2011 12th IEEE International Conference on Mobile Data Management

Online Clustering of Distributed Streaming Data using Belief Propagation Techniques

Maria Halkidi
Dept. of Digital Systems
University of Piraeus
Email: mhalk@unipi.gr

Iordanis Koutsopoulos
Dept. of Computer & Communication Engineering, Univ. of Thessaly
and Centre for Research & Technology, Hellas
Email: jordan@uth.gr

***Abstract*—Extraction of patterns out of streaming data that are generated from geographically dispersed devices is a major challenge in data mining. The sequential, distributed fashion in which data become available to the decision maker, together with the fact that the decision maker needs to rely only on recently received data due to storage and communication constraints, render the objective of keeping track of data evolution a nontrivial one. We consider a set of distributed nodes that communicate directly with a central location. We address the problem of clustering distributed streaming data through a two-level clustering approach. We adopt belief propagation techniques to perform stream clustering at both levels. At the node level, a batch of data arrives at each time slot, and the goal is to maintain a set of salient data (local exemplars) at each time slot, which best represents the data received up to that slot. At each epoch, the local exemplars from distributed nodes are sent to the central location, which in turn performs a second-level clustering on them to derive a data synopsis global for the whole system. The local exemplars that emerge from the second level clustering procedure are fed back to the nodes with appropriately modified weights which reflect their importance in global clustering.**

As demonstrated by our experiments, the two-level belief propagation-based clustering approach together with the feedback is ideal for handling data from different nodes, as it has the same performance in terms of clustering quality with the case where the clustering is performed on the raw data sent from nodes to the central location.

I. Introduction

Mining streaming data that are generated continuously at high rates is an area of data mining with many unresolved challenges. This area has recently spurred the interest of the community [1], [2], [3], [4] due to the diverse range of real life applications where such problems arise. Sensor network data emerging from continuous measurements, online Internet traffic and web usage logs monitoring, streaming video from surveillance cameras and RFID data are some examples where information naturally becomes available to the decision maker in *sequential* and *distributed* fashion through data streams.

Clustering is a fundamental task in data mining, aiming at identifying significant underlying patterns in data [5]. Such patterns may refer to statistical distributions followed by data or simply to certain distinct clusters to which the data can be categorized. Clustering is widely recognized as means to provide a synopsis, that is a compressed version of the data volume at hand. Since the transmission of huge amount of raw streaming data is prohibitively expensive, clustering is an approach that can assist with overcoming this problem.

Distributed streaming data management possesses some distinct challenges which arise due to the *sequential, online* fashion in which data from *distributed* sources become available to the decision maker. One could argue that the decision maker could wait until all data reach him and then apply a conventional clustering or other approach to make a decision. Or, that she/he could simply wait long enough so as to have a larger portion of the data available and make a more informed decision. There are two objections to that. First, in distributed data processing, the transmission of entire streaming data to a central site is infeasible, since it will dramatically increase the network communication cost. Therefore, a local clustering can assist with providing a compact version of data generated at each of the remote sites. Second, in streaming data, it is crucial for the decision maker to be able to make a decision online, precisely at the time the last segment of data arrives. Identifying a change in patterns of data should involve a series of reaction measures that will follow decision making. For example, in video surveillance, identifying a change in the received video segments should mobilize certain mechanisms for immediate reaction. Hence it would not be meaningful for the decision maker to wait much due to the incurred delay in decision making. Moreover, at each time instant, the local sites (and decision maker) may not have access to the full amount of previously received data due to *storage constraints*. Although storage space becomes cheaper and more available as a resource, it is evident that data volume grows gigantic, and it is many more orders of magnitude higher than available storage space, especially when considered over a certain time horizon. In light of the above, the goal of the local node and the decision maker at the end of each time slot is to maintain a set of salient data items that best represent data received up to that slot. These items will be fed as input to the next slot, they will be updated with the new arriving data and will be used in subsequent decision making.

978-0-7695-4436-6/11 $26.00 © 2011 IEEE
DOI 10.1109/MDM.2011.63

Overview of our approach and Contributions. In this paper, we develop an online data clustering method suitable for distributed streaming data processing and for capturing their dynamically changing characteristics. We adopt belief propagation techniques, also known as message passing ones, which migrated from statistical physics and have found fertile ground in computer science and networking ([6], [7], [8], [9]) in addressing computationally hard combinatorial optimization problems. The key idea at the interface between computer science and statistical physics is to view the computer science problem at hand as an evolving system of entities that interact with each other in a random and competing manner. The computer science optimization problem with a given objective over a space of possible configurations can then be mapped onto a statistical physics problem instance, with a given probability distribution over the set of possible configurations of a set of interacting particles. Computing the optimal solution of the optimization problem is then equivalent to identifying the minimum energy configurations at which the probability distribution concentrates.

In our case, the problem at each time slot is to identify a set of representative data items (exemplar items) based on certain similarity metrics. The innovative feature of belief propagation is that all data items in each slot are treated as representatives with different preferences. At each time slot, simple messages are exchanged among data items and update the appropriateness of each data item to serve as representative of another, until the set of optimal representatives emerges. Belief propagation methods are suitable for online stream processing since they do not require a priori knowledge of classification patterns. We provide evidence that belief propagation techniques can efficiently materialize the key ingredient of online clustering, namely the transfer of salient information from previous slots to the next one. Specifically, representative data items from previous slots are taken into account in the next slot through a sophisticated initialization of arising preferences in the belief propagation algorithm. Our experimental results on synthetic and real streaming data sets validate the effectiveness of our method in handling dynamically evolving data streams.

Our starting point is the Affinity Propagation (AP) [9] clustering algorithm which formulates the K-centroid clustering problem and uses a message passing algorithm to solve it. We introduce an online version of the AP clustering algorithm, *StreamAP*, that makes full use of its flexibility in addressing the evolution of streaming data clusters. Since streaming data flow continuously and there are storage constraints at the decision maker node, a succinct synopsis of data from previous slots is indispensable so as to keep track of trends of previously received data. This synopsis is a set of exemplars, also known as the cluster-heads, and it provides compact information about patterns of knowledge and significant data distributions identified in past data.

