Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/090458

Affinity Propagation-based Clustering For Data Streams

*Walid Atwa and Kan Li**

School of Computer Science and Technology, Beijing Institute of Technology, China

Received: 27 Nov. 2014, Revised: 27 Feb. 2015, Accepted: 1 Mar. 2015
Published online: 1 Jul. 2015

Abstract: Clustering data stream is an active research area that has recently emerged to discover knowledge from large amounts of continuously generated data. Several clustering algorithms have been proposed for static data. Nevertheless, data stream clustering imposes several challenges to be addressed, such as dealing with dynamic data that arrive in an online fashion, capable of performing fast and incremental processing of data objects, suitably addressing time and memory limitations, and how to handle the evolving patterns that are important characteristics of streaming data with dynamic distributions. In this paper, we propose an algorithm that extends Affinity Propagation (AP) to handle evolving data steam with dynamic distribution. Affinity Propagation was proposed as a clustering algorithm extracted a set of exemplars that best represent the dataset using a message passing method. We present a semi-supervised clustering technique (SSAP) that incorporates labeled exemplars into the AP algorithm to deal with changes in the data distribution, which requires the stream model to be updated as soon as possible. Experimental results with state-of-the-art data stream clustering methods demonstrate the effectiveness and efficiency of the proposed method.

Keywords: Affinity propagation, clustering data streams, exemplars.

1 Introduction

Mining streaming data that are generated continuously at high rates is an area of data mining with many unresolved challenges. Streaming data is the discipline specifically concerned with handling large-scale datasets in an online fashion [1,2]. Applications of streaming data include mining data generated by sensor networks, meteorological analysis, stock market analysis, and computer network traffic monitoring. These applications involve datasets that are far too large to fit in main memory and are typically stored in a secondary storage device. From this standpoint, mining the patterns in streaming data imposes a great challenge for cluster analysis.

Clustering is one of the most important unsupervised learning methods which partition a given set of objects into subsets called clusters, such that objects in the same cluster are similar and objects in different clusters are dissimilar. Clustering data stream continuously produces and maintains the clustering structure from the data stream in which the data items continuously arrive in the ordered sequence. Algorithms for clustering data streams should ideally fulfill the following requirements [3,4]: (1) provide timely results by performing fast and incremental processing of data objects; (2) rapidly adapt to change dynamics of the data, which means algorithms should detect when new clusters may appear, or others disappear; (3) scale to the number of objects that are continuously arriving; (4) provide a model representation that is not only compact, but that also does not grow with the number of objects processed; and (5) rapidly detect the presence of outliers and act accordingly. There are many data streaming algorithms have been adapted from clustering algorithms, e.g., the partitioning method k-means [5,6,7,8], the density based method DBSCAN [9,10], and grid based methods [11,12].

Semi-supervised clustering algorithms are concerned with finding good partitions of data in the presence of side information. Two popular forms of side information are partial labels and instance-level constraints. In this paper, we propose an algorithm that extends Affinity Propagation (AP) method [13] for streaming data (called SSAPStream). We incorporate the labeled exemplars into the AP algorithm to handle the evolving data stream with dynamic distribution. The proposed algorithm SSAPStream involves three main steps described as follow:

1. Using the AP method on the first bunch of data arrives at time t_0 to identify the exemplars and initialize the stream model.

* Corresponding author e-mail: likan@bit.edu.cn

2. As the stream flows in, each point p is compared to the exemplars; if too far from the nearest exemplar, p is put in the buffer, otherwise the stream model is updated.
3. If the buffer is full or a change in the data stream is detected, the stream model is rebuilt based on the current model and buffer using the set of labeled exemplars.

The experimental results on synthetic and real data sets validate the effectiveness of our method in handling dynamically evolving data streams. Also, we study the execution time and memory usage of SSAPStream, which are important efficiency factors for streaming algorithms.

The rest of the paper is organized as follows. Section 2 briefly reviews related work. Section 3 presents Affinity Propagation and describes the proposed Semi-Supervised Affinity Propagation (SSAP). Section 4 describes the proposed algorithm (SSAPStream) for streaming data. Section 5 shows the experimental results. Section 6 presents the conclusion and future work.

2 Related Work

Data streaming is one of the major data mining tasks, faces additional challenges compared to traditional data. Recently, the clustering of data streams has been attracting a lot of research attention. The methods that discuss the problem of data stream clustering can be categorized into: (1) *one-pass methods* that assume a unique underlying model of streaming data and cannot study the evolution of data distribution, (2) *evolving clustering methods* that take into account the behavior of data as it may evolve over time.

The first well-known algorithm performing clustering over entire data streams is the STREAM algorithm proposed by Guha et al. [5,14]. It is a Divide-and-Conquer algorithm that builds clusters incrementally and hierarchically using bi-criterion approximation algorithms. The stream is processed in a batch mode where a set of data points that fit in the main memory are clustered into a smaller set of intermediate medians, each weighted by the number of data points assigned to it. The process is repeated for subsequent batches of data points until the set of intermediate medians fit the memory space; they are then clustered and summarized into a higher level of intermediate medians. Such methods simply view data stream clustering as a variant of one-pass clustering.

