d-dimensional mean vectors μ_h, and k $d \times d$ covariance matrices. At each time slot of the sliding window, the memory space needs to maintain m micro components, each characterized by a weight N_ℓ, a d-dimensional vector sum θ_ℓ and a $d \times d$ matrix of squared sum Γ_ℓ. There are b time slots in the sliding window and therefore in total, the memory consumption of the algorithm is $O(n + (d^2 + d + 1)(mb + k))$. □

5 Experimental Results

5.1 Experimental Setup

We implement three algorithms, our SWEM technique, a standard EM algorithm denoted by stdEM [4], and CluStream algorithm [2] using Microsoft Visual C++ version 6.0. All experiments are conducted on a 1.9GHz Pentium IV PC with 1GB memory space running on the Windows XP platform. In the following, to describe a dataset we use D to denote its dimensions, K to denote its number of clusters, and N to its size in terms the number of data records. We first evaluate the clustering quality of SWEM based on the results returned by the standard EM technique. Then, its sensibility to various parameter settings is verified. Finally, the performance of SWEM is compared with that of CluStream algorithm.

5.2 Clustering Quality Evaluation

Using the method described in [18], we generate three datasets each with 100,000 data records and the number of dimensions and clusters are varied from 2 to 10. The data points of each dataset follow a series of Gaussian distributions. To simulate the evolution of the stream over time, we generate new Gaussian distribution for every 20k points by probability of 10%. With the above notations, three datasets are respectively denoted $D2.K10.N100k$, $D4.K5.N100k$ and $D10.K4.N100k$. Unless otherwise indicated, we set the sliding window equal to 5 time slots and use the following parameters: the number of micro components $m = 6K$ (where K is the number of global clusters), the error bound on average log likelihood $\epsilon = 0.001$, the merging threshold based on Mahalanobis distance $Avg(D_{i,j}) = 1$ and the fading factor $\lambda = 0.8$. Similar to other clustering algorithms relying on the EM technique [17, 18], the clustering qualities of SWEM and stdEM are evaluated by using the average log likelihood measure.

[4] stdEM works without any constraint of stream environments and it performs clustering on all data points in the sliding window.

	D2.K10.N100k			D4.K5.N100k			D10.K4.N100k		
TS	stdEM	w/oG	SWEM	stdEM	w/oG	SWEM	stdEM	w/oG	SWEM
2	-10.436	-10.512	-10.512	-19.252	-19.276	-19.276	-47.846	-47.869	-47.869
4	-10.427	-10.446	-10.446	-19.192	-19.215	-19.215	-47.933	-48.010	-48.010
6	-10.451	-10.604	-10.716	-19.164	-19.220	-19.326	-47.702	-47.712	-47.726
8	-10.444	-10.700	-10.735	-19.188	-19.226	-19.245	-47.859	-47.884	-47.886
10	-10.439	-10.523	-10.579	-19.202	-19.247	-19.258	-47.759	-47.820	-47.873

Table 1. Average log likelihood returned by stdEM, SWEMw/oG and SWEM

Table 1 shows the results returned by our algorithms where datasets are divided into 10 time intervals, each with 10k of data points. The results are reported after each of 2 time slots. This table also presents the outputs of SWEMw/oG, a variation of SWEM [5]. It can be seen that in all cases, our SWEM and SWEMw/oG algorithms almost obtain the accuracy close to that of the stdEM. It is important to note that SWEM and SWEMw/oG process these data streams incrementally whilst stdEM clusters entire data points appearing in the sliding window at once and without any streaming constraints. It is further to observe that the clustering results of SWEM are not much different from SWEMw/oG which clearly indicating that the technique of gradually reducing weights of expiring components from global clusters works effectively in SWEM. Notice that the number of data points SWEM has to compute at the expiring phase is fixed and only $(k+m)$ while that of SWEMw/oG is $b \times m$ and dependent on the window's size b.

In Table 2, we provide more details on our algorithms where the true means (TM column) and the approximate ones of each cluster are reported for the dataset $D4.K5.N100k$ (at its last time slot). We also further add 5% of random noise to this dataset and the corresponding results are reported in the last three columns of the table. It is observed that the approximate means returned by SWEM and SWEMw/oG remain very close to those of the true clusters despite the noise appearance.

