Distributed Data Stream Clustering: A Fast EM-based Approach

Conference Paper · May 2007

DOI: 10.1109/ICDE.2007.367919 · Source: IEEE Xplore

CITATIONS
49

READS
215

5 authors, including:

Aoying Zhou
Fudan University
493 PUBLICATIONS **4,970** CITATIONS
SEE PROFILE

Feng Cao
Fudan University
13 PUBLICATIONS **813** CITATIONS
SEE PROFILE

Ying Yan
Microsoft
26 PUBLICATIONS **358** CITATIONS
SEE PROFILE

Chaofeng Sha
Fudan University
48 PUBLICATIONS **350** CITATIONS
SEE PROFILE

Distributed Data Stream Clustering: A Fast EM-based Approach

Aoying Zhou[§] Feng Cao[§] Ying Yan[§] Chaofeng Sha[§] Xiaofeng He[†‡]
[§]Fudan University [†]Lawrance Berkeley Nat'l Lab [‡]Alexa Internet
{ayzhou, caofeng, yingyan, cfsha}@fudan.edu.cn xhe@lbl.gov

Abstract

Clustering data streams has been attracting a lot of research efforts recently. However, this problem has not received enough consideration when the data streams are generated in a distributed fashion, whereas such a scenario is very common in real life applications. There exist constraining factors in clustering the data streams in the distributed environment: the data records generated are noisy or incomplete due to the unreliable distributed system; the system needs to on-line process a huge volume of data; the communication is potentially a bottleneck of the system. All these factors pose great challenge for clustering the distributed data streams.

In this paper, we proposed an EM-based (Expectation Maximization) framework to effectively cluster the distributed data streams, with the above fundamental challenges in mind. In the presence of noisy or incomplete data records, our algorithms learn the distribution of underlying data streams by maximizing the likelihood of the data clusters. A test-and-cluster strategy is proposed to reduce the average processing cost, which is especially effective for online clustering over large data streams. Our extensive experimental studies show that the proposed algorithms can achieve a high accuracy with less communication cost, memory consumption and CPU time.

1 Introduction

The rapid progress of computer and sensor network technology has made the deployment of distributed system possible. The applications range from financial stock exchange, weather or environment monitoring to telecommunication call. In these applications, it is essential to aggregate information from remote sites. Since it is prohibitively expensive to transmit a huge amount of raw data streams from the remote sites to a central processing system, clustering the data streams is a natural approach to overcome this problem. Data clustering groups together data records with similar properties such that they can be represented by the same entity, which greatly reduces the size of information needed to be transmitted.

Previous works on stream clustering mainly focused on the popular k-means-type algorithms. The k-means algorithms [14, 1] assume each data record belongs to exactly one cluster. In real streaming applications, however, a data record can belong to multiple clusters with different degree of membership.

The hard partition may lose a significant amount of information (e.g., the network connection with 80% probability to be attacked by hackers is more informative than a simple "Yes/No" answer.). The situation becomes more complicated at the presence of noisy and incomplete data records. For instance, in the P2P network, the unreliable environment can produce incomplete or corrupted web click stream data; the telecom company may have the overlapping consumer clusters due to the IP sharing; sensing in the presence of obstacles degrades the capability of sensor data fusion in distributed embedded sensor network.

One remedy is to cluster the data records with probability. This soft clustering is achieved if the cluster models are capable of learning the underlying data distribution. An effective representation of the probability density function is the mixture model consisting of multiple component models such as Gaussians. We treat the data streams as being generated by these underlying mixture models.

The EM (Expectation-Maximization) algorithm [3, 23] is an effective technique for learning the mixture model parameters (cluster parameters and their mixture weights) in the presence of incomplete data. The EM algorithm iteratively refines the model parameters, maximizes the likelihood of the clusters, terminates at a locally optimal solution.

In most distributed data stream processing, transmitting entire stream data to a central site is almost impractical, since it will dramatically increase the network communication cost. While existing distributed clustering methods mainly focus on one-shot mining, they are unsuitable for continuously clustering distributed data streams. For example, a direct extension of DBDC [15] will lead to unceasingly exchange local models between remote sites and the coordinator.

In this paper, we address the problem of clustering dis-

1-4244-0803-2/07/$20.00 ©2007 IEEE.

tributed data streams by identifying the probability density in the dense region of the data streams. We propose an EM algorithm with Gaussian mixture model to learn the underlying data distribution. Our contributions are as follows:

1. We propose a framework to soft-cluster the distributed data streams. Our EM-based method is suitable for the streams with incomplete or noisy data records.
2. We present the novel idea of test-and-cluster strategy to capture the distribution of data stream at remote site. This event-driven strategy minimizes the communication cost as long as the local distribution at remote site remains unchanged.
3. We perform extensive experiments on both synthetic and real data. The theoretical analysis and the experimental results show the space and time efficiency, as well as the effectiveness of our framework.

The remainder of this paper is organized as follows. Section 2 surveys the related work. We introduces the preliminaries on Gaussian model and EM algorithm in Section 3. Section 4 describes clustering distributed data stream problem. Section 5 presents our framework of clustering distributed data streams. In section 6, we present our performance study. Section 7 extends our framework to deal with some related problems. We concludes the paper in Section 8.

2 Related Work

Previous work on clustering data streams focused on developing space-efficient and one-pass algorithms for centralized, one-shot clustering on massive data streams. Examples include k-means based divide and conquer algorithm [4, 13, 14], evolving cluster analysis [1, 2] and multi-data-streams clustering [9, 24]. The above streaming clustering algorithms are often k-means based algorithms, in which each data record belongs to exactly one cluster. When the cluster boundaries overlap, this simplified approach may lose significant amount of useful information. The incomplete data records also pose serious problem to this approach.

