A MPAA-Based Iterative Clustering Algorithm Augmented by Nearest Neighbors Search for Time-Series Data Streams

Jessica Lin[1], Michai Vlachos[1], Eamonn Keogh[1], Dimitrios Gunopulos[1], Jianwei Liu[2], Shoujian Yu[2], and Jiajin Le[2]

[1] Department of Computer Science and Engineering University of California, Riverside Riverside, CA 92521
{jessica, mvlachos, eamonn, dg}@cs.ucr.edu
[2] College of Computer Science & Technology, Donghua University
liujw@mail.dhu.edu.cn

Abstract. In streaming time series the Clustering problem is more complex, since the dynamic nature of streaming data makes previous clustering methods inappropriate. In this paper, we propose firstly a new method to evaluate Clustering in streaming time series databases. First, we introduce a novel multi-resolution PAA (MPAA) transform to achieve our iterative clustering algorithm. The method is based on the use of a multi-resolution piecewise aggregate approximation representation, which is used to extract features of time series. Then, we propose our iterative clustering approach for streaming time series. We take advantage of the multiresolution property of MPPA and equip a stopping criteria based on Hoeffding bound in order to achieve fast response time. Our streaming time-series clustering algorithm also works by leveraging off the nearest neighbors of the incoming streaming time series datasets and fulfill incremental clustering approach. The comprehensive experiments based on several publicly available real data sets shows that significant performance improvement is achieved and produce high-quality clusters in comparison to the previous methods.

1 Introduction

Numerous clustering algorithms of time series have been proposed, the majority of them work in relatively static model, while many current and emerging applications require support for on-line analysis of rapidly changing streaming time series. In this paper, we present a new approach for cluster streaming time series datasets.

Our work is motivated by the recent work by Jessica Lin and Eamonn Keogh on iterative incremental clustering of time series [1]. While we speed up clustering process by examining the time series at increasingly finer levels of approximation using multi-solution piecewise aggregate approximation (MPAA). We argue that MPAA has all the pruning power of Wavelet transform dimensionality reduction, but is also able to handle arbitrary length queries, is much faster to compute and can support a

T.B. Ho, D. Cheung, and H. Liu (Eds.): PAKDD 2005, LNAI 3518, pp. 333 – 342, 2005.

more general distance measures. Although there has been a lot of work on more flexible distance measures using Wavelet [2, 3], none of these techniques are indexable. While time series databases are often extremely large, any dimensionality reduction technique should support index method. For the task of indexing MPAA has all the advantages of Wavelet with none of the drawbacks.

Our work addresses four major challenges in applying their ideas for clustering time series in a streaming environment. Specifically, our work has fourfold main contribution:

Clustering Time Series in Streaming Environment: Streaming time-series are common in many recent applications, e.g., stock quotes, e-commerce data, system logs, network traffic management, etc [4]. Compared with traditional datasets, streaming time-series pose new challenges for query processing due to the streaming nature of data which constantly changes over time. Clustering is perhaps the most frequently used data mining algorithm. Surprisingly, clustering streaming time-series still have not explored thoroughly, to the best of our knowledge, no previous work has addressed this problem.

MPAA-based Iterative Time Series Clustering: PAA (Piecewise Aggregate Approximation) [5] transformation produces a piecewise constant approximation of the original sequence. In this paper, we introduce a novel multi-resolution PAA (MPAA) transform to achieve our iterative clustering algorithm.

Proposed stopping criteria for multi-level iterative clustering: We solve the difficult problem of deciding exactly how many levels are necessary at each node in iterative clustering algorithm by using a statistical result known as the Hoeffding bound [6].

Time Series Clustering augmented Nearest Neighbor: Our proposed inline clustering algorithm exploits characteristic of a neighborhood and significantly reduce clustering construction time and improve clustering quality.

The rest of the paper is organized as follows. In section 2, we develop enhanced iterative clustering and streaming clustering algorithm. Section 4 presents the experimental evaluation of our proposed algorithms both in offline and online form. We conclude in Section 5 with some summary remarks and future research directions.

2 Streaming Iterative Clustering Method

2.1 MPAA -Based Dimensionality Reduction

Our MPAA-based time series representation work is derived from the recent work by Eamonn Keogh [5] and Yi and Faloutsos [7] on segmenting time series representations of dimensionality reduction.