The CluStream framework proposed in [6] is effective in handling evolving data streams. It divides the clustering process into online and offline components. The online component uses microclusters to continuously captures synopsis information from the data stream while the offline component uses this information and other user inputs to yield on-demand clustering results. The main drawback of this algorithm is that the number of micro-clusters needs to be predefined. For high-dimensional data stream clustering, Aggarwal et al. [15] proposed HPStream, which reduces the dimensionality of the data stream via data projection before clustering.

Based on the online maintenance and offline clustering strategy, Cao et al. [9] proposed a DenStream algorithm, which extends DBSCAN [16] by introducing microclusters to the density-based connectivity search. It is an algorithm that forms local clusters progressively by detecting and connecting dense data item neighborhoods. During the online phase microclusters are maintained while the final clusters are defined (offline) on demand by the user. Independently, Chen and Tu [11,12] also proposed a density-based method termed D-Stream. Rather than using microclusters, D-Stream partitions the data space into grids and maps new data points into the corresponding grid to store density information, which are further clustered based on the density. One common limitation of the above algorithms is that they are computationally more expensive, requiring more memory to store nonlinear cluster structures and more time to update them.

Zhang et al. [17] presented a version of the AP algorithm called StrAP that is closer to handling data streaming. The StrAP algorithm proceeds by incrementally updating the current model if the current data item fits the model. Otherwise, detecting the data item as an outlier.

Recently, Ackermann et al. proposed StreamKM++, a two-step algorithm that is merge-and reduce [7]. The reduce step is performed by the coreset tree, considering that it reduces $2m$ objects to m objects. The merge step is performed by another data structure, namely the bucket set, which is a set of L buckets (also named buffers), where L is an input parameter. Each bucket can store m objects. The StreamKM++ algorithm can obtain better clustering results but takes more computing time, as its running time dependent on the dimensionality of the data and thus, unsuitable for clustering high dimensional data. Moreover, produces clusters of the large data set when the algorithm terminates, instead of forming clusters at any time step when data flow in.

It must be emphasized that both Divide-and-Conquer and two-level schemes keep their computational load within reasonable limits as they only build the data model upon the users explicit request. In the rest of time, they only maintain a summary of the data, which makes them ill-suited to applications such as system monitoring where the data model has to be continuously available at all times.

3 Semi-Supervised Affinity Propagation

In this section, we first describe the Affinity Propagation (AP) algorithm. An extension, Semi-Supervised AP, is then presented to find good partitions of data in the presence of side information (i.e. labeled data).

3.1 Affinity Propagation

Affinity Propagation (AP) [13] is a clustering method that takes as input similarities between data items. It aims to identify exemplars among data items and forms clusters around these exemplars. Each exemplar is characterized by a data item representative of the sample itself and some other similar items. The objective of AP is to maximize the sum of similarities between the data items and their 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 $i \neq j$. AP algorithm searches for a mapping $c(\cdot)$, which assigns each data item x_i to its nearest data item, referred to as exemplar of x_i (i.e. $c(x_i)$). This mapping should maximize the following objective function:

$$S(c) = \sum_{i=1}^{N} s(x_i, c(x_i)) + \sum_{k=1}^{N} \delta_k(c) \tag{1}$$

where $\delta_k(c)$ is an exemplar consistency constraint such that if data item x_i has selected x_k as its exemplar, i.e., $c(x_i) = x_k$, then data item x_k must select itself as an exemplar, i.e., $c(x_k) = x_k$.

$$\delta_k(c) = \begin{cases} -\infty, & \text{if } c(x_k) \neq x_k \text{ but } \exists x_i : c(x_i) = x_k \ ; \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The objective function defined by (1) is solved using a message passing algorithm. AP algorithm takes as input the set of similarities, $\{S(x_i, x_j)\}$, which describe how well the x_j item is suited to serve as exemplar for the x_i item. Usually, similarity is set to a negative squared error $S(x_i, x_j) = -\|x_i - x_j\|^2$. AP begins by simultaneously considering all data items as potential exemplars, and iteratively exchanges messages between data items until a good set of exemplars and clusters emerges. The messages can be combined at any stage to decide which data items are exemplars and, for every other item, which exemplar it belongs to. There are two kinds of messages exchanged between data items [13]:

1. The responsibility message $r(i,k)$, sent from data item x_i to candidate exemplar item x_k, reflects the accumulated evidence for how well-suited item x_k is to serve as the exemplar for item x_i.