Soft clustering algorithms have been proved to be able to improve the clustering solutions[12]. The EM based algorithms and fuzzy algorithms are proposed to deal with overlapping clusters, such as SMEM[23], SEM[6]. Several distributed clustering methods, e.g. DBDC [15], center-based [11], density-based [16] and model-based [18] clustering have been proposed to deal with one snapshot clustering. However, their underlying assumption is that the computation is triggered periodically in response to one-shot mining. They are not applicable for streaming cases where continuous mining is required.

To reduce the communication cost, clustering on individual nodes was investigated in DEM [20]. DEM assumes that the data records have the same number of common (unknown) distributions on every node. The learnt model parameters are passed to the next node based on the prespecified order. This communication is necessary due to the assumption of the same distributions on all computing nodes. Our approach, on the other hand, assumes any number of distributions which can be potentially different on individual nodes.

There are some other works on distributed data streams, including non-holistic aggregation [21], set-expression cardinalities [10], top-k [5], and holistic aggregation [7]. Different from these basic statistics, our clustering problem over distributed data streams is a much more complex mining problem.

3 Preliminaries

3.1 Gaussian mixture model

The Gaussian distribution of a cluster j is parameterized by a d-dimensional mean vector μ_j and a $d \times d$ covariance matrix Σ_j:

$$p(x|j) = \frac{1}{\sqrt{(2\pi)^d|\Sigma_j|}} \exp\{-\frac{1}{2}(x-\mu_j)^T\Sigma_j^{-1}(x-\mu_j)\}.$$

For K components, the probability density function of the Gaussian mixture is

$$p(x) = \sum_{j=1}^{K} w_j p(x|j) \quad (1)$$

with w_j being the weight of cluster j in the Gaussian mixture model. The Gaussian mixture model can be represented by the parameter set (w_j, μ_j, Σ_j), $j = 1, ..., K$.

Given record x, the probability of x belonging to cluster j (*posterior*) is

$$Pr(j|x) = \frac{w_j p(x|j)}{p(x)}. \quad (2)$$

Note that any distribution can be effectively approximated by a mixture of Gaussian models.

3.2 Classical EM algorithm

The EM algorithm is an iterative optimization method to learn the parameters of the fitting model. The algorithm first calculates an optimal lower bound according to current value of model parameters ("expectation-step" or E-step), then maximizes this bound to obtain an improved estimation ("maximization-step" or M-step). The process is repeated

until the difference of the log likelihood between two consecutive iterations is below a threshold. The log likelihood of the model is non-decreasing at each iteration [3].

The Gaussian mixture model based EM algorithm consists of the following steps:

1. Initialization: Initialize the iteration step $i = 0$ and the mixture model parameters $(w_j^i, \mu_j^i, \Sigma_j^i)$, $j = 1, ..., K$.
2. Repeat
   (a) E-Step. For each record x, compute the membership probability of x in each cluster: $Pr(j|x) = \frac{w_j^i \cdot p(x|j)}{p(x)}$, $j = 1, ...K$.
   (b) M-step. Update the model parameters: $w_j^{i+1} = \frac{1}{|D|}\sum_{x\in D} Pr(j|x)$, $\mu_j^{i+1} = \frac{\sum_{x\in D} x \cdot Pr(j|x)}{\sum_{x\in D} Pr(j|x)}$, $\Sigma_j^{i+1} = \frac{\sum_{x\in D} Pr(j|x)(x-\mu_j^{i+1})(x-\mu_j^{i+1})^T}{\sum_{x\in D} Pr(j|x)}$, $j = 1, ..., K$. D denotes the data set.
   (c) Compute the log likelihood: $L^{i+1} = \sum_{x\in D} \log p(x)$.

   Until $|L^i - L^{i+1}| \leq \varpi$, a user-defined parameter.

4 Problem Statement

Consider a distributed-computing environment with r remote sites and a coordinator site. On each remote site r_i, data stream S_i arrives continuously. The coordinator site accepts user mining request and generates the mining results over the union of all data streams $S = S_1 \cup S_2 \cup \ldots \cup S_r$. In the distributed architecture introduced in [5, 7, 10, 21], there is no direct communication between the remote sites. Each remote site exchanges information with the coordinator only. This communication model covers a lot of real applications.

Similar to [6], we model each cluster by a d-dimensional Gaussian probability distribution. EM algorithm is applied to train the Gaussian models. Unlike [20], we do not assume the constant number of component models for the data stream, nor do we assume the same distribution across the remote sites. Rather, new model is added to the model list if the data does not fit current models.

The average log likelihood is used to evaluate how the data fit into the current model. We define the average log likelihood below.

Definition 1: The average log likelihood of the model, Avg_{Pr}, is defined as

$$Avg_{Pr} = \frac{1}{|D|}\sum_{x\in D} \log(\sum_{j=1}^{k} w_j \cdot p(x|j)). \quad (3)$$

Before going to the details, we first present some analysis which form the theoretical basis of our algorithm.

4.1 Theoretical Basis

Lemma 1 *Assume random variable x follows normal distribution $N(0, \frac{1}{M})$, then $Pr(x \geq \epsilon) \leq 1 - \sqrt{1 - \exp(\frac{-M\epsilon^2}{2})}$, where $M > 0$.*

Proof (*sketch*) Let$I = \int_{-\epsilon}^{\epsilon} \sqrt{\frac{M}{2\pi}} exp(\frac{-Mt^2}{2})dt$, then $I^2 = \int_{-\epsilon}^{\epsilon}\int_{-\epsilon}^{\epsilon} \frac{M}{2\pi} exp(\frac{-M(s^2+t^2)}{2})dsdt$ Let $s = \rho\cos\theta$, $t = \rho\sin\theta$, $I^2 \geq \int_0^{2\pi} d\theta \int_0^{\epsilon} \frac{M}{2\pi}\exp(\frac{M\rho^2}{2})\rho d\rho = 1 - \exp(\frac{-M\epsilon^2}{2})$. Since $0 \leq I \leq 1$, $Pr(x \geq \epsilon) = 1 - I \leq 1 - \sqrt{1 - \exp(\frac{-M\epsilon^2}{2})}$ holds. □