To further simulate significant changes in the stream distribution, we randomly generate two completely different distributions ($D = 4$, $K = 5$, and random noise remains 5%), each with 50k data points. Consequently, the dataset $D4.K5.N100k.AB$ is formed by merging two distributions. In the following sections, we test the sensitivity of SWEM on various parameter settings by using this dataset.

Varying Number of Micro Components: It is obvious that the number of micro components should be larger than the number of natural clusters in order to obtain a clustering of good quality. Nevertheless, a very large number of micro components is inefficient in terms of running time and memory storage since the complexity of the SWEM's first stage increases linearly with the number of micro components maintained. In order to observe how this parameter affects the accuracy of our algorithm, we run SWEM with $D4.K5.N100k.AB$ where the ratio between the number of micro components and the natural clusters is varied from 2 to 10. Figure 2 reports the experimental results where we compute the average accuracy on all time slots. From the figure we observe that if the number of micro components is close to

[5] SWEMw/oG differs from SWEM in the expiring phase where it derives the global models by simply re-clustering all sets of micro components maintained in the entire window.

		D4.K5.N100k				D4.K5.N100k with noise		
		TM	stdEM	w/oG	SWEM	stdEM	w/oG	SWEM
C1	Dim 1	-165	-165.861	-165.618	-165.930	-163.054	-161.716	-162.050
	Dim 2	281	282.269	282.597	282.797	280.152	279.594	276.758
	Dim 3	-114	-114.070	-113.65	-113.800	-112.744	-110.741	-109.878
	Dim 4	175	175.863	176.609	176.471	172.770	172.969	172.368
C2	Dim 1	124	122.275	122.365	121.860	123.955	123.915	123.539
	Dim 2	-127	-125.064	-125.412	-125.454	-115.209	-116.539	-122.902
	Dim 3	188	188.376	188.3	188.527	179.520	177.993	186.276
	Dim 4	92	91.753	91.9252	91.523	89.571	91.093	97.919
C3	Dim 1	-3	-1.918	-1.90395	-1.745	-3.732	-2.686	-2.377
	Dim 2	224	223.657	223.699	223.446	222.349	222.635	222.477
	Dim 3	-53	-52.288	-52.2454	-52.113	-52.760	-51.036	-50.682
	Dim 4	-176	-175.382	-175.045	-175.102	-175.299	-174.635	-174.607
C4	Dim 1	295	297.043	297.839	297.536	295.111	294.120	296.555
	Dim 2	155	155.647	155.704	156.406	154.611	153.671	154.623
	Dim 3	276	275.964	275.681	275.236	275.875	274.569	274.624
	Dim 4	-73	-72.912	-73.3848	-73.182	-73.159	-75.363	-77.620
C5	Dim 1	245	246.302	246.922	246.851	245.685	245.970	243.827
	Dim 2	11	10.4482	8.990	9.525	11.182	14.011	9.430
	Dim 3	-154	-152.044	-152.077	-152.012	-155.230	-153.562	-152.924
	Dim 4	153	152.852	153.947	153.555	153.428	153.834	152.462

Table 2. Clustering means on $D4.K5.N100k$ and $D4.K5.N100k$ with 5% noise

the number of global clusters, the clustering quality is poor. This is because a very small number of micro components is harder to approximate the characteristics of the stream, especially in the situation where the distribution changes considerably. A poor approximation often causes worse in later phases where the SWEM needs to remove expiring information. When the number of micro components increases, the average log likelihood increases as well. However, we realize that this value becomes stable when the number of micro components is set around $m = 6K$. This indicates that we do not need to set the number of micro components too large, yet still able to achieve a high clustering accuracy.

Varying Merging Threshold: Recall that the merging and splitting operations are invoked to re-distribute micro components when SWEM detects a significant change happened in the stream's distribution. Furthermore, since SWEM always maintains a set of m micro components at each time slot, when two components are merged, another one with biggest summation in variances will be split. We report the clustering results when the merging threshold $Avg(D_{i,j})$ is varied. Figure 3 reports our results at the time slot 6 at which the algorithm detects a significant change in the data distribution. From the figure, it is observed that when the merging threshold is either too small or too large, the average log likelihood results are poor. The reason is that when $Avg(D_{i,j})$ is set too small, there are almost no micro components being merged (although they are close to each other). In the other extreme when $Avg(D_{i,j})$ is set too large, many micro components are frequently merged and split; this causes a poor result on clustering analysis. The highest average log likelihood is achieved when this merging threshold is set to be round 1.