The basic idea on which their work develops is as follows. Suppose, we denote the set of time series which constitute the database as $X = \{X_1, \cdots, X_n\}$. A time series

X_i of length n is represented in N space by a vector $\overline{X}_i = \overline{x}_{i1}, \cdots, \overline{x}_{iN}$. The i[th] element of $\overline{X}_i$ is calculated by the following equation:

$$\overline{X}_i = \frac{N}{n} \sum_{j=\frac{n}{N}(i-1)+1}^{\frac{n}{N}i} x_j \tag{1}$$

Our MPPA method divides time series X_i of length n into a series of lower-dimensional signal with different resolution N. where $N \in \{1, \cdots, n\}$. Simply stated, in first level, the data is divided into N "frames", whose sizes need not be contiguous and equal. The mean value of the data falling within a frame is calculated and a vector of these values becomes the data reduced representation. Then Recursively applying the above pairwise averaging process on the lower-resolution array containing the averages, we get a multi-resolution representation of time series.

We give a simple example to illustrate the MPAA decomposition procedure in Table 1. Suppose we are given a time series containing the following eight values A= [3, 5, 2, 6, 4, 8, 7, 1] and we initiate divide it into 4 segments. The MPPA transform of A can be computed as follows. We first average the values together pairwise to get a new “lower-resolution” representation of the data with the following average values [4, 4, 6, 4]. In other words, the average of the first two values (that is, 3 and 5) is 4 and that of the next two values (that is, 6 and 4) is 5, and so on. Recursively applying the above pairwise averaging process on the lower-resolution array containing the averages, we get the following full decomposition:

Table 1. A simple example to illustrate the MPAA decomposition procedure

Resolution	MPAA Values
8	3,5,2,6,4,8,7,1
4	4,4,6,4
2	4,5
1	4.5

The MPAA approximation scheme has some desirable properties that allow incremental computation of the solution. These properties are necessary in order for the algorithm to be able to operate efficiently on large datasets and streaming environment.

2.2 Enhanced Iterative Clustering Methods

Our iterative clustering method is similar to [1]. The algorithm works by leveraging off the multiresolution property of MPPA.

Note that an open problem that arise with this sort of iterative models is the definition of a minimum number of observations, i.e., devising an objective functions that determine the quality of clustering results from the previous stages to eliminate the need to re-compute all the distances.

Due to us perform the k-Means clustering algorithm, starting at the second level and gradually progress to finer levels, in order to find the stopping resolutions as low as possible to complete a good k-means clustering, it may be sufficient to consider only a small subset of the multi-level clustering examples that pass through the level of decomposition tree. We solve the difficult problem of deciding exactly how many levels are necessary at each node by using a statistical result known as the Hoeffding bound or additive Chernoff bound [6], which have in fact be successfully used in online decision trees [8][9]. After n independent observations of a real-valued random variable r with range R, the Hoeffding bound ensures that, with confidence $1-\delta$, the true mean of r is at least $\overline{r} - \varepsilon$, where $\overline{r}$ is the observed mean of the samples and

$$\varepsilon = \sqrt{\frac{R^2 In(1/\delta)}{2n}} \tag{2}$$

This is true irrespective of the probability distribution that generated the observations.

Table 2. The enhanced iterative clustering algorithms

	Algorithm SI-kMeans
1	Decide on a value for k.
2	Perform MPAA decomposition on raw data
3	Initialize the k cluster centers (randomly, if necessary).
4	Compute the hoeffding bound(ε)
5	Run the k-Means algorithm on the level i of MPAA representation of the data
6	Use final centers from level i as initial centers for level i+1. This is achieved by projecting the k centers returned by k-Means algorithm for the 2^i space in the 2^{i+1} space.
7	Compute the distance D_{center} between initial centers of level i and initial centers for level i+1
8	Compute respectively maximum values of the sum of squared intra-cluster errors in j^{th} iterative clustering and $(j+1)^{th}$ iterative clustering, i.e. $E_{\max(i)}$ and $E_{\max(i+1)}$
9	If $E_{\max(i+1)} - E_{\max(i)} > \varepsilon$, exit.
10	If $D_{center} > \varepsilon$, goto 3.