Theorem 1 *For any Gaussian distribution $N(\mu, \Sigma)$, $\mu = [\mu_1, \mu_2, \cdots, \mu_d]^T$, if the sample size M satisfies $M \geq \frac{-2d\ln(\delta(2-\delta))}{\epsilon}$, then the squared Mahalanobis distance from the sample mean $\overline{x}$ to the expectation μ, $ad = (\overline{x} - \mu)^T\Sigma^{-1}(\overline{x} - \mu)$, is less than ϵ with probability larger than $1 - \delta$.*

Proof (*sketch*) Since any covariance matrix is positive semidefinite, there exists an orthogonal matrix U satisfying $U^T\Sigma U = D$, where $D = diag(\sigma_1^2, \sigma_2^2, \cdots, \sigma_d^2)$ and $U^{-1} = U^T$. Therefore*, $ad = (\overline{x} - \mu)^T\Sigma^{-1}(\overline{x} - \mu) = (U(\overline{x} - \mu))^T D^{-1}(U(\overline{x} - \mu)) = \sum_{i=1}^{d}(\frac{U(\overline{x_i} - \mu_i)}{\sigma_i})^2$. Since the random variable $\frac{U(x_i - \mu_i)}{\sigma_i}$ follows normal distribution $N(0, 1)$, thus $\frac{U(\overline{x_i} - \mu_i)}{\sigma_i}$ follows $N(0, \frac{1}{M})$. According to Lemma 1, $Pr(\frac{U(\overline{x_i} - \mu_i)}{\sigma_i} \geq \sqrt{\frac{\epsilon}{d}}) \leq 1 - \sqrt{1 - \exp(\frac{-M\epsilon}{2d})} \leq \delta$. Therefore, $Pr(ad \geq \epsilon) \leq \delta$ holds. □

Theorem 2 *If data sets D_1 and D_2 have the same distribution and $M = |D_1| = |D_2| = \frac{-2d\ln(\delta(2-\delta))}{\epsilon}$, then under the identical mixture models, the difference between the average log likelihoods of D_1 and D_2, $|Avg_{Pr_1} - Avg_{Pr_2}|$, is less than ϵ with high probability when δ is small.*

Proof (*sketch*) For any record x, its probability density in any mixture of Gaussians distribution should be greater than 0 and less than 1, and its probability density in each Gaussian distribution is also greater than 0 and less than 1. Therefore,

$$|Avg_{Pr_1}| = \frac{1}{M}|\sum_{x\in D} \log(\sum_{j=1}^{k} w_j \cdot p(x|j))|$$
$$\leq \frac{1}{M}|\sum_{x\in D}\sum_{j=1}^{k} \log(w_j \cdot p(x|j))|$$

*The determinant of the covariance matrix equals zero, when one attribute has zero variance or two attributes are identical or multiples of them. We can exclude these situations from consideration. This is a sound assumption taken by most modelling methods.

$$= |(\log(w_j) - \frac{d}{2}\log(2\pi) + log(|\Sigma|))$$
$$+ \frac{1}{2}\sum_{j=1}^{k}(\overline{x} - \mu^j)^T\Sigma_j^{-1}(\overline{x} - \mu^j)|$$

From Theorem 1, $Pr((\overline{x} - \mu)^T\Sigma^{-1}(\overline{x} - \mu) \geq \frac{\epsilon}{2}) \leq \exp(\frac{-M\epsilon^2}{2}) = \delta$. D_1 and D_2 have the same distribution, i.e., they have the same (w_j, μ_j, Σ_j), $j = 1, ..., k$. For any cluster j, we can expect D_1 and D_2 share the same number of records which belong to cluster j with maximal probability. In order to sharp average log likelihood based test, for each record x, we use the maximal probability of x belongs to one of the clusters instead of the overall probability of x under the mixture model.

Therefore, $|Avg_{Pr_1} - Avg_{Pr_2}| \leq |Avg_{Pr_1}| - |Avg_{Pr_2}| \leq \epsilon$ with high probability when δ is small. □

From the theorems above, we can see that ϵ acts as the error bound for the difference of average log likelihood. δ controls the probability of the error. Both parameters are used to determine the data chunk size M.

5 Framework of Clustering Distributed Data Streams

In this section, we discuss in detail our our clustering approach on distributed data streams. We refer to it as the CluDistream . The CluDistream consists of two parts: remote site processing and coordinator processing. Global time is assumed in the distributed environment.

5.1 Remote Site Processing

The basic idea is test-and-cluster. We first test whether the incoming data fit the current model or not. Clustering is performed only after the test fails. Since clustering is more expensive, this strategy will greatly reduce the average processing cost without sacrificing the accuracy.

The remote site maintains a list of models, each of which with a unique model ID. An event table is kept to record the evolving behavior of the stream. Each entry of the table is a <start time, end time, model ID> triplet.

5.1.1 Algorithm Description

Based on Theorem 1, we conceptually divide the incoming data stream into chunks of size $M = \frac{-2d\ln(\delta(2-\delta))}{\epsilon}$. Each model is associated with a counter c recording the number of data records fitting the model.

The very first chunk is always clustered using the EM algorithm to get current mixture model: (w_j, μ_j, Σ_j), $j = 1, ..., K$. Counter c is set to M.