Data clusters are formed at each time slot based on current data similarities and appropriate statistics of previously defined clusters. The exchanged messages among data items carry information about data similarities, the significance of a data item to serve as exemplar, and its tendencies to select other items as exemplars. A most attractive feature of the AP algorithm is that the number of clusters need not be decided a priori as in conventional clustering algorithms. Instead, clusters evolve based on data similarities and knowledge from previous slots about the suitability of a item to be an exemplar. At different time slots, new clusters may emerge, vanish or merge. Most importantly, if the number of clusters or cluster composition tends to change, our method detects it in the evolving dynamics of the data stream.

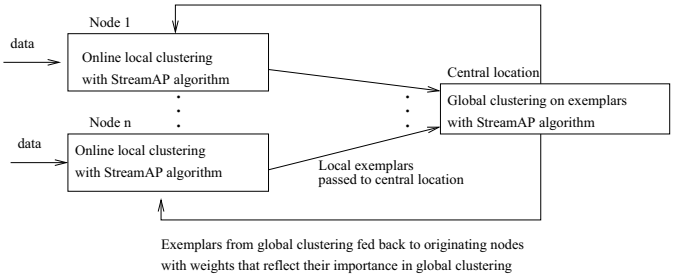


Figure 1. Architecture of the proposed clustering approach for distributed streaming data.

We consider a set of distributed nodes that communicate directly with a central location, the decision maker. The *StreamAP* algorithm is executed locally at each node, where a batch of local data arrives at each time slot. The goal of each node at each time slot is to maintain an appropriate set of salient data items (local exemplars) that best represent data received up to that slot. These are used as input at the next slot, and they are updated after consideration of the new data. At each epoch, local exemplars are sent to the central location, which in turn performs a high-level clustering to derive a global picture of the data in the system. The *StreamAP* algorithm is applied at the central location on the local exemplars. The local exemplars that emerge from the high-level clustering procedure are fed back to the nodes with appropriately modified weights which reflect their importance in global clustering. This feedback is used by local nodes in subsequent slots. Figure 1 presents the main components of the proposed clustering approach for distributed streaming data.

The main *contributions* of this work can be summarized as follows:

- We address the problem of streaming data online clustering with the framework of statistical physics belief propagation methods.
- We define a model for *building a succinct synopsis of data* processed at each time period. Our model does not make any assumptions about underlying statistical properties of data, yet it succeeds in maintaining the

most significant information at each time slot.

- We develop a method to *pass salient information to subsequent time instants* and show how it can actually guide the clustering algorithm in the next time slots. This method is the crucial step towards realizing an efficient online version of an algorithm.
- We develop a *clustering framework for distributed streaming data* that allows aggregation of clustering results from remote sites. Also it allows remote sites to receive a feedback from the central site in terms of the global data distribution and thus they can properly adjust their local clustering.
- We provide ample evidence that *our approach can identify changes in streaming data* although it is completely oblivious to underlying data distribution and the number of clusters involved. Our approach builds, merges or removes clusters on the fly as data flow into the controller.
- We *eliminate the dependence of the cluster extraction method on input information* (input parameters) that a user or decision maker needs to provide. Our algorithm exploits inherent data properties (e.g. data similarities) along the clustering process, thus removing the barriers placed by user defined parameters.

The rest of the paper is organized as follows. In section II we discuss related work. In section III we outline the basis of our approach, namely the AP algorithm. In section IV we develop the streaming and clustering models for our approach, and in section V we present the steps of our stream clustering algorithm (*APStream*). The distributed stream clustering approach is discussed in section VI. Section VII presents numerical results, notably the effectiveness of our approach to identify changes in streaming data, and section VIII concludes our study.

II. Related work

The approaches that tackle the problem of stream clustering can be categorized into: i) *one-pass methods*, that assume a unique underlying model of streaming data. These approaches cannot study the evolution of data distribution, ii) *evolving clustering methods* that take into account the behavior of data as it may evolve over time.

One-pass clustering methods use a divide-and-conquer method that achieves a constant factor approximation using a small amount of storage space [10], [11], [12]. Every time a new chunk of data arrives, a local clustering is performed to generate the cluster centers out of chunk data. Since all original data items have been processed, clustering is applied to all intermediate medians so that the k final medians are defined.

In [13], a stream clustering algorithm is proposed, which includes two clustering phases. At the online phase, the proposed approach periodically stores detailed summary statistics, while at the off-line phase, the decision maker uses these statistics to provide a description of clusters in the data stream. The main drawback of this algorithm is that the number of micro-clusters needs to be predefined. HPStream [14] incorporates a fading cluster structure and a projection-based clustering methodology to deal with the problem of high-dimensionality in data streams.

The Denstream [15] is based on DBSCAN [16], an algorithm that forms local clusters progressively by detecting and connecting dense data item neighborhoods. Cluster adjustment involves detection of data items that are neighbors to the newly inserted data, re-computation of their neighborhoods [17] and respective micro-clusters [15], and re-assignment of neighborhoods and micro-clusters to clusters if deemed necessary [17]. In Denstream, incremental computation is further enhanced through the treatment of (potential) outlier micro-clusters. The algorithm is a two-phase clustering approach. During the online phase micro-cluster are maintained while the final clusters are defined (offline) on demand by the user.

An EM-based clustering approach for distributed data streams was presented in [18]. The remote site defines their clustering model based on the idea of test-and-cluster while the coordinator combine all Gaussian models from each site directly, and the Gaussian mixture model components represent the distribution of overall distributed data streams.

Zhang *et al.* [19] presented a version of the AP algorithm that is closer to handling data streaming. According to their approach, as data flows, data items are compared one-by-one to the exemplars and they are assigned to their nearest one if a distance threshold condition is satisfied. Otherwise, data are considered outliers and are put in a reservoir. A cluster redefinition is triggered if the number of outliers exceeds a heuristic (user-defined) reservoir size or if a change in data distribution is detected. That is, the quality of clustering depends decisively on user defined thresholds. Moreover, at each time instant, there exists a number of data items in the reservoir that are similarly handled as outliers. The clustering algorithm runs anew to define a clustering of all data items arrived until the current time item only if a restart condition is satisfied.