We call the new iterative clustering algorithm supporting stopping criteria SI-kMeans, where S stands for "stopping criteria.", and I stands for "interactive." Table 2 gives a skeleton of this idea.

2.3 Proposed Streaming Clustering Algorithm

A key challenging issue with streaming time series clustering algorithm is the high rate of input sequences insertion.

To illustrate our application, consider the following issue. Most streaming time series are related to previously arrived time series or future ones, hence, this strong temporal dependency between the streaming time series should not be ignored when clustering streaming data collection. This issue can be addressed by considering the nearest neighbor. A simple distance metric between two new arriving time series and the clustering center will show how much they are related to each other. Hence, the nearest neighbor analysis allows us to automatically identify related cluster.

Below, we give a more formally definition in order to depict our Streaming Clustering algorithm.

Definition 1. Similarity measure: To measure closeness between two sequences, we use correlation between time series as a similarity measure. Supposed that time-series T_i and T_j in a sliding window which length is w is represented respectively by $\{< u_{i1}, t_{i1} >, \cdots, < u_{in}, t_{in} >\}$ and $\{< v_{j1}, t_{j1} >, \cdots, < v_{jn}, t_{jn} >\}$ The Similarity between two time series T_i and T_j is defined by

$$\text{si}\,milarity(T_i, T_j) = \frac{\sum_{k=1}^{w} u_{ik} \bullet v_{jk} - w\bar{u}\,\bar{v}}{\sqrt{\sum_{k=1}^{w} u^2_{ik} - w\bar{u}^2} \bullet \sqrt{\sum_{k=1}^{w} v^2_{ik} - w\bar{v}^2}} \tag{3}$$

Definition 2. Similar: If $\text{si}\,milarity(T_i, T_j) \geq \varsigma$, then a time series T_i is referred to as similar to a time series T_j .

Based on the definition of similar in **Definition 1**, we can define the ζ -neighborhood $N_\zeta(T_i)$ as follows:

Definition 3. ζ -neighborhood $N_\zeta(T_i)$: ζ -neighborhood for a time series T_i is defined as a set of sequences $\{X_j : \text{si}\,milarity(X_j, T_i) \geq \zeta\}$.

Our proposed clustering algorithm exploits characteristic of a neighborhood. It is based on the observation that a property of a time-series would be influenced by its neighbors. Examples of such properties are the properties of the neighbors, or the percentage of neighbors that fulfill a certain constraint. The above idea can be translated into clustering perspective as follows: a cluster label of a time-series depends on the cluster labels of its neighbors.

The intuition behind this algorithm originates from the observation that the cluster of time series sequences can often be approximately captured by performing nearest neighbor search. In what follows, our idea is explained in detail.

Initially, we assume that only time series in now window is available. Thus, we implement SI-kMeans clustering on these sequences itself and form k clusters. Adding new sequences to existing cluster structure proceeds in three phases: neighborhood search, identification of an appropriate cluster for a new sequences, and re-clustering based on local information. The proposed incremental clustering algorithm STSI-kMeans (streaming time series iterative K-means clustering algorithm) can be discribed as follow:

Step 1. Initialization. Get next new sequences $\{T_c - w+1, \cdots, T_c\}$ in now window.

Step 2. Neighborhood search. Given a new incoming sequences $\{T_c - w+1, \cdots, T_c\}$ and let C_{K_j} be the set of clusters containing any time series belonging to $N_\zeta(T_j)$, obtain $\{N_\zeta(T_c - w+1), N_\zeta(T_c - w+2), \cdots, N_\zeta(T_c)\}$ by performing a neighborhood search on $\{T_c - w+1, \cdots, T_c\}$, and find the candidate cluster C_{K_j} which can host a new sequence $T_j \in \{T_c - w+1, \cdots, T_c\}$, that mean to identify $C_{K_j} \supset N_\zeta(T_j)$.

Step 3. Identifying an appropriate cluster. Cluster If there exists a cluster C_K that can host a sequence T_j, and then add T_j to the cluster C_K. Otherwise, create a new cluster C_{new} for T_j.

