

Creating Streaming Iterative Soft Clustering Algorithms

Prodip Hore, Lawrence O. Hall, and Dmitry B. Goldgof
Department of Computer Science and Engineering, ENB118
University of South Florida
4202 E. Fowler Ave.
Tampa, FL 33620-9951
{phore, hall, goldgof}@cse.usf.edu

Abstract—There are an increasing number of large labeled and unlabeled data sets available. Clustering algorithms are the best suited for helping one make sense out of unlabeled data. However, scaling iterative clustering algorithms to large amounts of data has been a challenge. The computation time can be very great and for data sets that will not fit in even the largest memory, only carefully chosen subsets of data can be practically clustered. We present a general approach which enables iterative fuzzy/possibilistic clustering algorithms to be turned into algorithms that can handle arbitrary amounts of streaming data. The computation time is also reduced for very large data sets while the results of clustering will be very similar to clustering with all the data, if that was possible. We introduce transformed equations for fuzzy-c-means, possibilistic c-means, the Gustafson-Kessel algorithm and show the excellent performance with a streaming fuzzy c-means implementation. The resulting clusters are both sensible and for comparable data sets (those that fit in memory) almost identical to those obtained with the original clustering algorithm.

Keywords: fuzzy, possibilistic, clustering, streaming, scalable

I. INTRODUCTION

There are an increasing number of large unlabeled data sets available. Some of these data sets arrive as streams with chunks of data becoming available at different times. The streams may consist of homogeneous data or there may be changes in the data stream over time. Understanding these types of data sets can be facilitated by clustering them into groups and labeling homogeneous groups. Some of the best-known clustering algorithms such as fuzzy-c-means, hard-c-means, possibilistic-c-means, and the Gustafson-Kessel (GK) algorithms [1], [2], [3], [4] are iterative algorithms which for practical purposes require all the data be in memory at one time. They are well understood and have been widely applied, but they are not immediately applicable to extremely large data sets or streaming data.

In this paper, we provide a framework for adapting iterative clustering algorithms which result in centroids to streaming data sets. The streams could be a result of treating an extremely large data set as a stream with each chunk sized to comfortably fit into the memory of the computer on which clustering will be done or it could be a time changing stream of data. It will be shown that the framework fits fuzzy-c-means, the GK algorithm and possibilistic c-means. Within

the algorithms, it is only necessary that weighted examples be introduced. The equations needed for this will be shown for the three algorithms discussed here. We'll argue that the framework also applies to other types of iterative clustering algorithms which have clusters described as centroids.

Experimental results will be shown using an implementation of fuzzy-c-means applied to a extremely large data set, but small enough that it will fit in 32 GB of memory so that we can cluster in the usual way for comparison purposes. Comparisons will show that the streaming algorithm results in almost identical partitions.

The paper proceeds with Section 2 on related work, Section 3 which introduces necessary mathematical modifications to the clustering algorithms for use in our streaming framework. Section 4 describes the process of wrapping the streaming framework around the iterative clustering algorithm. Section 5 describes data sets and Section 6 contains experimental results. Finally, Section 7 is a summary and discussion.

II. RELATED WORK

In [5], a multistage random sampling method was proposed to speedup fuzzy c means. There were two phases in the method. In the first phase, random sampling was used to obtain an estimate of centroids and then fuzzy c means (FCM) was run on the full data with these centroids initialized. A speed-up of 2-3 times was reported. In [6], speed up is obtained by taking a random sample of the data and clustering it. The centroids obtained then were used to initialize the entire data set. This method is similar to that in [5]; the difference is they used one random sample where in [5] they may use multiple random samples. In [7], another method based on sampling for clustering large image data was proposed, where the samples were chosen by the chi-square or divergence hypothesis test. It was shown that they achieved an average speed-up of 4.2 times on image data while providing a good final partition using 24% of the total data.

Another speed up technique for image data was proposed in [8]. In this method FCM convergence is obtained by using a data reduction method. Data reduction was done by

quantization, aggregating similar examples, which were then represented by a single weighted exemplar. The objective function of the FCM algorithm was modified to take into account the weights of the exemplars. However, the presence of similar examples might not be common in all data sets. They showed that it performs well on image data. In summary, the above algorithms attempt to speed up fuzzy c means either by reducing the number of examples through sampling or by data reduction techniques or by providing good initialization points to reduce the number of iterations. However, the above algorithms do not seem to address the issue of clustering large or very large data sets under the constraint of limited memory. Moreover, some of them address the speed up issues for image data only where the range of features may be limited. Some work on parallel/distributed approaches has been done, where multiple processors could be used in parallel to speed up fuzzy c means [9], [10]. In [11] a parallel version of the adaptive fuzzy Leader clustering algorithm has been discussed, whereas, in [12] an efficient variant of the conventional Leader algorithm known as ARFL (Adaptive rough fuzzy leader) clustering algorithm was proposed.