$$r(i,k) = S(x_i, x_k) - \max_{j \neq k} \{S(x_i, x_j) + a(i,j)\} \tag{3}$$

If $k = i$, the responsibility message is adjusted as:

$$r(k,k) = S(x_k, x_k) - \max_{j \neq k} \{S(x_i, x_j)\} \tag{4}$$

2. The availability message $a(i,k)$, sent from candidate exemplar item x_k to item x_i, reflects the accumulated evidence for how appropriate it would be for item x_i to choose item x_k as its exemplar, taking into account the support from other items that item x_k should be an exemplar. It is initialized as zero and updated as follows:

$$a(i,k) = \begin{cases} \sum_{i' \neq k} \max[0, r(i',k)], & i = k; \\ \min[0, r(k,k) + \sum_{i' \notin \{i,k\}} \max[0, r(i',k)]], & i \neq k. \end{cases} \tag{5}$$

After convergence, the exemplars are obtained by calculating the set of positive $a(i,i) + r(i,i)$ messages for each x_i, and the items are assigned to their respective exemplars (clusters) according to the following rule, $c(x_i) = \arg\max_k (a(i,k) + r(i,k))$. 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.

3.2 Semi-Supervised AP (SSAP)

The original affinity propagation is an unsupervised clustering method. To utilize the partially labeled data, semi-supervised clustering aims to improve the clustering performance by learning from a combination of both labeled samples and unlabeled data.

Assume we have L labeled data set $\{(x_1, y_1), (x_2, y_2), \ldots, (x_l, y_l)\}$, where y_l is the cluster label of data item x_l, and U unlabeled data items $x_{l+1}, x_{l+2}, \cdots, x_{l+u}$. Let C be the set of exemplars in the data set. For a certain labeled sample x_i $(1 \leq i \leq l)$ and unlabeled data item x_j $(l+1 \leq j \leq u)$ we can have two possible situations where the labeled sample may be associated with the unlabeled data item after a run of the AP algorithm.

- The unlabeled data item x_j takes the labeled sample x_i as the cluster exemplar. The message $a(x_i, x_i) + r(x_i, x_i)$ is positive, i.e., $x_i \in C$, and if $x_i = \arg\max\{a(x_j, x_k) + r(x_j, x_k)\}$ for each $k = \{1, 2, \ldots, N\}$.
- The labeled sample x_i takes the unlabeled data item x_j as the cluster exemplar. The message $a(x_i, x_i) + r(x_i, x_i)$ is negative, i.e., $x_i \notin C$, and if $x_j = \arg\max\{a(x_i, x_k) + r(x_i, x_k)\}$ for each $k = \{1, 2, \ldots, N\}$.

If one of the two conditions is satisfied, the unlabeled data item x_j is the most similar to the labeled sample x_i. Then, the unlabeled data item x_j is selected and set to the label of x_i. Therefore, we can select the most similar unlabeled data item x_j as follows:

$$V = \begin{cases} x_j & if x_i = \arg\max_{1 \leq k \leq N} \{a(x_j, x_k) + r(x_j, x_k)\} and x_i \in C \\ & if x_j = \arg\max_{1 \leq k \leq N} \{a(x_i, x_k) + r(x_i, x_k)\} and x_i \notin C \end{cases}$$

where V is the new labeled sample set picked from U. Selecting the unlabeled data items is defined according to the operational mechanism of the AP algorithm. This process repeats until no unlabeled data items left in U. At each iteration, the selection process takes advantage of the updated results of the AP algorithm.

4 Clustering Streaming Data

4.1 Streaming Data Model

In clustering data stream, data evolve over time and thus new clusters may appear, clusters may merge or deleted. The goal is to identify clusters of data and study the evolution of clusters over time. We consider the problem of clustering a data stream in the damped window model, in which the weight of data items decreases with time t according to the decay function $f(t) = 2^{-\lambda \cdot t}$, where, $\lambda > 0$ is a decay factor. In other words, the weight at time t_k of a data item x_i with time stamp t_i (i.e. an item that arrived at time t_i) will be given by:

$$w(x_i, t_k) = 2^{-\lambda (t_k - t_i)} \tag{6}$$

Data items that have arrived at previous time cannot be assumed to be available at future time. Therefore, we would like to maintain a succinct synopsis of data generated during previous time, taking also into account the weight of these historical data. The exemplars can be considered as representatives of the data in their clusters, and thus we define an exemplar vector to provide a synopsis of underlying data. The exemplar vector is consists of a set of 4-tuple (e_i, n_i, w_i, t_i), where e_i ranges over the exemplars, n_i is the number of items associated to exemplar e_i, w_i is the weight of e_i (see Definition 1), and t_i is the last time stamp when an item was associated to e_i.

Definition 1: Weight of exemplar. For an exemplar e, at a given time t, let n be the set of data items that are associated to e at or before time t. The weight of e is defined as the sum of the weights of all data items that is associated to e. Namely, the weight of e at t is:

$$w(e, t) = \sum_{j=1}^{n} w(x_j, t) \tag{7}$$

The weight of any exemplar is constantly changing. However, we have found that it is unnecessary to update the weight values of all data items and exemplars at every time step. Instead, it is possible to update the weight of an exemplar only when a new data item is associated to that exemplar. For each exemplar, the time when it receives the last data item should be recorded so that the weight of the exemplar can be updated according to the following result when a new data item arrives at the exemplar.