Algorithm 1 ProcessingSubStream (S_i, ϵ, δ)

```
1: M = -2d ln(δ(2-δ)) / ε, chunk number n = 1;
2: Apply EM clustering to chunk 1 of S_i, obtaining model
   parameters w[j], μ[j], Σ[j], j = 1, ..., K, c = M;
3: chunk number n = 2
4: for Each chunk n do
5:    if (FitDistribution(chunk n, w[j], μ[j], Σ[j])) then
6:       c = c + M;
7:    else
8:       Insert (w, μ, Σ) and c into model list;
9:       Append a new entry <current model ID, start
         chunk, n - 1 > into event list;
10:      Apply EM clustering to chunk n, update w, μ, Σ,
         c = M;
11:   end if
12: end for
```

Each following chunk n is tested first to determine whether it fits the current model or not. If does, we simply update the counter of current model as $c = c + M$. Otherwise, we insert current model and its counter into the model list, cluster chunk n using EM algorithm, and obtain the new model parameters (w_j, μ_j, Σ_j), $j = 1, ..., K$, along with the log likelihood Avg_{Pr_0} of the new model. The counter of the new model is set to M. Algorithm 1 shows the detailed process.

5.1.2 Test strategy

The incoming data stream is tested only once using current model. It is suitable for the case where the data streams are generated by different distributions, for instance, by alternating models. There is a tradeoff between the maximal number of tests and efficiency. The efficiency may decrease with the increase of the maximal number of tests. The maximal number of tests is determined by parameter c_{max}. Our experiments show that $c_{max} = 3$ or 4 is often a good choice.

Test Criterion We calculate Avg_{Pr_n}, the average log likelihood of chunk n under current model. Then we compare the difference between Avg_{Pr_n} and Avg_{Pr_0}, the average log likelihood of current model.

$$J_{fit} = |Avg_{Pr_n} - Avg_{Pr_0}| \quad (4)$$

If $J_{fit} \leq \epsilon$, based on Theorems 2, we claim that the data chunk n fits current model. Otherwise, it has different distribution, hence EM clustering need to be applied.

5.1.3 Computation cost

Theorem 3 *The memory consumption of Algorithm CluDistream on each remote site is* $O(\frac{-2d\ln(\delta(2-\delta))}{\epsilon} + BK(d^2 + d + 1))$, *where* B *is the number of different distributions of the evolving data stream.*

Proof The memory consumption incudes two parts. The first part is the data buffer storing the new records. According to Theorem 1, it is $\frac{-2d\ln(\delta(2-\delta))}{\epsilon}$. The second part is the space for recording the parameters of the mixture Gaussian models. For a Gaussian mixture model of K components, the memory consumption is K weights (or priors) w_j, K d-dimensional means and K $d \times d$ covariance matrixes (For diagonal Gaussians, the covariance can be represented by a d-dimensional vector). Therefore, the memory consumption of Algorithm CluDistream on each remote site is $O(\frac{-2d\ln(\delta(2-\delta))}{\epsilon} + BK(d^2+d+1))$. □

Theorem 4 *Let the cost of clustering a data set be C, then the cost of checking distribution is λC, $0 < \lambda < 1$. The average processing cost of CluDistream on one remote site is $(P_d + \lambda(1-P_d))C$, where P_d is the probability of generating data stream with new underlying distribution.*

Many natural and social systems exhibit power law characteristic[22]. As the number of events grows, their distribution converges to the steady state $p(y) = \beta y^{-q}$ with $q = 1/(1-\gamma)$ where γ is the average growth rate of events. The expected value of the distribution is $P_d = \frac{\beta}{2-q}$, often less than 0.1, meaning that the probability of emerging new underlying distribution is less than 10%. When a burst of data, often from similar distribution, arrive, then the cost reduction will be more obvious.

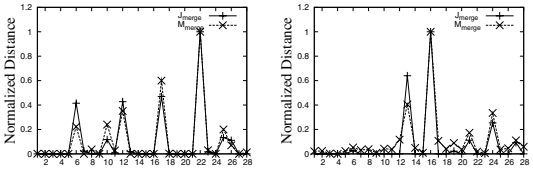
5.2 Coordinator Processing

Coordinator maintains the models from all r remote sites. One simple procedure at the coordinator is to combine all Gaussian models from each site directly, and the $r * K$ Gaussian mixture model components represent the distribution of overall distributed data streams. Reasonable for small r, it is not scalable if the problem involves a large number of streams, which is often the case in practical scenario (e.g., sensor network consisting thousands of nodes). More importantly, the local maxima pose a problem if there are too many components in a Gaussian mixture model.

5.2.1 Merge

Merging techniques were investigated to avoid local maxima. Ueda *et al.* proposed an effective method SMEM [23] to deal with the local maxima problem of EM algorithms in a centralized situation. Our merging strategy is inspired by SMEM with the difference in that it does not involve raw data transmission.

In general, when two components result in almost equal posterior probability for many data records, it can be thought that these two components might be merged. Based on this observation, SMEM [23] introduces the J_{merge} criterion:



(a) NFD (b) Synthetic

Figure 1. Merge Criterion J_{merge} vs. M_{merge}

$$J_{merge}(i,j) = \sum_{x \in D} Pr(i|x) * Pr(j|x)$$

. Since it is prohibitive to transmit the entire raw data to the coordinator, Instead of using J_{merge}, we define the following new merge criterion on the coordinator:

$$M_{merge}(i,j) = \frac{1}{(\mu_i - \mu_j)^T(\Sigma_i^{-1} + \Sigma_j^{-1})(\mu_i - \mu_j)} \quad (5)$$

Two components with large M_{merge} should have higher opportunity to be merged since they have small Mahalanobis distance between their means. In fact, this criterion can also be derived by the sum of the KL distances between these two components from two directions.

To compare J_{merge} and M_{merge}, we adopt NFD real data set which consists of the net flow data from Shanghai Telecom Co. Ltd. The data has six attributes: source host, destination host, source TCP port, destination TCP port, packet count and number of data bytes.

With 8 component models, we compute M_{merge} and J_{merge} for the 28 combinations. We normalize the M_{merge} as follows: $M_{merge} = \frac{m_{mg} - min(\ m_{mg})}{max(\ m_{mg}) - min(\ m_{mg})}$. similar process is done to J_{merge}.