Contrary to the above, our algorithm is capable of *handling data in batches*, and at each time period a clustering of all available data is provided. Furthermore, our approach is completely *released from user tuned parameters* that evidently confine the solution space. Our approach keeps track of data processed up to the current time slot through their cluster representatives which are provided as input to the next time slot. At each slot, all available data items are considered as potential candidate cluster-heads, and the most significant clusters are defined through a message passing procedure. Changes in clusters are identified based on inherent data properties, while the user controls only the amount of influence of historical data to current clusters. Thus, our approach efficiently tracks the evolution of data

and identifies significant changes to clusters as data flows. Moreover we introduce a distributed clustering approach that run at two levels (a node and a decision maker one).

III. Background: Affinity Propagation algorithm

Affinity propagation (AP) is a clustering method originally proposed by Frey and Dueck [9]. It belongs to the class of message passing methods and it is used on a given data set that becomes *available* in its entity to the decision maker at one time interval, so as to identify a set representative items (exemplars) out of the data set. Initially, all data items are considered equally probable to be exemplars of the data set. Real-valued messages are exchanged among data items, until a set of representative items together with the respective clusters emerges. These messages are updated at each iteration and they reflect the current affinity that a data item has to select another item as an exemplar at that iteration. The goal of the AP algorithm is to identify a set of exemplars and come up with an assignment of each data item to one of the exemplars such that the sum of similarities between data items and exemplars is maximized. Also the AP imposes constraints on exemplars to be self-exemplars.

Assume that $X = \{x_1, x_2, \ldots, x_n\}$ is a set of distinct data items and let $S(x_i, x_j)$ denote the similarity between the data items x_i and x_j, with $i \neq j$. The AP algorithm searches for a mapping $\phi(\cdot)$ which assigns each data item x_i to its nearest data item, referred to as exemplar of x_i (i.e. $\phi(x_i)$). This mapping should maximize an appropriately chosen energy function defined as:

$$\mathcal{E}[\phi] = \sum_{i=1}^{n} S(x_i, \phi(x_i)) - \sum_{i=1}^{n} \chi_i[\phi] \tag{1}$$

with $\chi_i[\phi] = \infty$, if $\phi(\phi(x_i)) \neq \phi(x_i)$ and 0 otherwise. The first term in (1) denotes the overall similarity of the data items to their exemplars, while the second term expresses the constraint that if a data item is selected as an exemplar by some data items, it should be its own exemplar.

The optimization problem defined by (1) is solved using a message passing algorithm. The AP algorithm takes as input the set of pair-wise real-valued similarities, $\{S(x_i, x_j)\}$, which describe how well the j-th item is suited to serve as exemplar for the i-th item. For $i = j$, the similarity of a item i, $S(x_i, x_i)$ indicates how likely the i-th item is to be chosen as an exemplar, and it is further called *preference*. Usually, similarity is set to a negative squared error $S(x_i, x_j) = -||x_i - x_j||^2$. The number of selected exemplars (i.e. the defined clusters) is influenced by the preference values, but it also comes out of the message passing procedure. There are two kinds of exchanged messages between data items [9]:

1) A *Responsibility message*, $r(i, k)$ that a data item i sends to candidate exemplar k. This message shows the accumulated evidence for how well item k serves as exemplar for item i. Its value is iteratively adjusted as follows:

$$r(i,k) = S(x_i, x_k) - \max_{k', k' \neq k} \{\alpha(i, k') + S(x_i, x_{k'})\} \tag{2}$$

where $\alpha(i, k')$ is the availability message presented below. This message captures evidence as to how appropriate it would be for item i to select item k' as its exemplar.
If $k = i$, the responsibility message is adjusted as:

$$r(k,k) = S(x_k, x_k) - \max_{k', k' \neq k} \{S(x_k, x_{k'})\} \tag{3}$$

2) An *Availability message*, $\alpha(i, k)$, that a candidate exemplar k sends to data item i. The value of $\alpha(i, k)$ is updated as:

$$\alpha(i,k) = \min \left\{ 0, r(k,k) + \sum_{i', i' \notin i,k} \max\{0, r(i', k)\} \right\} \tag{4}$$

For $k = i$, the *availability message* reflects accumulated evidence that item k is an exemplar and is adjusted by:

$$\alpha(k,k) = \sum_{i', i' \neq k} \max\{0, r(i', k)\}. \tag{5}$$

At each iteration, the assignments of items to exemplars (clusters) are defined as, $\phi(x_i) = \arg\max_k \{r(i,k) + \alpha(i,k)\}$, where $\phi(x_i)$ denotes the exemplar for data item x_i. The message passing procedure stops after a specific number of iterations, or after cluster structure does not significantly change for a given number of iterations.

IV. Streaming Data and Clustering Models

Streaming data arrive in batches in successive time intervals of duration T time units each, henceforth referred to as *epochs*. At each epoch i, $i = 1, 2 \ldots$, a new batch of data items $b_i = \{d_1^i, \ldots, d_M^i\}$ arrives and is processed, where M is the number of data items per batch. This is assumed to be fixed, and it is usually dictated by storage space or memory constraints at the decision maker node. Hence, the batch size is of the same order as the storage or memory space. The batch of data that arrived at epoch i obtains a time stamp T_i. If data flow is periodic with period T, the time stamp is $T_i = iT$, otherwise it depends on the batch arrival pattern. In this work, we do not make any special assumptions about the batch arrival pattern. Data are buffered as they arrive and the clustering process is applied at the end of each epoch. Once a batch of data is processed, it is discarded from the memory so that new data are collected. In stream clustering, data evolve and thus new clusters may appear, clusters may merge or vanish. The goal is to identify clusters of data and study the evolution of clusters across epochs.

We adopt a damped window model for data, according to which the weight of data items decreases with time according to an attenuation function $f(t) = 2^{-\lambda t}$, where $\lambda > 0$ is a decay factor. In other words, the weight at epoch k of a data item x_i with time stamp $T_i < T_k$ (i.e. an item that arrived at epoch i) will be given by:

$$w(x_i, T_k) = 2^{-\lambda(T_k - T_i)}. \tag{6}$$

The impact of historical data on the clustering process can then be controlled through decay factor λ. For increasing values of λ, we place less emphasis on historical data compared to most recent ones.