Proposition 1. Suppose an exemplar e receives a new data item x_n at time t_n, and suppose the time when e receives the last data record is $t_l (t_n > t_l)$, then the weight of e can be updated as follows:

$$w(e, t_n) = 2^{-\lambda (t_n - t_l)} w(e, t_l) + 1 \tag{8}$$

Proof. Let the data items $\{x_i\}_{i=1}^{m}$ with time stamps $\{t_i\}_{i=1}^{m}$ be associated to the exemplar e at time t_l, we have:

$$w(e, t_l) = \sum_{i=1}^{m} w(x_i, t_l) \tag{9}$$

According to (6), we have that:

$$\begin{aligned} w(x_i, t_n) &= 2^{-\lambda (t_n - t_i)} = 2^{-\lambda (t_n - t_l)} 2^{-\lambda (t_l - t_i)} \\ &= 2^{-\lambda (t_n - t_l)} w(x_i, t_l), \quad for \quad i = 1, \cdots, m. \end{aligned} \tag{10}$$

Therefore, we have:

$$\begin{aligned} w(e, t_n) &= \textstyle\sum_{i=1}^{m} w(x_i, t_n) + w(x_n, t_n) \\ &= \textstyle\sum_{i=1}^{m} w(x_i, t_n) + 2^{-\lambda (t_n - t_n)} \\ &= \textstyle\sum_{i=1}^{m} w(x_i, t_n) + 1 = \sum_{i=1}^{m} 2^{-\lambda (t_n - t_l)} w(x_i, t_l) + 1 \\ &= 2^{-\lambda (t_n - t_l)} \textstyle\sum_{i=1}^{m} w(x_i, t_l) + 1 \\ &= 2^{-\lambda (t_n - t_l)} w(e, t_l) + 1 \end{aligned}$$

From Proposition 1, we save huge amount of computation time. To update all exemplars at each time step requires O(N) computing time for weight update at each time step. In contrast, using Proposition 1 allows us to update only one exemplar, leading to O(1) running time.

4.2 SSAPStream Algorithm

We develop SSAPStream a variation of the initially AP clustering algorithm, that is capable of handling sequences of data in an online fashion under the limited memory constraints imposed by streaming applications. The detailed procedure of SSAPStream is described in Algorithm 1.

Firstly, we apply AP clustering algorithm on the first bunch of data items that arrived at time t_0 to identify the exemplars and initialize the stream model (step 2). For each new data item p, we try to merge p into its nearest exemplar e_i in the current model (w.r.t. distance d). If $d(p, e_i)$ is less than some threshold ε (heuristically, set to the average distance between data items and exemplars in the initial model), p is affected to the i-th cluster and the model is updated accordingly; otherwise, p is put in the buffer (steps 4-10).

In order to prevent the number of exemplars from growing beyond control, one must be able to forget the exemplars that have not been visited for a long time. In other words, for each existing exemplar e_i, if no new data item is merged into it, the weight of e_i will decay gradually and it should be deleted and its memory space released for new exemplar. Accordingly, we compare the weight of each exemplar with its lower limit of weight (denoted as η). If the weight of an exemplar is below its

lower limit of weight, we can safely delete it (steps 12-14). The lower limit of weight is defined as:

$$\eta(t_c,t_i)=\frac{1-2^{-\lambda(t_c-t_i+1)}}{1-2^{-\lambda}} \tag{11}$$

where t_c is the current time and t_i is the last update time of the exemplar e_i. This function $\eta(t_c,t_i)$ is an increasing function for fixed t_i value. Thus, the longer an exemplar exists, the larger its weight is expected to be.

A key difficulty in streaming data is to detect a change in the generative process underlying the data stream, referred to as *concept drift*, which requires the stream model to be updated as soon as possible. The clustering results are said to be changed according to the following two criteria. The first criterion is based on the number of data items in the buffer; when it exceeds the buffer size, the stream model should be updated. The second criterion is based on the variation of clusters distribution, when the ratio of data items in the clusters is changed dramatically, e.g., the cluster that contains half of the data items in the last clustering result has suddenly disappeared in the current clustering result.

In order to detect the changes in streaming data, we adopt a double-threshold method. One threshold θ named *exemplar variation threshold* is utilized to determine that the variation of the ratio of data items associated to an exemplar is big enough. The exemplar that exceeds the exemplar variation threshold is seen as a different exemplar (steps 16-17). And then, the number of different exemplars is counted, and the ratio of different exemplars is compared with the other threshold φ named *exemplar difference threshold*. If the ratio of different exemplars is larger than the exemplar difference threshold, a large number of exemplars are varied in the ratio of data items, and thus the stream model should be updated (steps 20-21).