There has been research on clustering large or very large data sets [13], [14], [15], [16], [17], [18]. Birch [13] is a data clustering method for large data sets. It loads the entire data into memory by building a CF (Clustering Feature) tree, which is a compact representation of the data using the available memory. Then the leaf entries of the CF tree can be clustered to produce a partition. A hierarchical clustering algorithm was used in their paper. It provides an optional cluster refining step in which quality can be further improved by additional passes over the dataset. However, the accuracy of data summarization depends on available memory. It was pointed out in [14] that depending on the size of the data, memory usage can increase significantly as the implementation of Birch has no notion of an allocated memory buffer. In [14], a single pass hard c means clustering algorithm was proposed under the assumption of a limited memory buffer. They used various data compression techniques to obtain a compact representation of data. In [15], another single pass scalable hard c means algorithm was proposed. This is a simpler implementation of Bradley's single pass algorithm [14], where no data compression techniques have been used. They showed that complicated data compression techniques do not improve cluster quality much while the overhead and book-keeping of data compression techniques slow the algorithm down. Bradley's algorithm compresses data using a primary compression and secondary compression algorithm. In primary compression, data points which are unlikely to change membership were put into a discarded set. The remaining points were then over clustered with a large number of clusters compared to the actual number of clusters. This phase is known as secondary compression and its objective is to save more buffer space by representing closely packed points by their clusters. A tightness criterion was used to detect clusters which can be used to represent points. These sub-clusters were further joined by an agglomerative

clustering algorithm provided they are still tightly packed after merging. So, in summary Bradley's implementation has various data compression schemes which involve overhead and book-keeping operations. Farnstorm's single pass [15] algorithm is basically a special case of Bradley's where after clustering, all data in memory is put into a discard set. A discard set is associated with every cluster, and it contains the sufficient statistics of data points in it. The discarded set is then treated as a weighted point in subsequent clustering. Other relevant single pass algorithms for the crisp case can be found in [19] and [20]. Compressing data was also studied in [21], where the number of templates needed by the probabilistic neural network (PNN) was reduced by compressing training examples with an exemplar. This was done by estimating the probability density functions for the PNN with Gaussian models.

Recently in [22], a sampling based method has been proposed for extending fuzzy and probabilistic clustering to large or very large data sets. The approach is based on progressive sampling and is an extension of eFFCM [7] to geFFCM, which can handle non-image data. However, the termination criteria for progressive sampling could be complicated as it depends upon the features of the data set. They used 4 acceptance strategies to control progressive sampling based on the features of the data. The first strategy (SS1) is to accept the sampled data when a particular feature signals termination. The second one (SS2) is to accept when any one of the features signals termination. The third one (SS3) is to accept when all the features sequentially signal termination and the last one accepts (SS4) when all the features simultaneously signal termination. However, the method could be complicated for a large number of features and the sample size could grow large also.

In [20], a streaming algorithm for hard- c -means was proposed in which data was assumed to arrive in chunks. Each chunk was clustered using a LOCALSEARCH algorithm and the memory was freed by summarizing the clustering result by weighted centroids. This is similar to the method of creating discard set in the single pass hard c means algorithm [15]. Finally, the weighted centroids were clustered to obtain the clustering for the entire stream. We also summarize clustering results in a similar way; however, the objective function optimized by the soft clustering algorithms are not the same as the crisp clustering algorithms. The difference between [20], [15] and our approach is in the fact that in fuzzy/soft clustering an example may not completely belong to a particular cluster. Our method of summarizing clustering results involves the softness of the clustering solutions (fuzzy membership matrix in case of FCM), which does not exist for the crisp cases.

III. WEIGHTED EXAMPLE CALCULATION FOR CLUSTERING ALGORITHMS

The underlying idea behind our streaming clustering framework is to create clusters of the examples encountered thus far and the centroids serve as representatives of the examples assigned to the cluster. The number of examples

assigned to the cluster serve as the weight for a new weighted example to be used in conjunction with the next set of unlabeled examples. Hence, any clustering algorithm must be able to incorporate weighted examples. In fact, they will be integer weighted and in principle simply amount to having a number of coincident examples. So, one can imagine they should have no negative effect on any of the characteristics of the algorithm and should be reasonably straightforward to incorporate. Below, we show how to incorporate them for three iterative clustering algorithms.