Owing to storage of memory constraints, data items that have arrived at previous epochs cannot be assumed to be available at future epochs. We would therefore like to maintain a succinct synopsis of data generated during previous epochs, taking also into account the weight of these historical data. That is, at each epoch i, we would like to come up with a representation for all data that arrived at epochs $i' < i$ so that, together with the new arriving data at epoch i we are able to track the stream process, detect changes into data distributions and study data evolution.

Recall that $b_i = \{d_1^i, \ldots, d_m^i\}$ is the batch of m data items that arrived during epoch i. Denote by E_{i-1} the ensemble synoptical representation of data that arrived up to epoch $i-1$. At each epoch i we would like to come up with an update rule of the form: $E_i = f(b_i, E_{i-1})$, where $f(\cdot)$ is an appropriately defined mapping operator on current batch b_i and the previous data representation E_{i-1}.

In this work, we advocate that E_{i-1} must consist of appropriately weighted exemplars of clusters defined in previous epochs. The exemplars can be considered as representatives of the data in their clusters, and thus we use them to provide a synopsis of underlying data. The weight of an exemplar e_i is the sum of the weights of the data that it represents. In the sequel, we provide some definitions that expose our methodology.

DEFINITION 1- *Weight of exemplar (or cluster)*. Suppose that data items $\{x_j\}_{j=1}^n$ with time stamps $\{T_j\}_{j=1}^n$ have selected e_i as their exemplar. The weight of e_i at epoch i is given by:

$$w(e_i, T_i) = \sum_{j=1}^{n} w(x_j, T_i - T_j). \tag{7}$$

At each epoch i, consider batch b_i and the data set E_{i-1} that contains (weighted) exemplars coming from the previous epoch, $i-1$. Our clustering algorithm operates on set $b_i \cup E_{i-1}$ in order to define new clusters. We can easily prove that the weight of an exemplar can be calculated incrementally at each epoch. Specifically, assume that $w(e_k, T_{i-1})$ is the weight of exemplar e_k at epoch $i-1$ and $\{d_1^i, \ldots, d_m^i\}$ is the subset of data items that arrived during the next epoch, i (with time stamp T_i) and have also selected e_k as their exemplar. Based on (6) the weight of data items arrived at the current epoch i is: $w(d_j^i, T_i) = 1$, $j = 1, \ldots, m$. If $T_i - T_{i-1} = 1$, the weight of e_k defined in (7) will be updated as follows:

$$w(e_k, T_i) = 2^{-\lambda} \cdot w(e_k, T_{i-1}) + m \tag{8}$$

where m is the number of data items that arrived during epoch i and are assigned to the cluster of e_k (which is denoted by $C(e_k)$).

For each exemplar e_k, the following vector is forwarded to the next epoch, $i+1$: $[\, e_k,\, w(e_k, T_i),\, \bar{D}(C(e_k), T_i)\,]$, where $\bar{D}(C(e_k), T_i)$ is the sum of weighted dissimilarities between data items in $C(e_k)$ and their exemplar e_k, that is:

$$\bar{D}(C(e_k), T_i) = \sum_{x \in C(e_k)} w(x, T_i) \cdot ||x - e_k||^2 \tag{9}$$

It can be proved that $\bar{D}(C(e_k), T_i)$ can be maintained incrementally, i.e.

$$\bar{D}(C(e_k), T_i) = 2^{-\lambda} \cdot \bar{D}(C(e_k), T_{i-1}) + \sum_{d_j^i \in C(e_k)} ||d_j^i - e_k||^2 \tag{10}$$

At each epoch i, pairwise similarities $S(x_i, x_j)$ of data items $x_i, x_j \in (b_i \cup E_{i-1})$ are properly adjusted, so that patterns of knowledge extracted from previously arrived data are also exploited in data clustering. At each epoch, our algorithm takes as input the updated similarities and defines clusters based on a message passing procedure. In the sequel, we provide a method for updating similarities between new arrived data at each epoch and exemplars from the previous epoch. This turns out to be the key for transferring salient data features from one epoch to the next one.

DEFINITION 2.1 - *Preference of exemplars from a previous epoch*. The preference of an exemplar e_k at epoch $i+1$ will be defined based on its preference in the previous epoch i and the statistics of its cluster in i. Specifically, the preference of e_k are updated as follows:

$$S_{i+1}(e_k, e_k) = S_i(e_k, e_k) \cdot \left(1 - \frac{w(e_k, T_i)}{\sum_j w(e_j, T_i)}\right) - var(C(e_k), T_i) \tag{11}$$

where $var(C(e_k), T_i) = \frac{\bar{D}(C(e_k), T_i)}{w(e_k, T_i)}$ is the weighted variance of cluster $C(e_k)$ which captures dissimilarities between the exemplar and the other data items in its cluster. At each epoch, the variance of a cluster changes as its structure changes, i.e. as data items can be added or forgotten. Thus, each of the previously defined exemplars gains an advantage over the other data items that is analogous to the weight of the data items that have selected it, while it undergoes a penalty related to the variance of its cluster.

DEFINITION 2.2 - *Preference of current data items*. The preferences of the data items generated at the current epoch $i+1$ are defined as follows:
$\forall\, d_j^{i+1} \in b_{i+1},\ j = 1, \ldots, M,$

$$S_{i+1}(d_j^{i+1}, d_j^{i+1}) = \frac{\sum_{x \in (b_{i+1} \cup E_i), x \neq d_j^{i+1}} S_{i+1}(x, d_j^{i+1})}{|b_{i+1} \cup E_i|} \tag{12}$$

where $|b_{i+1} \cup E_i|$ is the number of data under analysis (i.e. exemplars from the previous epoch and the current set of arrived data items) at epoch $i+1$.