The variation of the ratio of data items associated to the exemplar e_i between the last clustering result and the current temporal clustering result is calculated and compared by a zero-one function $f(e_i^{t_i},e_i^{t_c})$, where t_i is the last update time and t_c is the current time of exemplar e_i.

$$f(e_i^{t_i},e_i^{t_c})=\left\{\begin{array}{ll}1, & \text{if}\left|\frac{n_i^{t_i}}{\Sigma_{x=1}^{C}n_x^{t_i}}-\frac{n_i^{t_c}}{\Sigma_{x=1}^{C}n_x^{t_i}}\right|>\theta\\ 0, & \text{otherwise}\end{array}\right\} \tag{12}$$

where C is the number of exemplars and n_i is the number of data items associated to exemplar e_i.

Finally, the number of different exemplars is summed, and the ratio of different exemplars is compared with φ. Thus, the stream model should be updated according to the following:

$$UpdateModel=\left\{\begin{array}{ll}Yes, & \text{if the buffer is full}\\ Yes, & \text{if } \Sigma_{i=1}^{C}\frac{f(e_i^{t_i},e_i^{t_c})}{C}>\varphi\ )\\ No, & \text{otherwise}\end{array}\right\} \tag{13}$$

Algorithm 1. SSAPStream Algorithm

1. Initialize Buffer = {}
2. Apply AP method on the first set of data arrived at time t_0
3. **While** data stream is active **do**
4. Read data item x_t;
5. Compute the nearest exemplar e_i to x_t;
6. **If** $d(x_t,e_i)<\varepsilon$ **then**
7. Update the stream model ;
8. **Else**
9. Insert x_t into the Buffer ;
10. **End If**
11. $numdiffexemplars=0$;
12. **For** each exemplar e_i in C **do**
13. **If** w_i (weight of exemplar e_i) $<\eta(t_c,t_i)$
14. Delete e_i ;
15. **Else**
16. **IF** $\left|\frac{n_i^{t_i}}{\Sigma_{x=1}^{C}n_x^{t_i}}-\frac{n_i^{t_c}}{\Sigma_{x=1}^{C}n_x^{t_i}}\right|>\theta$ **then**
17. $Numdiffexemplars{+}{+}$;
18. **End If**
19. **End For**
20. **If** Buffer is full or $\frac{Numdiffexemplars}{C}>\varphi$ **then**
21. Rebuild the stream model using SSAP ;
22. Buffer = {} ;
23. **End If**
24. **End While**

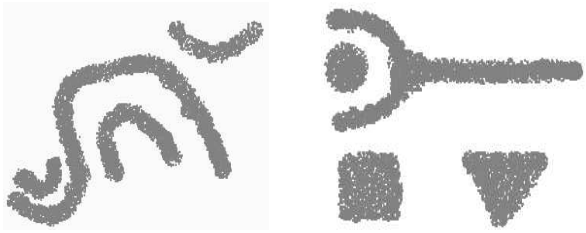
A new model is rebuilt by launching SSAP on the dataset and data items in the buffer. As mentioned previously, we have two possible situations where the labeled samples represented in the exemplars can be associated with the unlabeled data items in the buffer. The cost of selecting current exemplar e_i to be the exemplar of data item x_j is ordinary similarity $-d(x_j,e_i)^2$, while the cost of selecting x_j to be the exemplar of e_i is increased by a factor of n_i (i.e. n_i is the number of data items associated to exemplar e_i). Therefore, the current exemplar e_i will have more chance to be an exemplar again. SSAP accordingly selects the set of data items from buffer that are most similar to the exemplar e_i and thus, merges them to the exemplar e_i. This process repeats until no data items left in the buffer.

5 Experiments

In this section, we evaluate the effectiveness and efficiency of SSAPStream and compare it with several well-known data stream clustering algorithms, including CluStream [6] , DenStream [9], StrAP [17] and StreamKM++ [7]. All the experiments are conducted on a 2.4 GHz Intel Core 2 PC running Windows XP with 2 GB main memory.

5.1 Data Sets and Evaluation

To evaluate the effectiveness and efficiency of SSAPStream algorithm, both synthetic and real data sets



(a) (b)

Fig. 1: Synthetic data sets.

are used. Two synthetic data sets, DS1 and DS2, are generated as shown in Figures 1(a) and (b), respectively. Each of them contains 10,000 points. We generate an evolving data stream (denoted as EDS) by randomly choosing one of the data sets (DS1 and DS2) 10 times, for each time the chosen data set forms a 10,000 points segment of the data stream, and the total length of the evolving data stream is 100,000. Because we know the true cluster labels of each point in the data sets DS1 and DS2, we can get the true cluster label of each point in EDS.

Two real data sets, KDDCUP99 [19] and URLs [20], are employed for the evaluation. KDDCUP99 is a data set that evolves significantly over time and has been widely used to evaluate data stream clustering algorithms [6,7,9, 17]. It consists of a series of TCP connection records of LAN network traffic managed by MIT Lincoln Labs. The complete data set contains approximately 4.9 million records, and as in the previous work [6,9,17], a sub-sampled subset of length 494020 is used. Each connection is classified into either a normal connection or specific kinds of attack (such as *buffer overflow*, *ftp write*, *guess passwd*, and *neptune*). Each connection record in this data set contains 42 attributes, and as in [6,7,9,17], all 34 continuous attributes are used for clustering.