Figures 1(a) and (b) show the comparison result on the NFD and the synthetic data, respectively. M_{merge} curve and J_{merge} curve are very similar. We believe that M_{merge} is a sufficiently good replacement for J_{merge}.

Define the accuracy loss function $l(x)$ as L_1 distance between the density function of new component i' and the sum of those of components i and j:

$$l(x) = \int |w_i p(x|i) + w_j p(x|j) - (w_i + w_j)p(x|i')|dx$$

After merging two components i and j with the largest M_{merge}, we hope to reduce the accuracy loss. We obtain the parameters of new component i' by minimizing $l(x)$. Since we do not know the derivatives of $l(x)$, downhill simplex method [19] is used to find the minimum, which is a fast multidimensional minimization method with low computer burden. The new component i' is called the father of components i and j.

Algorithm 2 OnUpdates (r_i)

1: **if** (remote site r_i updated) **then**
2: **for** (Each component m_j of remote site r_i) **do**
3: Calculate the $M_{split}(m_j)$;
4: **if** ($M_{split}(m_j) > \frac{1}{M_{merge}(m_j)}$) **then**
5: Split component m_j from its father component;
6: **end if**
7: **end for**
8: **for** (Each split component m_j) **do**
9: **for** (each mixture model Mix_s) **do**
10: Calculate and maintain the largest $M_{remerge}(m_j)$ of m_j and Mix_s;
11: **end for**
12: Merge m_j into the Mix_s with the largest $M_{remerge}(m_j)$;
13: **end for**
14: **end if**

5.2.2 Action on Updates

When some remote sites update their models, the coordinator may need to refine existing Gaussian mixture models. The coordinator maintains a tree structure to record the hierarchy of r Gaussian mixture models along with K component models from each remote site. The i-th component model corresponding to the updated models of remote sites needs to be checked to determine whether to split from its father or not. We define the split criterion as follows:

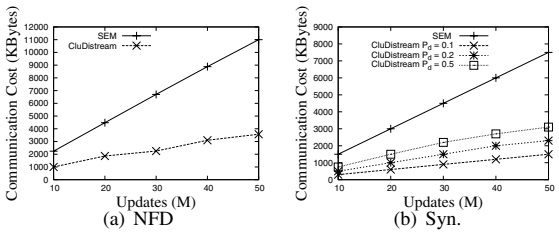
$$M_{split}(i, Mix) = (\mu_i - \mu_{Mix})^T(\Sigma_i^{-1} + \Sigma_{Mix}^{-1})(\mu_i - \mu_{Mix}) \quad (6)$$

where (u_{Mix}, Σ_{Mix}) corresponds to the Gaussian mixture model Mix (i.e., the father of component i). The component with larger M_{split} should have larger chance to be split. Then the split component should be re-merged into the Gaussian mixture model. We use the criterion below for the re-merge:

$$M_{remerge}(i, Mix) = \frac{1}{(\mu_i - \mu_{Mix})^T(\Sigma_i^{-1} + \Sigma_{Mix}^{-1})(\mu_i - \mu_{Mix})}$$

Again, we calculated the Mahalanobis distance between the means of component i and each Gaussian mixture model Mix_s (Mix_s is the sibling component of Mix), merge component i into the Gaussian mixture model with the smallest $M_{remerge}$.

Note that $M_{split}(i, Mix) = \frac{1}{M_{remerge}(i,Mix)}$, when component i is merged into a Gaussian mixture model Mix, we maintain its $M_{remerge}(i, Mix)$ value. After the component is updated, we calculate the $M_{split}(i, Mix)$ and compare the $M_{split}(i, Mix)$ with $\frac{1}{M_{remerge}(i,Mix)}$. If $\frac{1}{M_{remerge}(i,Mix)} \leq M_{split}(i, Mix)$, there is no need to split the component from the mixture Gaussians. Otherwise, we should split the component. Algorithm 2 shows the details.



(a) NFD (b) Syn.

Figure 2. Communication cost

5.3 Communication Cost Reduction

In distributed stream applications, the communication is often the bottleneck of the system due to the limited bandwidth. Similar to [7], our framework addresses the communication cost reduction in following three aspects.

1. Synopsis-based information exchange. The remote site transmits to the coordinator the *cluster model parameters* only. In the multi-test strategy, the remote sites may need to send weight update message to the coordinator.

2. Stability. There is no communication between the remote sites and the coordinator if the distribution at remote sites remains approximately the same.

3. Minimal global information exchange. As the number of remote sites increases, there still exists considerable communication cost. It becomes worse in the solution where the global cluster information needs to be exchanged, such as regularly collecting or broadcasting information between the remote site and coordinator. Our solutions explicitly avoid such expensive operation.

6 Experimental Results

All experiments are conducted on a 2.8 GHz PentiumIV PC with 512MB memory running Windows 2000 professional. For easier control of experiments, we use a discrete event simulation package C++Sim to simulate the distributed processing effect. In each experiment, the total communication cost is collected every second.

The real data used is the NFD data set. We normalize each attribute to reduce the data range effect of different attributes. Synthetic data sets are generated to test the scalability of our algorithms. The data records in each synthetic data set follow a series of Gaussian distributions. To reflect the evolution of the stream data over time, we generate new Gaussian distribution for every 2K points by probability P_d. To test our algorithm's capability of modeling the unprecise data. we generate some synthetical data with noise. The cluster quality is evaluated by the average log likelihood of the result model.