DEFINITION 2.3 - *Similarities between an exemplar from a previous epoch and a current data item*. At each epoch $i+1$, the similarities between an exemplar e_k coming from the previous epoch and a current data item are defined as follows:

$$S_{i+1}(d_j^{i+1}, e_k) = -||e_k - d_j^{i+1}||^2 \tag{13}$$

$$S_{i+1}(e_k, d_j^{i+1}) = -w(e_k, T_i) \cdot ||e_k - d_j^{i+1}||^2 \tag{14}$$

Equation (13) indicates that the suitability of an old defined exemplar e_k to be selected as an exemplar of a current item depends on their distance. On the other hand, an old exemplar will evaluate the suitability of a currently arrived data item to be its exemplar based on their similarity and the significance of its cluster (see (14)). This is due to the fact that an exemplar is not actually a single item but it carries a synopsis of the data in its cluster. Then the assignment of an exemplar, e_k, to a new exemplar (cluster) results in the redefinition of exemplar for all data items in $C(e_k)$ and the cost of this reassignment is related to the weight of its cluster data (weight of the e_k's cluster).

DEFINITION 2.4 - *Similarity between exemplars coming from previous epoch.* If e_k, e_m are two exemplars coming from epoch i their similarity at epoch $i+1$ will be defined as:

$$S_{i+1}(e_k, e_m) = -w(e_k, T_i) \cdot ||e_k - e_m||^2 \tag{15}$$

Since an exemplar e_k represents all data items in its cluster, we assume that it approximately corresponds to a set of n_k identical copies of data items (where n_k denotes the number of items that have selected e_k as their exemplar). Then, the similarity of e_k to an exemplar e_m should reflect the similarity of the respective data cluster (i.e. n_k copies) to e_m. Considering that each data item in $C(e_k)$ influence the similarity analogously to its significance (weight), the sum of similarities of n_k copies to e_m is modeled as $w(e_k, T_i) \cdot ||e_k - e_m||^2$.

DEFINITION 2.5. The *similarity between a pair of data items arrived at the current epoch*, T_{i+1}, will be given by

$$S_{i+1}(d_j^{i+1}, d_k^{i+1}) = -||d_j^{i+1} - d_k^{i+1}||^2 \tag{16}$$

V. CLUSTERING STREAMING DATA WITH BELIEF PROPAGATION TECHNIQUES

We now develop *StreamAP* a variation of the initially proposed AP clustering algorithm, that is capable of handling sequences of data in an online fashion under the limited memory constraints imposed by streaming applications. Data arrive and are processed in batches. At each epoch, a synopsis of data is defined through their cluster representatives (exemplars) and statistics (weight of cluster and variance). Both the current batch of data and the representatives of previously processed data items are used in the clustering procedure. Once a batch of data arrives, the similarity matrix of data is updated based on the proposed stream clustering model discussed in section IV. The clusters of data are defined using the AP algorithm and the results are forwarded to the next epoch. Specifically, the proposed algorithm, *StreamAP* involves the following steps:

STEP1: The first batch of data arrives at epoch 0 and the AP algorithm (presented in Section III) is used to initialize the clustering process.

While the stream flows,

STEP2: For each exemplar, e_k of the i-th epoch,

- Vector $[\, e_k,\, w(e_k, T_i),\, \bar{D}(C(e_k), T_i)\,]$ (defined in section IV) is forwarded to the next epoch.
- The preferences of defined exemplars are updated based on (11).

STEP3: At the next epoch $i+1$, a new batch of data, b_{i+1} is received. The currently received batch of data and the set of weighted exemplars E_i received from previous epoch define the new set of data considered for clustering, referred to as $b_{i+1} \cup E_i$.

The similarity matrix of data items in $b_{i+1} \cup E_i$ is updated. Specifically:

- $\forall\ d_j \in b_{i+1}$, preferences $S_{i+1}(d_j, d_j)$ are updated as in (12).
- $\forall,\ d_i, d_j \in (b_{i+1} \cup E_i)$, similarities $S_{i+1}(d_i, d_j)$ are adjusted according to (13)-(15), as described in section IV.

STEP4: The extended version of AP clustering algorithm is applied to $b_{i+1} \cup E_i$.

Then the values of messages exchanged between data items in $b_{i+1} \cup E_i$ are adjusted as in equations (2) and (4).

Based on the method outlined above, our approach provides a mechanism that efficiently summarizes the most significant information from previously processed data and transfers this information to the next epoch of the clustering process. Thus, it enables online adaptive (re)definition of clusters and monitoring of their evolution while new batches of data arrive. The clustering results only depend on intrinsic properties of data such as data similarities, while the decision maker may control the impact of historical data to the current data clustering though decay factor λ.

VI. Distributed Data Stream Clustering Approach

We consider a distributed computing environment with n remote sites (nodes), $\{s_i\}_{i=1}^n$, and a central site (CS) of a decision maker. At each remote site, streaming data arrive continuously. At each epoch t, the *StreamAP* algorithm (discussed in section V) runs at each node, s_i, to define a local clustering model, C_i^t. The local clustering of s_i is described by its exemplars and their respective weights, that is $C_i^t = \{(le_1^i, w(e_1, T_i)) \ldots, (le_k^i, w(e_k, T_i))\}$. The local clustering models are sent to the central site (CS) in the form of stream. After receiving the clustering models of all remote sites at each epoch, the CS properly aggregates them so as to define an updated global clustering. Specifically, at epoch t, the exemplars selected in remote sites arrive to CS and the *StreamAP* algorithm runs on them to properly update the previously defined global clustering, i.e. $\hat{C}^t = f(\hat{C}^{t-1}, C_1^t, \ldots, C_n^t)$, where $f(\cdot)$ is the defined clustering operator on new arrived exemplars, $\{C_1^t, \ldots, C_n^t\}$ and the previous selected ones, $\hat{C}^{t-1}$. The updated global clustering $\hat{C}^t$ is transmitted back to the remote sites so that they adjust their local clusters (exemplars). Hence the remote sites obtain a holistic view of the data generated in the entire network though the global clustering that is disseminated to them.