The streaming URLs are studied to predict malicious URLs from benign URLs [20], and thus protect users from accessing on rogue Web sites. The 120-day streaming data consists of more than 2 million URLs, each of which is described by lexical features, e.g., the unusual appearance of .com in the middle of a URL, and host-based features, e.g., the ownership and IP prefix of a web site host. A URL is labeled as benign or malicious.

For the performance measure we used the Rand index [18], which measures how accurately a cluster can classify data items by comparing cluster labels with the underlying class labels.

$$RI = \frac{a+b}{\binom{n}{2}}$$

where a is the number of pairs of data items having the same cluster label and the same class label, and b is the number of pairs of items having different cluster labels and different class labels. The value of Rand index lies between 0 and 1. A higher Rand index indicates better clustering results [18].

5.2 Parameter Setting

The initialization of the SSAPStream algorithm considers the first 1000 points of DS1 and DS2 and the first 1000 connections of the KDDCUP99 data set which totally includes 494,021 network connection records, and the first 1000 URLs of the 2,396,130 URLs data set. The distance threshold ε heuristically, set to the average distance between data items and exemplars in the initial model, and the maximum number of data items stored in the buffer is 100 items. The parameter λ is the decay factor that controls the importance of historical data to current clusters, we set λ = 0.98.

The parameters for detecting evolving data distribution, i.e., exemplar variation threshold θ and exemplar difference threshold φ, may change for different application data. In our experiments, the exemplar variation threshold θ is set to 0.2, and, the exemplar difference threshold φ is set to 0.4. Several observations provide the clues to set these parameters:

– Detecting small changes requires a low threshold value.
– Detecting dramatic changes requires a high threshold value.
– Detecting changes in unstable data sets requires a high threshold value.
– Detecting changes in stable data sets requires a low threshold value.

The four compared algorithms and their parameter settings are summarized as follow:

– CluStream [6] separates the clustering process into an online microclustering and an offline microclustering. As in [6], the parameters are set as follows: the number of microclusters $q = 10 \times k$, $InitNumber = 1000$, the maximum boundary factor $t = 2$, the user-defined threshold $\sigma = 512$, the pyramidal time frame parameters $\alpha = 2$ and $l = 10$, and the time horizon $h = 100$.
– DenStream [9] divides the clustering process into an online part and an offline part. As in [9], the parameters are set as follows: $InitNumber = 1000$, the fading factor $\lambda = 0.25$, the maximum radius threshold $\varepsilon = 16$, the weighting threshold $\mu = 10$, and the outlier threshold $\beta = 0.2$.
– StrAP [17] splits the data set into some subsets, followed by performing AP on each subset to generate exemplars with each being weighted by the number of assigned data points. According to [17], the parameters are set as follows: $InitNumber = 1000$, the maximum number of outliers stored in reservoir $M = 100$, the time length $\Delta = 50$ and the PH test threshold $\lambda = 0.01$.

– StreamKM ++ [7] is an algorithm which processes data streams in chunks of M data items. It summarizes each chunk by clustering its data into k centers. The chunk size M is set to 500.

5.3 Experimental Results

At first, we test the clustering quality of SSAPStream algorithm and compare with state-of-the-art data stream clustering methods. Figure 2 shows the average Rand indices by the five algorithms. In general, SSAPStream obtains the highest Rand indices on the three testing data sets. On the two synthetic data streams, SSAPStream has significantly outperformed the clustering methods such as CluStream, StrAP and StreamKM++. For instance, on DS1 data set, SSAPStream has obtained the average Rand index as high as 0.95, which is 0.12 higher than the second winner DenStream and is 0.22 higher than StrAP algorithm. Similarly, on DS2 data set, SSAPStream has obtained the average Rand index of 0.97, which is 0.09 higher than the second winner DenStream and is 0.18 higher than StrAP.

On the KDDCUP99 real data streams, SSAPStream also outperforms the other four data stream clustering methods. SSAPStream has obtained the average Rand index of 0.77, which is slightly better than the second winner StrAP by 0.09 and makes a significant improvement compared with the three clustering methods (i.e., CluStream, DenStream and StreamKM++).

The efficiency of algorithms is measured by the execution time. The execution time includes the time used to update the cluster structure and the time used to realize cluster labeling, over the whole data stream. For instance, In CluStream and DenStream, the execution time includes two parts, the time used to assign the data item to the nearest microcluster (online component) and the time used to generate the final clusters (offline component).

We test and compare the execution time consumed by the five algorithms on the KDDCUP99 data set. First, the algorithms are tested on the KDDCUP99 data with different sizes as shown in Figure 3. From the figure, SSAPStream algorithm has a significant advantage over its competitors in terms of the execution time, i.e., StrAP, and StreamKM++. It can also be seen that SSAPStream has better scalability since its clustering time grows slower with an increasing data size.