To identify a cluster C_K which can absorb the new time-series T_j from the set of candidate clusters C_{K_j}, we employ a simple but effective approach, which measures the Euclidean distance between the center of candidate clusters and the new time-series T_j, the cluster which returns the minimum distance is selected as a cluster C_K which can absorb the new time-series T_j.

Step 4. Re-clustering over affected cluster. If T_j is assigned to C_K or create a new cluster C_{new} for T_j, then a merge operation needs to be triggered. This is based on a locality assumption [10]. Instead of re-clustering the whole dataset, we only need to focus on the clusters that are affected by the new time-series. That is, a new time-series is placed in the cluster, and a sequence of cluster re-structuring processes is performed only in regions that have been affected by the new time-series, i.e., clusters that contain any time-series belonging to the neighborhood of a new time-series need to be considered.

Note that based on SI-kMeans re-clustering, the number of clusters, k' value Decide by the number of affected clusters k'' by absorbing the new time-series. Where $k' = k''$.

Step 5. Repetition. Repeat Step 2-4 whenever new sequences available in the next window.

3 Experimental Evaluation

In this section, we implemented our algorithms SI-kMeans and STSI-kMeans, and conducted a series of experiments to evaluate their efficiency. We also implemented the I-kMeans algorithm, to compare against our techniques. When not explicitly mentioned, the results reported are averages over 100 tests.

3.1 Datasets

The data using in our experiment is similar to [1]. We tested on two publicly available, real datasets: JPL datasets and heterogeneous datasets [11]. The dataset cardinalities range from 1,000 to 8,000. The length of each time series has been set to 512 on one dataset, and 1024 on the other. Each time series is z-normalized to have mean value of 0 and standard deviation of 1.

3.2 Offline Clustering Comparison

To show that our SI-kMeans approach is superior to the I-kMeans algorithm for clustering time series in offline form, in the first set of experiments, we performed a series of experiments on publicly available real datasets. After each execution, we compute the error and the execution time on the clustering results.

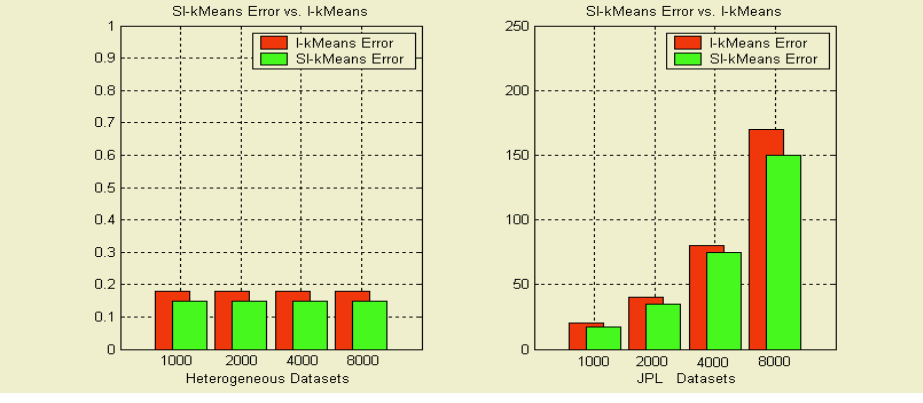

Fig. 1. Comparison of the clustering approximation error between SI-kMeans and I-kMeans. (a) Error of SI-kMeans algorithm on the Heterogeneous dataset, presented as fraction of the error from the I-kMeans algorithm. (b) Objective functions of SI-kMeans algorithm on the JPL dataset, presented as fraction of error from the I-kMeans algorithm

Figure 1 illustrates the results of clustering approximation error. As it can be seen, our algorithm achieves better clustering accuracy.

Figure 2 shows Speedup of SI-kMeans against I-kMeans. the SI-kMeans algorithm finds the best result in relatively early stage and does not need to run through all levels.

3.3 Online Clustering Comparison

In the next set of experiments, we compare the inline performance of STSI-kMeans to I-kMeans, which is essentially a comparison between an online and the corresponding offline algorithm. Since original I-kMeans algorithm is not suitable for online clustering streaming time series, we revise it and adapt it to online clustering.