The main steps of the proposed distributed stream clustering approach can be summarized as follows:

While the stream flows,

1) *StreamAP* runs at remote sites, $\{s_i\}_{i=1}^n$. Let C_i^t be the clustering model of site s_i at epoch t.
2) The remote sites transmit their clusterings $\{C_i^t\}_{i=1}^n$ to CS.
3) The CS receives the stream of local clusterings (exemplars) and runs *StreamAP* to update the global clustering defined in epoch $t-1$ with the new arrived local clusters. Let $\hat{C}^t$ be the global clustering at epoch t and $\{ge_1^t, \ldots, ge_m^t\}$ be the respective set of selected exemplars. We note that the global exemplars could be exemplars defined in previous epochs, or one of the exemplars that arrived from the remote sites at the current epoch.
4) The weight factor of global exemplars at CS is defined as $fw(ge_i^t) = \frac{w(ge_i^t, t)}{\sum_k w(ge_i^t, t)}$. It indicates the global significance of each exemplar ge_i^t with respect to the weight of points that have selected it as their exemplar.
5) CS provides feedback to remote sites, forwarding them the global clustering and respective weight factors, that is, it sends the vector $[(ge_1^t, fw(ge_1^t)), \ldots, (ge_m^t, fw(ge_m^t))]$.
6) Once a remote site s_i receives the feedback from CS, it updates the weights of its exemplars that belong to the intersection of the set denoting its local clustering C_i^t and the set denoting the global clustering. That is,

$\forall\, le_j^i \in C_i^t$, if $le_j^i \in \hat{C}^t \cap C_i^t$ then
$w(le_j^i, t) = w(le_j^i, t) \cdot (1 + fw(ge_k^t))$, where $ge_k^t \equiv le_j^i$.

VII. Experimental Study

In this section we evaluate the performance of our approach in terms of clustering accuracy using synthetic and real data sets. Also we show the effectiveness of our approach to adapt to the evolution of data and identify the changes of the data stream distribution at different time periods. Our approach is implemented in MATLAB.

A. Evaluation criterion

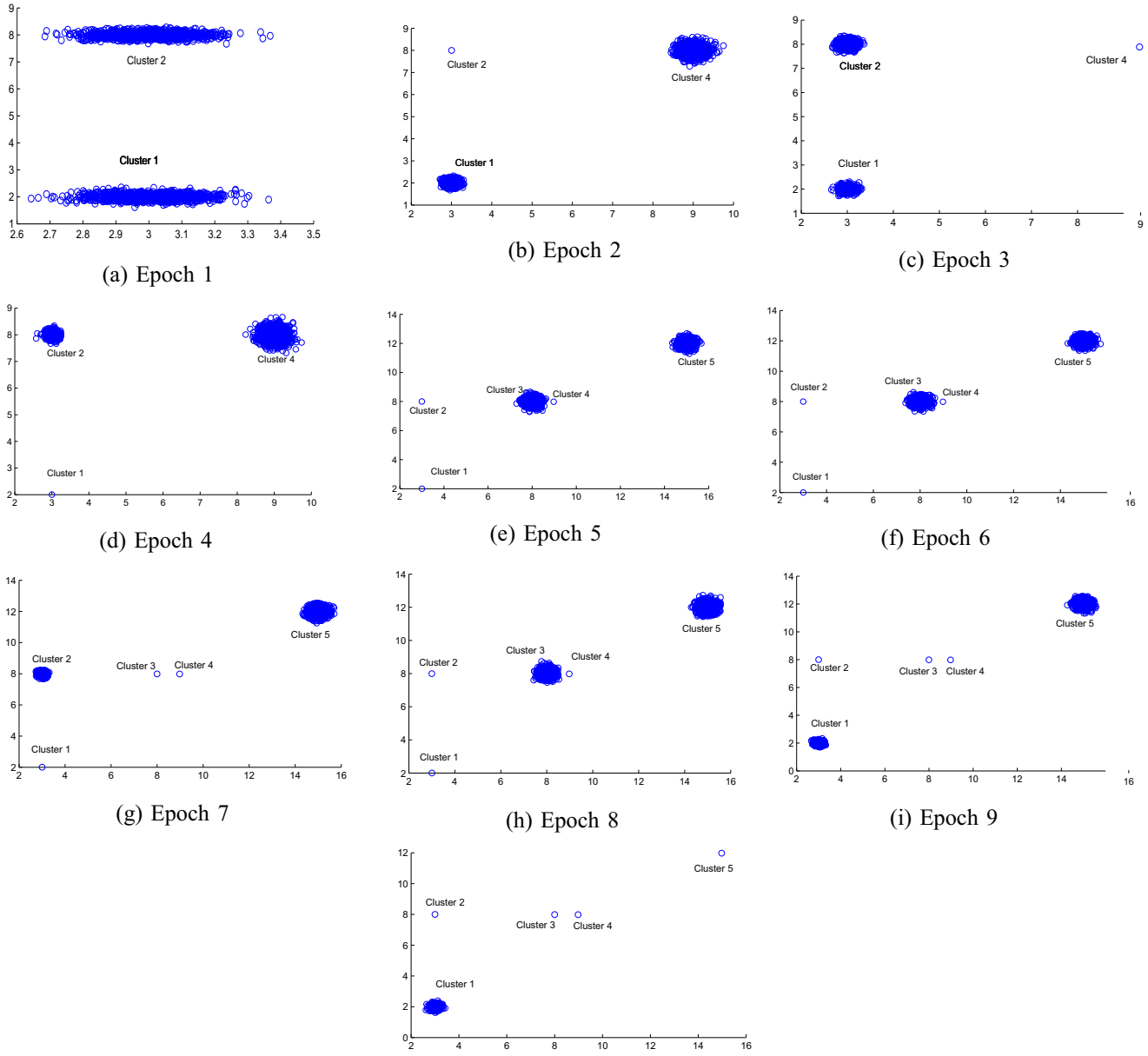
We evaluate clustering quality based on the purity of clusters. Evaluation of quality of clustering amounts to measuring the purity of defined clusters with respect to the true class labels. For experimental purposes, we assume that we know the label of data items. Then, we run our approach on the set of unlabeled data and we assess the purity of the defined clusters. An item assignment to an exemplar is correct if its class is the same as its exemplar class. Then we can define the purity measure as follows: $Purity = \frac{1}{k} \cdot \sum_{i=1}^{k} \frac{|C_i^d|}{|C_i|}$, where $|C_i^d|$ denotes the number of majority class items in cluster i, $|C_i|$ is the size of cluster i and k is the number of clusters.