A. Fuzzy c -means

The objective function (J_m) minimized by FCM is defined as follows:

$$J_m(U, V) = \sum_{i=1}^c \sum_{k=1}^n U_{ik}^m D_{ik}(x_k, v_i) \quad (1)$$

U and V can be calculated as:

$$U_{ik} = \frac{D_{ik}(x_k, v_i)^{\frac{1}{1-m}}}{\sum_{j=1}^c D_{jk}(x_k, v_j)^{\frac{1}{1-m}}} \quad (2)$$

$$v_i = \frac{\sum_{j=1}^n (u_{ij})^m x_j}{\sum_{j=1}^n (u_{ij})^m} \quad (3)$$

Where,

U_{ik} : is the membership value of the k^{th} example, x_k , in the i^{th} cluster.

v_i : is the i^{th} cluster centroid.

n : is the number of examples

c : is the number of clusters

$D_{ik}(x_k, v_i) = \|x_k - v_i\|^2$: is the norm. We have used the Euclidean distance.

B. Possibilistic c -means

The objective function (J_m) minimized by PCM is defined as follows:

$$J_m(U, V, \eta) = \sum_{i=1}^c \sum_{k=1}^n U_{ik}^m D_{ik}(x_k, v_i) + \sum_{i=1}^c \eta_i \sum_{k=1}^n (1 - U_{ik})^m \quad (4)$$

U and V can be calculated as:

$$U_{ik} = [1 + (\frac{D_{ik}^2}{\eta_i})^{\frac{1}{m-1}}]^{-1} \quad (5)$$

$$v_i = \frac{\sum_{j=1}^n (u_{ij})^m x_j}{\sum_{j=1}^n (u_{ij})^m} \quad (6)$$

Where,

U_{ik} : is the membership value of the k^{th} example, x_k , in the i^{th} cluster.

v_i : is the i^{th} cluster centroid.

n : is the number of examples

c : is the number of clusters

$D_{ik}(x_k, v_i) = \|x_k - v_i\|^2$: is the norm.

$\eta = (\eta_1, \eta_2, \dots, \eta_c)^T$; $\eta_i \in \mathbb{R}^+$ is the i^{th} penalty term. Each η_i is a fixed user specified positive weight.

The weighted version of PCM (allowing examples to have weights) is:

$$J_w(U, V) = \sum_{i=1}^c \sum_{k=1}^n U_{ik}^m w_k D_{ik}(x_k, v_i) + \sum_{i=1}^c \eta_i \sum_{k=1}^n w_k (1 - U_{ik})^m \quad (7)$$

U and V can now be calculated as:

$$U_{ik} = [1 + (\frac{D_{ik}^2}{\eta_i})^{\frac{1}{m-1}}]^{-1} \quad (8)$$

$$v_i = \frac{\sum_{j=1}^n (u_{ij})^m w_j x_j}{\sum_{j=1}^n w_j (u_{ij})^m} \quad (9)$$

where w_j is the weight of the j^{th} example.

C. Gustafson-Kessel (GK) clustering

This is the same as FCM except that $D_{ik}(x_k, v_i) = \|x_k - v_i\|_{A_i}$, where $A_i = [\rho_i \det(C_i)]^{1/p} C_i^{-1}$, $1 \leq i \leq c$ and C_i is the fuzzy covariance matrix of cluster i ,

$$C_i = \sum_{k=1}^n U_{ik}^m (x_k - v_i)(x_k - v_i)^T / \sum_{k=1}^n U_{ik}^m \quad (10)$$

where $1 \leq i \leq c$; $m \geq 1$

$\rho_i = 1, \forall i$ is a typical choice. The change to C_i shown in (11) below takes care of the weighting issue for the distance function.

$$C_i = \sum_{k=1}^n U_{ik}^m w_k (x_k - v_i)(x_k - v_i)^T / \sum_{k=1}^n w_k U_{ik}^m \quad (11)$$

where $1 \leq i \leq c$; $m \geq 1$

IV. STREAM CLUSTERING FRAMEWORK

There are two major issues when thinking about clustering a stream of data. The first is the size of the data which may often exceed the size of main memory (even if the stream stops running and the data stops coming). The second is that the characteristics, and in our case makeup of the clusters, can change over time in the stream. So, one needs a method of clustering all the data and recognizing and responding to underlying changes in distribution.