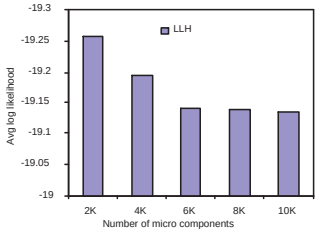


Fig. 2. Micro Components vs. Accuracy

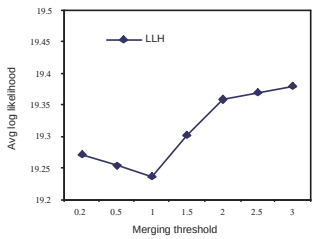


Fig. 3. Merging threshold $Avg(D_{i,j})$ vs. Accuracy

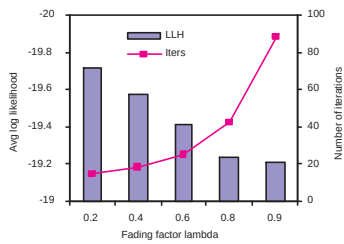


Fig. 4. Fading Factor vs. Accuracy

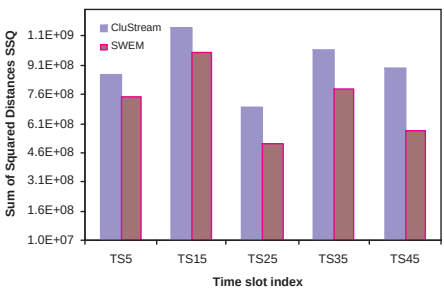


Fig. 5. Clustering quality comparison on real-world Network Intrusion dataset

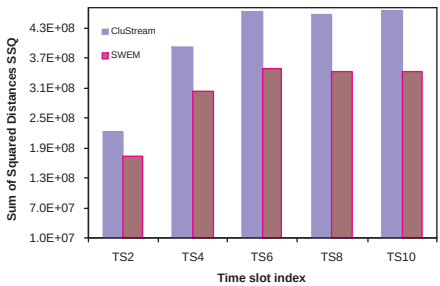


Fig. 6. Clustering quality comparison on D10.K4.N100k dataset

Varying Fading Factor: The last parameter that may impact the clustering quality of SWEM is the fading factor λ. Figure 4 shows the relationship between the fading factor, the number of iterations used in the expiring phases, and the average log likelihood quality. The results are reported at the time slot 10 where the last interval of the first distribution in $D4.K5.N100k.AB$ is eliminated from the sliding window. As expected, when λ is set closer to 1, the algorithm reduces the weight of each expiring micro component slowly and needs more iterations. Accordingly, the quality of the clustering results is better (indicating by the larger average log likelihood value). In order to achieve high quality of clustering results, the best value of λ can be set between 0.8 and 0.9. It is also worth noting that since this expiring step is executed in the second stage where SWEM works only with micro components and global clusters (which actually are only a small number of weighted data points), the large number of iterations utilizing in this stage is therefore generally acceptable.

5.3 Comparison with CluStream

As presented in Section 2, CluStream [2] is an algorithm proposed to cluster entire data streams. Nonetheless, it is possible to modify it for working in a sliding window model. Specifically, instead of storing the snapshots at different levels of granularity, we let CluStream maintain each snapshot precisely at every time slot and those that are expired from the sliding window will be deleted immediately. The time horizon in CluStream is chosen equal to the size of the sliding window. Furthermore, we

keep the factor 2 for the root mean square (RMS) deviation, as indicated in [2] this value produces the best CluStream's results.