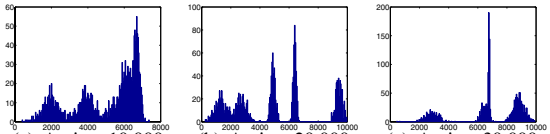

(a) At time 10,000 (b) At time 20,000 (c) At time 30,000

Figure 3. Histogram of synthetic data

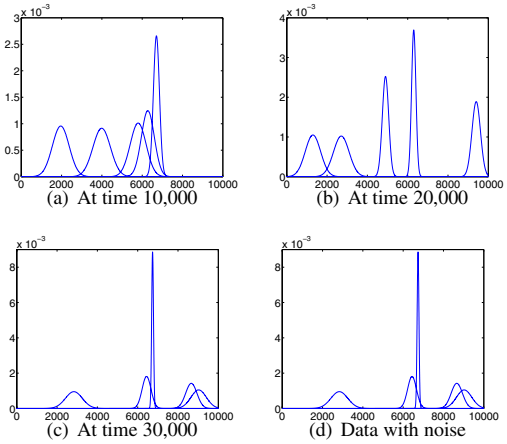

(a) At time 10,000 (b) At time 20,000 (c) At time 30,000 (d) Data with noise

Figure 4. Clustering results

We compare the CluDistream with the scalable EM (SEM) on both synthetic and real data sets. Both algorithms are implemented in Microsoft Visual C++.

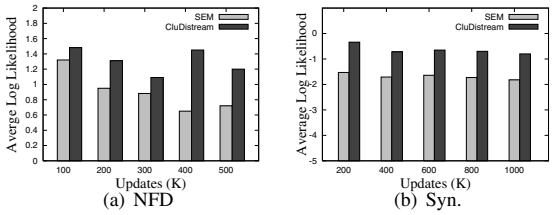
Unless mentioned otherwise, we use the following parameter setting in our experiments: probability error bound $\delta = 0.01$, error bound on average log likelihood $\epsilon = 0.02$, dimension $d = 4$, number of clusters $K = 5$, probability of new distribution $P_d = 0.1$, number of remote sites $r = 20$, maximal number of tests $c_{max} = 4$ and updates$= 100k$.

6.1 Communication cost analysis

We compare the CluDistream with a simple strategy of periodically reporting current models generated by SEM on each remote site. This strategy is adopted by many distributed clustering methods, such as DBDC [15]. Figure 2(a) shows that the communication cost of CluDistream increases at lower rate than SEM, especially after a number of updates when the model has learned the distribution. We also compare them on the synthetic data. Figure 2(b) shows similar trend. As the probability P_d increases from 0.1 to 0.5, the communication cost of CluDistream increases, but still much less than that of SEM.

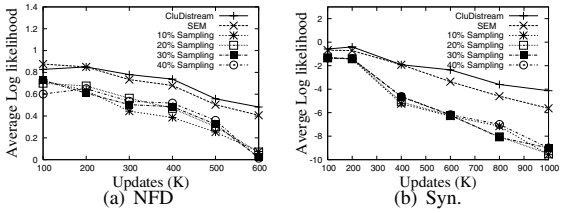
6.2 Clustering quality evaluation

We compare the clustering quality of CluDistream with SEM on remote sites and the coordinator, respectively. In order to make the results more accurate, we run each algorithm five times and compute their average log likelihood of the resulting models.



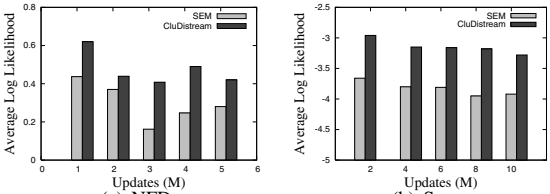
(a) NFD (b) Syn.

Figure 5. Cluster quality in horizon



(a) NFD (b) Syn.

Figure 6. Cluster quality in a landmark window



(a) NFD (b) Syn.

Figure 7. Cluster quality in coordinator

On remote site

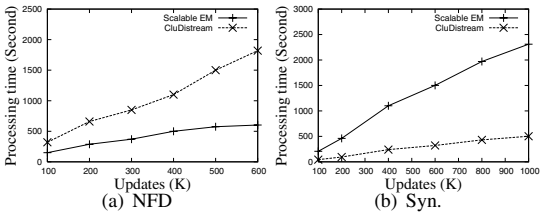
The CluDistream is capable of clustering the data already being seen (landmark window) and the data in a horizon of current time (sliding window [4]), but SEM can only generated one data model at a time. We compare these two algorithms in horizons and in land mark windows, respectively.

Horizons To simplify the visualization of clustering, we use one dimensional synthetic data. Figures 3(a), (b) and (c) show the histogram of the data set in horizon $H = 2k$ at three different time points. Figures 4(a), (b) and (c) show the corresponding CluDistream results.

We further add 5% random noise to synthetic data set. Figures 4(d) shows CluDistream captures the same model as Figures 4(c) in noisy environment.

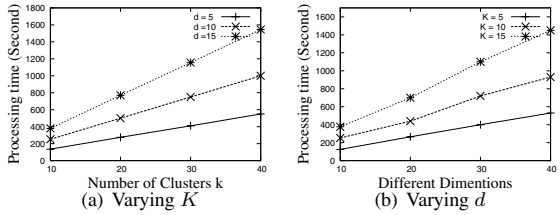
Figure 5 shows the clustering results of the updates in a horizon at different time points. Clearly CluDistream outperforms SEM. This is because CluDistream can effectively learn different distributions from the data chunks, whereas SEM tries to fit data chunks from different distributions into one model, inevitably reducing the clustering quality.

Landmark windows Figure 6 shows the clustering quality in a landmark window. It can be seen that CluDistream



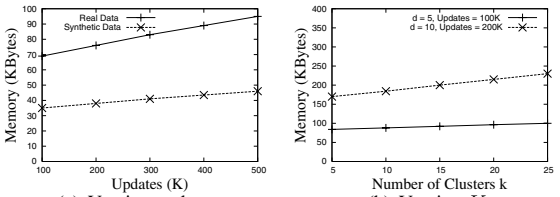
(a) NFD (b) Syn.