B. Clustering quality evaluation

1) Capturing data evolution in clustering process: The synthetic collection of streaming data was created with different numbers of clusters and dimensions. We assume that the synthetic data follow a set of Gaussian distributions, while the mean and variance of data distribution vary during data generation.

Here, we chose to present a two-dimensional data set for the sake of visualizion of clustering results. Streaming data are generated so as to follow five Gaussian distributions. Figure 2 depicts the evolution of two-dimensional streaming data as they arrive in batches of $2,000$ items. We assume that at every epoch, a new batch of data arrives. Data distribution changes as new batches of data arrive and thus new clusters may appear or existing clusters may disappear. Our approach manages to identify the new emerging clusters at different epochs, while it also updates with new data items the previously defined clusters. We evaluate the purity of clusters at different time periods and our algorithm achieves to assign all data items to the proper clusters (i.e. $purity = 1$).

Table I and Table II show the weights of clusters identified at each epoch (when a new batch of data arrived) when λ is set to 0.25 and 5, respectively. In general terms, the weight of clusters reflects the number and the age of data items that they contain. We expect that some of the exemplars may be forgotten if they have not been selected for a long time period. As we have already noted in section IV, the time period that an exemplar could be considered as active



(a) Epoch 1 (b) Epoch 2 (c) Epoch 3

(d) Epoch 4 (e) Epoch 5 (f) Epoch 6

(g) Epoch 7 (h) Epoch 8 (i) Epoch 9

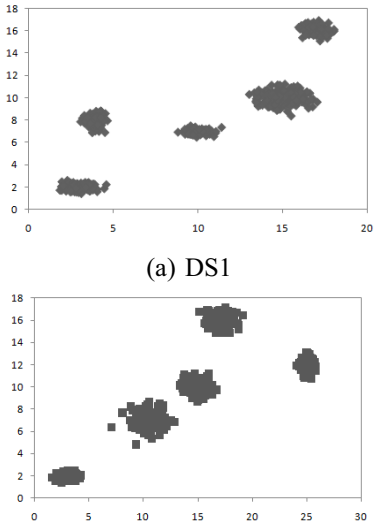
(j) Epoch 10

Figure 2. Visualization of data under process at different time slots and the defined clusters.

is user-defined and depends on the definition of factor λ. By comparing the clustering results in Table I and Table II, we can observe that by increasing λ, we eliminate the influence of historical data and thus only the most recently updated clusters are kept. For instance, Figure 2 manifests that no new data assigned into to cluster 4 during the last three epochs. Besides Tables I and II show that the weight of cluster 4 decreases as time passes (for both values of λ value). Specifically, when $\lambda = 5$, the importance of previously arrived data is significantly eliminated and thus cluster 4 is totally forgotten during the last three epochs.

The experimental study above shows the effectiveness of our approach to capture data evolution and identify the emerging clusters as data flows.

2) Purity of clustering results: We assess the clustering quality of APStream using synthetic and real data. We consider the synthetic data sets, DS1, DS2 depicted in Figures 3(a) and (b), respectively. Each dataset contains 1,200 points. We generate an evolving streaming data set (further referred to as DS) by randomly choosing one of



(a) DS1

(b) DS2

Figure 3. Visualization of Synthetic data sets (DS1 and DS2).

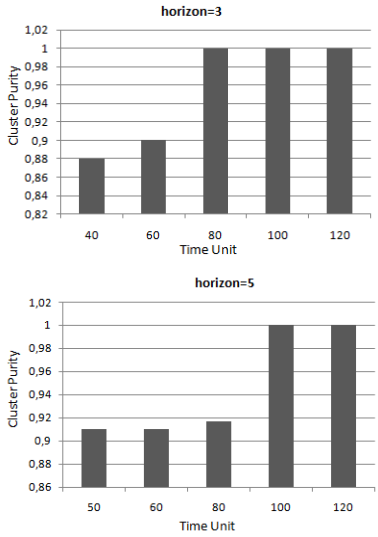


Figure 4. Clustering quality for the DS data stream (stream speed=2,000): (a) horizon=3, (b) horizon=5.

the data sets DS1 and DS2 20 times. Figure 4 depicts the purity of clustering results for DS. Since the weight of data fades out gradually, we compute the purity considering only the points arriving in a pre-defined horizon (h) from current time. We set the stream arrival rate at 2,000 points per epoch and test the clustering purity for h=3 and h=5. We observe that the clustering purity is always higher than 88%. APStream can result in higher clustering quality in a relatively large horizon.

The real data set is the KDD CUP'99 Network Intrusion Detection data set [20]. For our experiments we use a sample

	Cluster weights				
Epochs	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
1	1000	1000	-	-	-
2	1841.89	841.89	-	1000	-
3	2049.84	2208.95	-	841.89	-
4	1724.71	2358.49	-	2208.95	-
5	1451.29	1984.25	993	1865.49	1000
6	1221.39	1669.55	1823.01	1582.69	1841.89
7	1028.065	1904.92	1533.96	1331.877	3049.84
8	865.495	1602.838	1786.903	1124.971	4065.60
9	1728.792	1348.82	1503.60	946.98	4419.75
10	2454.735	1135.219	1265.373	797.315	3717.553

Table I
EVOLUTION AND WEIGHTS OF CLUSTERS EXTRACTED FROM SYNTHETIC STREAMING DATA AT DIFFERENT EPOCHS WHEN $\lambda = 0.25$.