It is also important to note that, in CluStream, only data points arriving in the first time slot are clustered until the k-means method converges. For the rest of the stream, each arriving point is clustered online by absorbing it to one micro cluster or forming itself as a new micro cluster. Hence, the convergence of CluStream is not guaranteed after the first time slot. We experimentally realize that such a process is only effective when the stream distribution is relatively stable over time. When the stream distribution remarkably changes, CluStream's results reduce considerably. Thus in the following experiments, we let CluStream run until its convergence at each time slot. This compromises the processing time but improves CluStream's clustering. We compare SWEM and CluStream on their clustering quality and execution time. [6]

In order to be consistent with CluStream, the clustering qualities of two algorithms are evaluated based on the sum of squared distance (SSQ) measure presented in [2]. Therefore, the SWEM's results need to be converted by assigning each data point to exactly one cluster to which it has the highest probability. Figures 5 and 6 report the SSQ values of two algorithms on the KDD-CUP'99 Network Intrusion Detection dataset and D10.K4.N100k (with 5% of random noise). As observed from the figures, the SSQ value of SWEM is always smaller than that of CluStream in both these real-world and artificial datasets. For example, at time slot 25 of the network intrusion dataset, the SSQ of SWEM is 35% smaller than that of CluStream. This explicitly indicates that the data points in each of the clusters obtained by SWEM are more similar and compact than those obtained by CluStream. To explain for these results, note that CluStream computes micro clusters based on Euclidean distance and it does not take into account clusters' shapes in identifying closest center for each data point. Furthermore, the hard assignment (due to using K-means) is highly sensitive to the noise and outliers since a small number of noise data can influence the computation of the clusters' means substantially [14, 18]. On the other hand, both micro clusters' centers and shapes are effectively utilized in SWEM. The using of a Mahalanobis-based distance has improved the SWEM's capability in identifying correct cluster centers. Additionally, SWEM is less affected by the noise data due to the advantage of soft assignment of the EM technique. Its approximate micro components are therefore usually produced in better quality and consequently the global clusters are also derived more accurately.

In order to provide more insights, we observe the clustering quality of two algorithms on $D4.K5.N100k.AB$ dataset. Figure 7 shows the clustering results in which we set the window's size equal to 4.[7] As observed from the figure, the SSQ values of both SWEM and CluStream are linearly increased in the first four time slots and slightly changed in the fifth one (since data are generated from the same distribution). However, the clustering quality of CluStream is remarkably worse than that of SWEM at time slot 6 and subsequent ones. This can be understood by the design of CluStream, when a new data point is determined too far from the set of micro clusters and cannot be absorbed by any, CluStream simply creates a new micro clus-

[6] The memory usage is not compared since both algorithms use fixed memory space in terms of number of weighted data points.

[7] This allows the window slide once within each distribution.

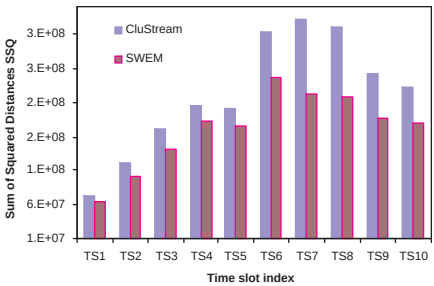


Fig. 7. Clustering quality comparison on $D4.K5.N100k.AB$

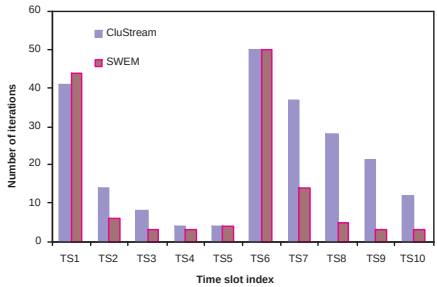


Fig. 8. Execution time comparison on $D4.K5.N100k.AB$

ter for it and merges other ones. As such, some new micro clusters have only one or a few points whereas the others have many. This causes the clusters' weights very imbalance and usually leads to poor approximation on new changing distribution. On the contrary, SWEM re-distributes the set of micro components by applying the merging and splitting operations. A large component having the highest variance sum will be split whilst two small ones will be merged if they are sufficiently close. Such operations not only *discretely* re-distribute micro components in the entire data space but also manage to make the weight of each component not much different from one another. Consequently, new evolving changes in the stream can be essentially approximated by SWEM. As in Figure 7, the SSQ value reported at the last time slot of SWEM is only $180k$ whereas that of CluStream is $233k$.