Figure 8. Processing time vs. updates



(a) Varying K (b) Varying d

Figure 9. Processing time vs. K and d



(a) Varying updates (b) Varying K

Figure 10. Memory

has highest average log likelihood, slightly better than SEM and much better than sampling based EM algorithms, because the test-and-cluster strategy captures the distribution of data streams more precisely than the compress based method in SEM. The CluDistream clearly outperforms the sampling based methods, since the sampling may lose a lot of valuable clustering information.

On Coordinator site

Since the SEM does not support distributed computing, in order to simulate its performance over all updates in distributed data streams, we apply the SEM to all the updates directly in a centralized environment. Figure 7(a) shows the quality comparison of CluDistream and SEM in a small horizon H=50,000 on the NFD data stream. It can be seen that CluDistream has better clustering quality even compared with a centralized SEM algorithm. We also test the clustering quality over synthetic data set in a larger horizon H=400,000. Figure 7(b) shows similar result.

6.3 Scalability Results

We test the scalability of CluDistream against the SEM algorithm on the remote site.

Processing Time

Varying updates Figure 8(a) shows the processing time for NFD data set. We can see that the processing time for both CluDistream and SEM grows linearly as the stream proceeds, while CluDistream is more efficient. It takes CluDistream less than 1 second to process 1,000 updates, while the best processing rate of SEM is less than 400 updates per second. Figure 8(b) shows that CluDistream is also more efficient for the synthetic data set. We further apply CluDistream to synthetic data with varying probabilities of generating new distributed data. Figure 14 shows that the processing time of CluDistream increases slowly when P_d is small. When P_d becomes large, e.g. P_d=1, the processing time increases dramatically. According to the power law theorem, however, in real applications it is unlikely for every new data chunk to have many different distributions.

Varying dimension d and cluster number K To obtain the data set with any number of natural clusters and dimensions, synthetic data sets are used in these tests.

The first series of data sets are generated by varying the cluster number K from 10 to 40, while fixing the updates$=100k$ and d. Figure 9(a) shows that CluDistream is of linear processing time, in proportion to the cluster number K.

Another series of data sets were generated by varying d from 10 to 40, while fixing K and the updates$=100K$. Figure 9(b) shows that CluDistream scales linearly with respect to dimensionality.

Memory usage on remote site

Varying updates Figure 10(a) shows that the memory usage of CluDistream grows slowly as the data streams proceed. For example, for NFD data stream, when the updates change from 100,000 to 500,000, the memory usage only increases by $10k$ Bytes.

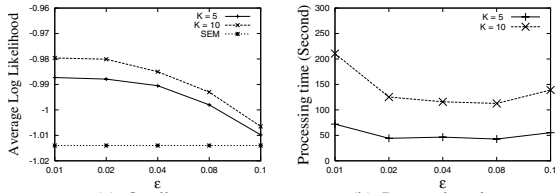
Varying cluster number K Figure 10(b) shows that the memory consumption is linear in K. The larger the K, the more memory used to store the parameters of the Gaussian models. Note that the slopes of the lines are different. The larger the d, the higher slopes of the lines, because larger d requires more memory to maintain the parameters.

6.4 Parameter Sensitivity

Varying ϵ An important parameter of CluDistream is the error parameter ϵ. It controls the tolerance of different average log likelihoods as well as the chunk size to be processed. Therefore, it affects both the efficiency and the effectiveness of CluDistream.

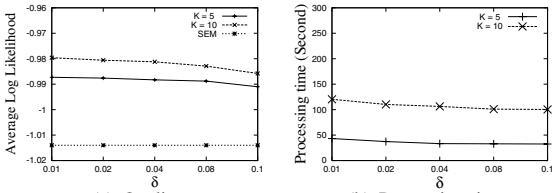
In our pervious experiments, we set ϵ to 0.02. We also test the clustering quality on synthetic data by varying ϵ from 0.01 to 0.1. Figure 11(a) shows that when ϵ becomes large, the clustering quality decreases greatly. However, the quality of CluDistream is still higher than that of SEM, always above -1.01.

Figure 11(b) shows the processing time on synthetic data by varying ϵ from 0.01 to 0.1. When ϵ is a moderate value,



(a) Quality (b) Processing time

Figure 11. Varying ϵ



(a) Quality (b) Processing time

Figure 12. Varying δ

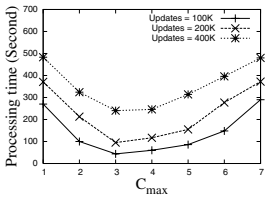


Figure 13. Processing time vs. c_{max}

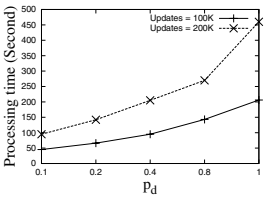


Figure 14. Processing time vs. P_d

e.g., $\epsilon = 0.04$, the processing time is small. If ϵ is set to be either small or large, the processing time becomes large. Generally, the larger the ϵ, the more time spend on processing each data chunk. The smaller the ϵ, the more time spend on processing more data chunks.

Varying δ Another important parameter is the probability error δ. Figure 12(a) shows the clustering quality of CluDistream when δ varies from 0.01 to 0.1 on synthetic data. If δ is between 0.01 and 0.04, the clustering quality is high. However, if δ becomes larger, such as 0.1, the quality deteriorates greatly, since the algorithm may merge chunks of different distributions with higher probability. Still, the CluDistream outperforms SEM in these cases.

It can be seen from Figure 12(b) that the processing time of CluDistream decreases as δ increases, since the size of data chunks is less sensitive to δ. A larger δ allows a larger tolerance in probability error, thus speeds up the execution at the cost of the quality.

Varying c_{max} The parameter c_{max} determines the maximal number of tests. Figures 13 shows the processing time of CluDistream by varying c_{max} from 1 to 7 on synthetic data set. It can been seen that, when $c_{max} = 3$ or 4, the minimal processing time is achieved. The efficiency decreases when c_{max} is either too low or too high. When c_{max} is low, the current chunk may not be tested against the models in the model list. On the other hand, large c_{max} results in too much time spent on testing the fitness of current chunk to existing models, which also takes a lot of CPU time.