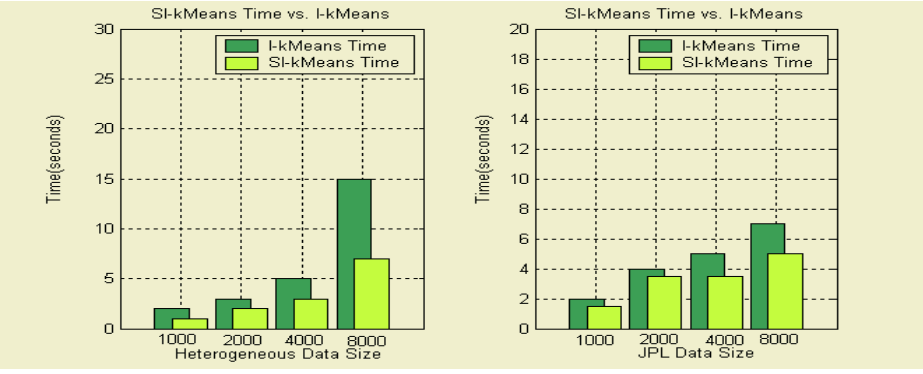

Fig. 2. Speedup of SI-kMeans against I-kMeans. (a) SI-kMeans vs. I-kMeans algorithms in terms of clustering error and running time for in the Heterogeneous dataset. (b) SI-kMeans vs. I-kMeans algorithms in terms of objective function and running time for JPL dataset

We quantify firstly the differences in the performance of the two algorithms. We report the cumulative relative error over count-based or sequence-based windows, which measure the relative increase in the cumulative error when using STSI-kMeans and I-kMeans.

$$CRE = \left| \frac{\sum_{j=1}^{q} Error_{\text{STSI-kMeans}}(w_j) - \sum_{j=1}^{q} Error_{\text{I-kMeans}}(w_j)}{\sum_{j=1}^{q} Error_{\text{I-kMeans}}(w_j)} \right| \times 100 \quad (4)$$

Where, q is the number of elapsed windows. In Figure 3, we depict CRE as a function of q and k. In the experiment of Figure 5, the length of streaming time series 1000,2000,4000,8000 points, through, for increasing q we observe a very slow build-up of the relative error. Our algorithm performs better as the number of q increases.

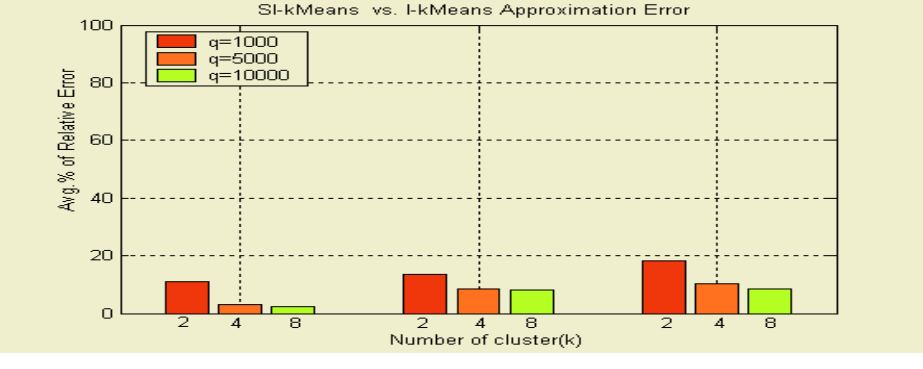

Fig. 3. Comparison of the clustering approximation error between STSI-kMeans and I-kMeans

The second measure of interest is the speedup, which measures how many times faster STSI-kMeans is when compared to I-kMeans.

$$speedup = \frac{\sum_{j=1}^{q} time_{\text{STSI-kMeans}}(w_j)}{\sum_{j=1}^{q} time_{\text{I-kMeans}}(w_j)} \tag{5}$$

Figure 4 shows the speedup that our algorithm achieves, which translates to one or two orders of magnitude faster execution than the offline I-kMeans algorithm (for the experiments we ran). The STSI-kMeans algorithm is 10-30 times faster than I-kMeans. We observe that the speedup increases significantly for decreasing k. This is because the amount of work that STSI-kMeans does remains almost constant, while I-kMeans requires lots of extra effort for smaller values of k. As expected, the speedup gets larger when we increase q.