Next, the algorithms are tested on the KDDCUP99 data with different dimensionality. We set the size of data set as 100K and vary the dimensionality in the range of 2 to 40. We list the time costs under different dimensionality by the five algorithms as shown in Figure 4. SSAPStream is faster than other algorithms and scales better with an increasing dimensionality. For instance, when the dimensionality is increased from 2 to 40, the time of SSAPStream only increases by 20 seconds which is better than the second faster DenStream that increases by 50 seconds.

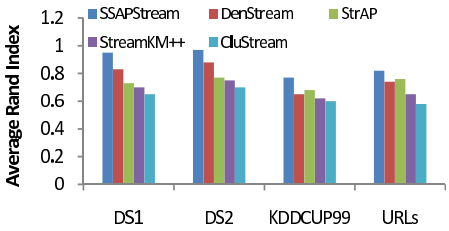


Fig. 2: Clustering quality comparison.

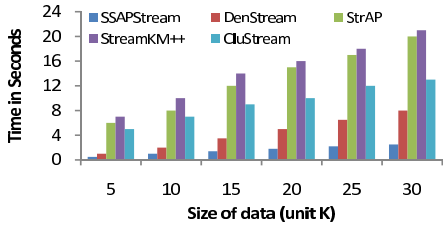


Fig. 3: Execution time with different sizes of data sets.

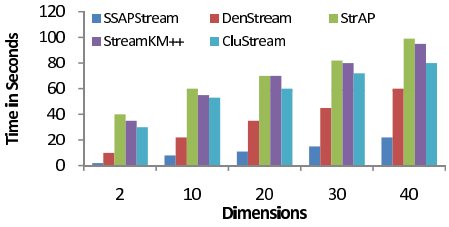


Fig. 4: Execution time with different dimensions.

One common feature for the algorithms applied to data stream is their limited upper bounds for the memory usage. Since the memory usage may fluctuate in the progress of data streams. The memory usage is directly measured by the peak memory usage of each algorithm during the stream clustering process. We used both synthetic data sets and the KDDCUP99 data set to evaluate the memory usage of SSAPStream.

Figure 5 shows the proposed SSAPStream algorithm has a significant advantage over its competitors in memory usage. For instance, on the two synthetic data streams, SSAPStream requires, respectively, 56.5 and 57.8 KB memory, which are only about 22 percent of the memory used by the second winner DenStream algorithm. On KDDCUP99, SSAPStream consumes only 18 percent of the memory used by DenStream and no more than 8 percent of the memory used by StreamKM++.

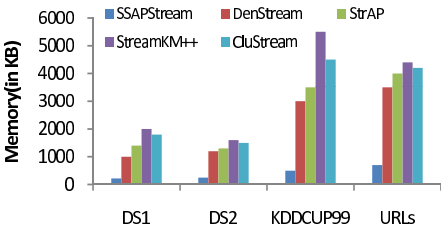
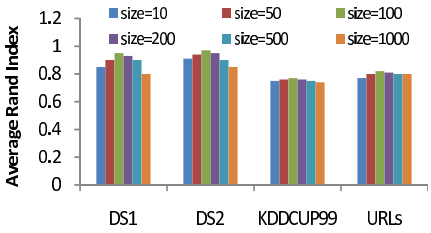
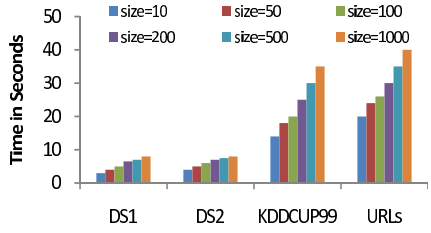


Fig. 5: Memory usage comparison.



(a)



(b)

Fig. 6: Average rand index and execution time with different buffer sizes.

5.4 Sensitivity Analysis

In this section, we report and compare the clustering results by SSAPStream on the four testing data sets when using different buffer sizes and decay factor parameter λ.

We first analyze the performance in terms of average Rand index and execution time when using different buffer size on the four testing data streams. Figure 6 reports the results. From figure 6(a), it is clear that on all testing streams, the average Rand index remains stable when different buffer sizes are used, with the highest average Rand index achieved in buffer size = 100 in most cases. The execution time however strongly depends on the buffer size as shown in figure 6(b). Based on the above analysis, considering both the effectiveness and efficiency, a rational choice for size equal 100, which is used as the default parameter in this paper.

An important parameter of SSAPStream is the decay factor λ that controls the importance of historical data to current clusters. In our pervious experiments, we set it to 0.98, which is a moderate setting. We test the clustering quality on the KDDCUP99 data set by varying λ from 0.1 to 4 as shown in Figure 7. By setting λ to the values

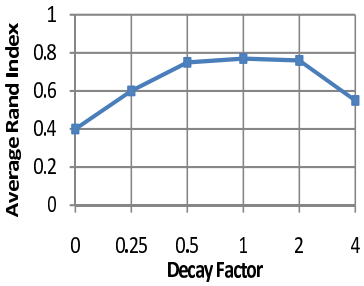


Fig. 7: Clustering quality vs. λ.

ranging between 0.1 and 0.5, SSAPStream generates poor clustering results, achieving the minimum average Rand index. Also, when λ set to high values, it achieves the minimum average Rand index. And it achieves the highest average Rand index when λ is ranges from 0.5 to 2.