7 Further Discussion

Change Detection and Evolving Analysis In the distributed system, the underlying process generating the data stream can change over time. Quantifying and detecting such change is one of the fundamental challenges in data stream settings [8, 17]. Model fitting approach provides an alternative way for change detection. A change emerges when new chunk does not fit the existing models.

The CluDistream framework can also be extended to evolving analysis using the event list. Users input a start time and a window size they are interested. By searching the event list, the algorithm presents a series of Gaussian mixture models to reflect the evolving process of data stream within that window. Previous efforts such as CluStream [1] often adopt a static strategy to record the evolving data stream. When a pyramid time arrives, a snapshot of current cluster model (micro-clusters) is stored. This strategy may introduce redundant records, while missing some important events. The novel events-driven maintenance mechanism in our method provides an adaptive way. A new model is created whenever a new distribution emerges. The absolute error between the user query window and our result window is half of the chunk size $M/2 = \frac{-d\ln(\delta(2-\delta))}{\epsilon}$.

Landmark Windows and Sliding Windows The CluDistream directly fits landmark window scenarios where only insertion exists. In sliding window settings[4], however, the data stream often has both insertion and deletion. With the events list maintained on the remote sites, our framework can be applied to such applications. In the presence of deletion, we can upload the model ID with negative weight corresponding to data deletion. The coordinator subtracts the weight of the related model in its Gaussian mixture models accordingly. The model is deleted from the model list if its weight becomes non-positive.

Multi-Layer Network A more complex and general distributed streams scenario is the tree-structured hierarchy of the communication network. By running the CluDistream between each internal node and its children, we can compute the Gaussian mixture model over the union of streams on the leaf nodes. Each internal node clusters the streams of its children, then uploads the summary information to the parent if its locally-observed Gaussian mixture model changes.

8 Concluding Remarks

In this paper, we presented a novel framework for clustering distributed data streams containing noisy or incomplete data records. It combines the advantage of Gaus-

sian mixture model based EM algorithm with the test-and-cluster strategy. Our extensive experiments show the effectiveness of our algorithm in clustering the distributed data streams. Comparing with SEM, it has less communication cost, memory consumption and CPU time, but higher clustering quality in terms of average log likelihood.

Our framework can be extended to other related problems, such as evolving analysis, change detection and sliding window clustering. It can also be extended to cluster the distribution streams in a tree-structured network system.

Our future plan is to further improve the efficiency of CluDistream, e.g., constructing index structure to accelerate merge and split based on the mixture models.

Acknowledgement

We like to thank Mr Jianlong Chang for providing the real data and the anonymous reviewers for their comments.

References

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *Proc. of VLDB*, 2003.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for projected clustering of high dimensional data streams. In *Proc. of VLDB*, 2004.

[3] A.P.Dempster, N.M.Laird, and D.B.Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal statistical Society, Series B*, 39(1):1–38, 1977.

[4] B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proc. of PODS*, 2003.

[5] B. Babcock and C. Olston. Distributed top-k monitoring. In *Proc. of SIGMOD*, 2003.

[6] P. Bradley, C. Reina, and U. Fayyad. Clustering very large databases using em mixture models. In *15th International Conference on Pattern Recognition (ICPR'00)*, 2000.

[7] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *Proc. of SIGMOD*, 2005.

[8] G. Cormode and S. Muthukrishnan. What's new: Finding significant differences in network data streams. In *Proc. of INFOCOM*, 2004.

[9] B. Dai, J. Huang, M. Yeh, and M. Chen. Clustering on demand for multiple data streams. In *Proc. of ICDM*, pages 367–370, 2004.

[10] A. Das, S. Ganguly, M. Garofalakis, and R. Rastogi. Distributed set-expression cardinality estimation. In *Proc. of VLDB*, 2004.

[11] G. Forman and B. Zhang. Distributed data clustering can be efficient and exact. *SIGKDD Explorations*, 2(2):34–38, 2000.

[12] C. Glymour, D. Madigan, D. Pregibon, and P. Smyth. Statistical themes and lessons for data mining. *Data Mining and Knowledge Discovery*, 1(1), 1997.

[13] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams:theory and practice. In *IEEE TKDE*, pages 515–528, 2003.

[14] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data stream. In *Proc. of FOCS*, 2000.

[15] E. Januzaj, H.-P. Kriegel, and M. Pfeifle. Towards effective and efficient distributed clustering. In *Workshop on Clustering Large Data Sets (ICDM2003)*, 2003.

[16] E. Januzaj, H. P. Kriegel, and M. Pfeifle. Scalable density based distributed clustering. In *Proc. of EDBT*, pages 88–105, 2004.

[17] D. Kifer, S. Ben-David, and J. Gehrke. Detecting change in data streams. In *Proc. of VLDB*, 2004.

[18] H.-P. Kriegel, P. Kroger, A. Pryakhin, and M. Schubert. Effective and efficient distributed model-based clustering. In *Proc. of ICDM*, pages 258–265, 2005.

[19] J. A. Nelder and R. Mead. A simplex method for function minimiztion. *The Computer Jouranl*, 7(4):308–313, January, 1965.

[20] R. D. Nowak. Distributed em algorithms for density estimation and clustering in sensor network. In *IEEE Trans. on Signal Processing*, 2003.

[21] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *Proc. of SIGMOD*, 2003.

[22] H. Simon. Models of man. *New York*, 1957.

[23] N. Ueda, R. Nakano, Z. Ghahramani, and G. E. Hinton. Smem algorithm for mixture models. *Neural Computation*, 12(9):2109–2128, 2000.

[24] J. Yang. Dynamic clustering of evolving streams with a single pass. In *Proc. of ICDE*, 2003.