	Cluster weights				
Epoch	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
1	1000	1000	-	-	-
2	1841.89	32.25	-	1000	-
3	533.26	1502.01	-	32.25	-
4	17.667	547.94	-	1502.008	-
5	1.55	18.12	993	54.94	1000
6	1.049	1.56	1019.03	15.716805	1032.25
7	1.033	501.05	32.84	1.491150	1533.26
8	1.032	16.66	503.026	-	1548.914
9	1001.032	1.52	16.72	-	1049.4036
10	1032.28	1.047	1.52	-	33.79

Table II
EVOLUTION AND WEIGHTS OF CLUSTERS EXTRACTED FROM SYNTHETIC STREAMING DATA AT DIFFERENT EPOCHS WHEN $\lambda = 5$.

of the KDD CUP'99 data set (as in [19], [15]) that contains 10% of its data ($494,021$ items) and all continuous attributes out of the total 42 available. The data set is converted into data stream by taking the data input order as order of streaming. The decay factor λ is set to 0.25 and the stream rate is set at $1,000$ data items per time slot. Assessing the purity of clusters at the end of stream flow, our algorithm achieves to build a clustering model with purity 99%. Moreover, the quality of clustering is comparable to that in the results of static clustering, i.e. when we run the AP algorithm on the whole data set. Thus our approach seems to efficiently identify the data distribution as data flows, even though only a portion of the data becomes available at each time epoch. By providing a synopsis of previously arrived data through their exemplars and statistics, our approach succeeds in keeping track of evolving data and in forwarding the appropriate synoptical information to the next iteration of clustering procedure. We also evaluate the performance of our algorithm in comparison with the STRAP algorithm [19] on the Intrusion Detection data set. We compute the purity of clustering results at selected time items similar to those considered in [19]. Figure 5 shows that the results of our approach is comparable to those found in [19] and in all four cases it achieves to define clusters with high purity.

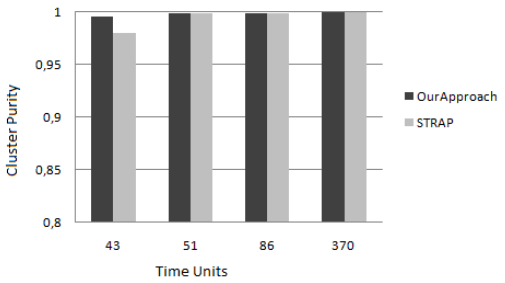


Figure 5. Clustering Purity - Comparative study on Intrusion detection dataset

3) Evaluating the Distributed Stream Clustering approach: We assume a network of 20 sensors and a base station which collects and processes sensor data and aims at identifying clusters of data generated in the network. Each sensor generates streaming data corresponding to one of the data sets DS1, DS2. We evaluate two clustering scenarios. The first one assumes that sensors forward the stream of their measurements directly to the base station, which then runs the APStream algorithm to define the global clustering. The second scenario refers to our clustering approach for distributed streaming data. Each sensor runs APStream on its local streaming data and forwards the defined clusters to the base station. Then, APStream is applied to the stream of local exemplars as described in Section VI so that the global clustering of sensor data is defined. The base station sends feedback to the sensors so that they properly adjust their clusters at the next epoch. Our experimental study shows that the results of the two clustering scenarios are similar in terms of cluster purity. Our clustering approach achieves to define clusters with purity 1. Moreover, it is clear that the communication cost in the case of the proposed clustering approach is significantly lower than that in the first scenario since each sensor needs to send only a synopsis of its data to the base station.

VIII. Conclusion

We develop a novel approach for online distributed clustering of streaming data using belief propagation techniques. The key ingredient of our streaming clustering approach, APStream, is the method by which a compressed, synoptical version of data is passed on to the next epoch and aids in re-initialization of the clustering process. The adopted message passing approach runs at each epoch, it successfully identifies clusters, and it maintains an accurate representation of data, although they become gradually available to the decision maker. To deal with the streaming data generated in distributed setting, we proposed a two-level approach. We provide ample evidence that our approach can successfully identify changes in streaming data although it is completely oblivious to underlying data distribution and the number of clusters involved. Moreover the proposed two-level belief propagation-based clustering approach is proved to have the same performance in terms of clustering quality with the case where the clustering is performed on the raw streaming data that are sent from nodes to the central location.

Acknowledgment. The authors acknowledge the support of the European Commission through the FET STREP project STAMINA (FP7-265496) and the STREP project CON4COORD (FP7-223844).

References

[1] W. Fan, H. Wang, and P. Yu, "Active mining of data stream," in *SDM*, 2004.
[2] S. Papadimitriou, A. Brockwell, and C. Faloutsos, "Adaptive hands-off stream mining," in *VLDB*, 2003.
[3] B. Babcock and C. Olston, "Distributed top-k monitoring," in *SIGMOD*, 2003.
[4] P. Kranen, I. Assenty, C. Baldauf, and S. T., "Self-adaptive anytime stream clustering," in *ICDM*, 2009.
[5] J. Han and M. Kamber, *Data Mining:Concepts and Techniques*. Morgan Kaufmann, 2006.
[6] D. Shah, "Network scheduling and message-passing," in *Book chapter in Performance Modeling and Engineering*, a collection of tutorials from ACM Sigmetrics/Performance, 2008.
[7] M. Mezard, "Passing messages between disciplines," *Science*, vol. 301, 2003.
[8] S. Sanghavi, D. Shah, and A. S. Willsky, "Message passing for maximum weight independent set," *IEEE Transactions on Information Theory*, vol. 55, no. 11, 2009.
[9] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, 2007.
[10] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: Theory and practice," *IEEE Trans. of Knowlende and Data Eng.*, vol. 15, no. 3, pp. 515–528, 2003.
[11] M. Charikar, O. L, and R. Panigraphy, "Better streaming algorithms for clustering problems," in *STOC*, 2003, pp. 10–17.
[12] S. Guha, N. Mishra, R. Motwani, and O'Callaghhan, "Clustering data streams," in *Proc. of FOCS*, 2000.
[13] C. Aggarwal, J. Han, J. Wang, and P. Yu, "A framework for clustering evolving data streams," in *VLDB*, 2003.
[14] C. Aggarwal, J. Han, J. Wang, and P. Yu, "A framework for projected clustering of high dimensional data streams," in *VLDB*, 2004.
[15] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *SDM*, 2006.
[16] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Database with Noise," in *KDD*, 1996.
[17] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu, "Incremental Clustering for Mining in a Data Warehousing Environment," in *VLDB*, 1998.
[18] Z. Aoying, C. Feng, Y. Ying, S. Chaofeng, and X. He, "Distributed data stream clustering: A fast em-based approach," in *ICDE*, 2007.
[19] X. Zhang, C. Furtlehner, and M. Sebag, "Data streaming with affinity propagation," in *ECML/PKDD*, 2008.
[20] KDDCup, "Computer network intrusion detection," in *http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html*, 1999.