6 Conclusion and Future Work

In this paper, we propose SSAPStream, a semi-supervised clustering algorithm that extends Affinity Propagation for clustering evolving data streams. The algorithm uses the exemplars to summarize information of the historical data items and thus, it has limited memory consumption. Our goal is to make full use of both the existing labeled exemplars and the unlabeled data items to improve the clustering performance in streaming data. Experimental results on KDD99 data demonstrate the effectiveness and efficiency of the proposed algorithm. This work opens several perspectives for further research. A main limitation of AP is that it cannot model the category consisting of multiple subclasses since it represents each cluster by a single exemplar, such as scene analysis and character recognition. To remedy this deficiency, we extend the single-exemplar model to a multi-exemplar model.

Acknowledgement

The Research was supported in part by The National Basic Research Program of China (973 Program) (No.2013CB329605), Natural Science Foundation of China(No.60903071), Specialized Research Fund for the Doctoral Program of Higher Education of China, and Training Program of the Major Project of BIT.

References

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

[2] G. Cormode, S. Muthukrishnan, and W. Zhuang, Conquering the divide: Continuous clustering of distributed data streams, Proceedings of the International Conference on Data Engineering (ICDE), 1036-1045, (2007).
[3] A. S. Jonathan, R. F. Elaine, C. B. Rodrigo, R. H. Eduardo, C. P. Andr, and J. Gama, Data stream clustering: A survey, ACM Computing Surveys (CSUR), **46** (1), 1-31, (2013).
[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems, Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. 1-16, (2002).
[5] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. OCallaghan, Clustering data streams: Theory and practice, IEEE Transactions on Knowledge and Data Engineering (TKDE), **15**, 515-528, (2003).
[6] C.C. Aggarwal, J. Han, J. Wang, and P. S. Yu, A framework for clustering evolving data streams, Proceedings of the International Conference on Very Large Data Bases (VLDB), 81-92, (2003).
[7] M. R. Ackermann, M. Maartens, C. Raupach, K. Swierkot, C. Lammersen, and C. Sohler, StreamKM++: A clustering algorithm for data streams, Journal on Experimental Algorithmics, **17**(1), (2012).
[8] M. Shindler, A. Wong, and A. Meyerson, Fast and accurate k-means for large datasets, Advances in Neural Information Processing Systems (NIPS), 2375-2383, (2011).
[9] F. Cao, M. Ester, W. Qian, and A. Zhou, Density-based clustering over an evolving data stream with noise, SIAM Conference on Data Mining (SDM), 326-337, (2006).
[10] C. Ruiz, E. Menasalvas, and M. Spiliopoulou, C-DenStream: Using Domain Knowledge on a Data Stream, LNCS, **5808**, 287-301, (2009).
[11] Y. Chen and L. Tu, Density-based clustering for real-time stream data, Proceedings of the 13th ACM international conference on Knowledge discovery and data mining (SIGKDD), 133-142, (2007).
[12] Y. Chen and L. Tu, Stream data clustering based on grid density and attraction, ACM Transactions on Knowledge Discovery from Data (TKDD), **3**(3), 1-27, (2009).
[13] B. Frey, and D. Dueck, Clustering by passing messages between data points, Science, 972-976 (2007).
[14] S. Guha, N. Mishra, R. Motwani, and L. OCallaghan, Clustering Data Streams, Proceedings. 41st Ann. IEEE Symp. Foundations of computer Science, (2000).
[15] C.C. Aggarwal, J. Han, J. Wang, and P.S. Yu, A Framework for Projected Clustering of High Dimensional Data Streams, Proceedings. 30th Intl Conf. Very Large Data Bases (VLDB), (2004).
[16] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, Proceedings. Second Intl Conf. Knowledge Discovery and Data Mining, (1996).
[17] X. Zhang, C. Furtlehner, J. Perez, C. Germain-Renaud, and M. Sebag, Toward autonomic grids: Analyzing the job flow with affinity streaming, Proceedings of the 15th ACM international conference on Knowledge discovery and data mining (SIGKDD), (2009).
[18] W. M. Rand, Objective Criteria for the Evaluation of Clustering Methods, J. Am. Statistical Assoc., **66**(336), 846-850, (1971).
[19] KDD99, KDD Cup 1999 data (computer network intrusion detection): http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.
[20] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, Identifying suspicious URLs: An application of large-scale online learning, Proceedings of the International Conference on Machine Learning (ICML), 2009, 681-688.

Walid Atwa
received his M.Sc.degrees in the Software Engineering, Menoufia University. He is currently a Ph.D.student in Beijing Institute of Technology, China. His research interests are data mining and machine learning.

Kan Li
Professor in Beijing Institute of Technology, China. His research interests are in the area of data mining and machine learning.