In order to evaluate the execution time of two algorithms, we continue using $D4.K5.N100k.AB$. The maximal number of iterations (until convergence) of both algorithms is set to 50 and the execution time is measured in terms of this value. Figure 8 reports the performance of two algorithms at each time slot. At the first time interval, SWEM uses a slightly more number of iterations than Clustream. For the next four ones, the number of iterations of both algorithms reduces considerably since the dataset's distribution is steady. At the critical time slot 6, the number of iterations in both algorithms reaches the maximum value due to the change in distribution. However, it is interesting to observe the results in the subsequent intervals. While SWEM's iterations reduces sharply, that number of CluStream remains very high. The main reason explained for the better performance of SWEM over CluStream is that SWEM computes a set of micro components for each time slot separately; yet it always uses the converged models of the previous time slot as the initial parameters for the next one. This approach usually makes the converged parameters to be quickly achieved if the stream distribution does not significantly change between two consecutive time slots. On the other hand, CluStream always tries to update new data points into a set of micro clusters it has maintained so far in the stream (regardless of how much the current distribution is different or similar from the past one). Consequently, CluStream often needs more time to converge even in the case the stream distribution is stable between time intervals. As observed from Figure 8, the difference in execution time of two algorithms is clearly reflected in the last four time slots of the stream.

6 Conclusions

In this paper, we have addressed the problem of clustering data streams in a sliding window, one of the most challenging mining model. We proposed SWEM algorithm that is able to compute clusters in a strictly single scan over the stream and work within confined memory space. Importantly, two techniques of splitting and merging components have been developed in SWEM in order to address the problem of time-varying data streams. The feasibility of our proposed algorithm was also verified via a number of experiments. Furthermore, SWEM has a solid mathematical background as it is designed based on the EM technique. Such a mathematically sound tool has been shown to be stable and effective in many domains despite the mixture models it employs being assumed to follow multivariate Gaussian distributions.

References

1. Charu C. A., Jiawei H., and Philip S. Yu. A framework for projected clustering of high dimensional data streams. In *VLDB conference*, pages 852–863, 2004.
2. Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *VLDB Conference*, pages 81–92, 2003.
3. Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *PODS*, pages 1–16, 2002.
4. Brian Babcock, Mayur Datar, and Rajeev Motwani. Sampling from a moving window over streaming data. In *SODA*, pages 633–634, 2002.
5. Brian Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. Maintaining variance and k-medians over data stream windows. In *PODS*, 2003.
6. Moses C., Liadan O., and Rina P. Better streaming algorithms for clustering problems. In *ACM symposium on Theory of computing*, pages 30–39, 2003.
7. Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *SDM*, 2006.
8. Yixin Chen and Li Tu. Density-based clustering for real-time stream data. In *SIGKDD Conference*, pages 133–142, 2007.
9. Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. In *SODA*, pages 635–644, 2002.
10. Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Querying and mining data streams: you only get one look a tutorial. In *SIGMOD Conference*, 2002.
11. C. Giannella, J. Han, J. Pei, X. Yan, and P.S. Yu. *Mining Frequent Patterns in Data Streams at Multiple Time Granularities*. Next Generation Data Mining. 2003.
12. S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams: Theory and Practice. *IEEE TKDE*, 15, 2003.
13. S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams. In *Proc. Symp. on Foundations of Computer Science*, November 2000.
14. Jiawei Han and Micheline Kamber. *Data mining: concepts and techniques*. 2001.
15. Liadan O., Nina M., Sudipto G., and Rajeev M. Streaming-data algorithms for high-quality clustering. In *ICDE*, 2002.
16. Nesime T., Ugur Ç., Stanley B. Z., and Michael S. Load shedding in a data stream manager. In *VLDB*, pages 309–320, 2003.
17. Naonori Ueda and Ryohei Nakano. Deterministic annealing em algorithm. *Neural Netw.*, 11(2):271–282, 1998.
18. Aoying Z., Feng C., Ying Y., Chaofeng S., and Xiaofeng H. Distributed data stream clustering: A fast em-based approach. In *ICDE Conference*, pages 736–745, 2007.
19. Yunyue Zhu and Dennis Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB Conference*, 2002.