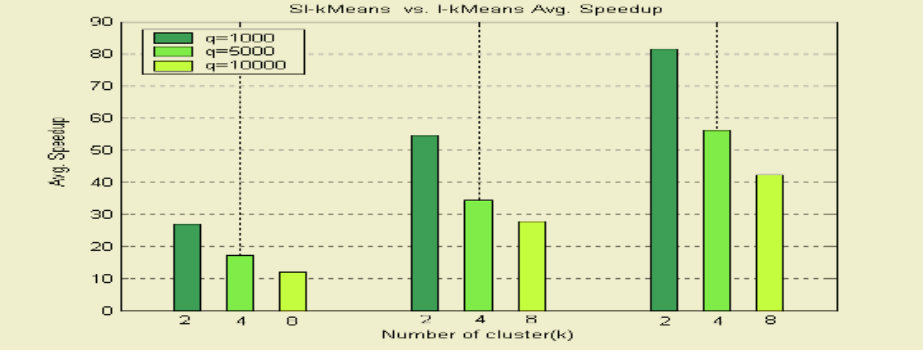

Fig. 4. Speedup of STSI-kMeans against I-kMeans

4 Conclusions

In this paper, we have presented firstly an approach to perform incremental clustering of time-series at various resolutions using the multi-resolution piecewise aggregate transform. The algorithm equipping a stopping criteria based on Hoeffding bound stabilizes at very early stages, eliminating the needs to operate on high resolutions. This approach resolves the dilemma associated with the choices of initial centers for k-Means and at which stage terminate the program for I-kMeans. This allows our algorithm to terminate the program at early stage with quality guarantee, thus eliminate the need to re-compute all the distances and significantly improves the execution time and clustering quality. We also expend our method to streaming time series environment. Our streaming time-series clustering algorithm works by leveraging off the nearest neighbors of the incoming streaming time series datasets and fulfill incremental clustering approach. Our experimental results based on several publicly available real data sets shows that significant performance improvement is achieved and produce high-quality clusters in comparison to the previous methods.

References

1. Lin, J., Vlachos, M., Keogh, E., & Gunopulos, D.: Iterative Incremental Clustering of Time Series. In proceedings of the IX Conference on Extending Database Technology (EDBT 2004). Crete, Greece. (2004) 14-18
2. Huhtala, Y., Kärkkäinen, J., & Toivonen. H.: Mining for Similarities in Aligned Time Series Using Wavelets. In Data Mining and Knowledge Discovery: Theory, Tools, and Technology. SPIE Proceedings Series Vol. 3695. Orlando, Florida. (1999)150 - 160
3. Struzik, Z. , Siebes, A.: The Haar Wavelet Transform in The Time Series Similarity Paradigm. In Proc 3rd European Conference on Principles and Practice of Knowledge Discovery in Databases. (1999)12-22
4. D. Carney, U. Cetinternel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, S. Zdonik.: Monitoring streams: A New Class of Data Management Applications. In Proc. 28th Int. Conf. on Very Large Data Bases, (2002) 215-226.
5. Keogh, E., Chakrabarti, K. Pazzani, M , Mehrotra, S.: Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. Journal of Knowledge and Information Systems. Vol. 3, No. 3. (2001) 263-286
6. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association (1963) 13-30
7. Yi, B. , Faloutsos, C.: Fast Time Sequence Indexing for Arbitrary Lp Norms. In proceedings of the 26th Int'l Conference on Very Large Databases. Cairo, Egypt, Sept 10-14. pp 385-394.l Database Management. Berlin, Germany, Jul 26-28. (2000)55-68.
8. Domingos, P., Hulten, G.: Mining High-Speed Data Streams. In: Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining, Boston, MA, ACM Press (2000) 71-80
9. Gama, J., Medas, P., Rodrigues, P.: Concept Drift in Decision-Tree Learning for Data Streams. In: Proceedings of the Fourth European Symposium on Intelligent Technologies and their implementation on Smart Adaptive Systems, Aachen, Germany, Verlag Mainz (2004) 218-225
10. L. Ralaivola, F. dAlche-Buc.: Incremental Support Vector Machine Learning: A Local Approach. In Proceedings of the Annual Conference of the European Neural Network Society. (2001) 322-329
11. Bay, S. D.: The UCI KDD Archive [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science. (1999)