Our streaming algorithm requires that we use a subset of the data during the clustering process. To adapt to changes in the data, we can keep varying amount of history. The history can be complete if, for example, you are treating a very large data set as a stream or it can be limited to enable fast detection and adaptation to changes in the structure of the underlying streaming data.

As FCM assumes that you know the required number of clusters beforehand. Our algorithm assumes each chunk of data has a class distribution that loosely approximates the overall distribution. That is, the first is not all one class, the second another class, etc. Extensions remain to be made to handle these types of problems.

- For $t = 1$ to m
 - if ($t > 1$) then add 1 or more sets of weighted centroids to data to be clustered.
 - cluster data set of $n_i + hc$ examples, where h indicates how much history is used, into c classes with any clustering algorithm that will result in centroids that can be weighted (FCM, GK and PCM for example)
 - weight the final centroids using (12) or (13), which results in c weighted examples
 - end for
- output the final centroids which can be used to group any examples.

Fig. 1. Framework for streaming clustering algorithms with m chunks of data.

The general idea now is that each chunk of data will result in a set of cluster centroids. The centroids can be weighted by the number of examples they fuzzily represent. You can add them to be next batch of examples (chunk of data). You can take only one set of weighted centroids representing all the data previously seen or h sets of centroids where $1 \leq h < PS$ where PS is the number of previously seen chunks of data. If h is 1, only the last chunk of data serves as history and that is probably too little history. A good method for determining h is needed, but we note that it is likely domain dependent and a domain expert may provide the value.

We assume n_i examples arrive at time t_i and there are m chunks of data clustered where $1 \leq i \leq m$. The examples will be clustered into c classes. The n_i can be approximately the same for all i but this is not necessary. At t_i we cluster the weighted examples and obtain c new weighted examples which are the final centroids. The weight, w_i , for the i^{th} cluster center is calculated as

$$w_i = \sum_{j=1}^{n_i+c} U_{ij}, 1 \leq i \leq c, \quad (12)$$

or

$$w_i = \sum_{j=1}^{n_i} U_{ij}, 1 \leq i \leq c \quad (13)$$

depending on whether you want full history (12) (add c heavily weighted points) or intend to use several weighted points in a partial history (13).

The general training algorithm is now shown in Figure 1. You would use (13) to weight the examples if using limited history (for example $h = 2$ or $h = 3$) to capture drift. It should be noted that for capturing drift the weight is obtained from the current chunk only (13). In equation (12) h is always 1, but it still summarizes the entire history by condensing all past history in only c weighted points (weights increases with time). So, using equation (12) means the entire history is summarized by a single set of centroids, which is a low resolution summarization generally suited for less evolving data.

V. DATA SETS

A single large data set was used. The MRI-2 data set is created by concatenating 2 Volumes of human brain MRI data. Each Volume consists of 144 slices of MR images of size 512X512 from modalities T1, PD, and T2. The magnetic strength is 1.5 Tesla. For this data set air was not removed and the total size of data set was slightly above 75 million (75,497,472 examples, 3 features). We clustered into 10 classes.

All experiments were run on UltraSPARC-III with 8 processors each of 750 MHz. There was 32GB of main memory. None of the programs were multithreaded.

The value m was 2. We used the reformulated the FCM objective function R_m [23] to avoid keeping all the examples when doing comparisons. It is:

$$R_m(V) = \sum_{k=1}^n \left(\sum_{i=1}^c D_{ik} (x_k, v_i)^{\frac{1}{1-m}} \right)^{(1-m)} \quad (14)$$

We compare results using the difference in quality in equation 15.

$$DQ = \left(\frac{m_2 - m_1}{m_1} \right) * 100 \quad (15)$$

So, a positive value means average value of R_m of FCM is better (lower) while negative value means R_m of our streaming framework was better.

VI. EXPERIMENTS

We treated the data as a stream but kept complete memory. So, we added c examples weighted by (12) at each time step beyond 1.

The MRI-2 experiment consists of 75,497,472 examples and it takes about 4 to 5 days for FCM to complete each experiment. We clustered the whole data 10 times using FCM. So, we compare it with the first 10 experiments of streaming and thus the average results are computed on 10 experiments only. On this data set we loaded only 1% of the data for streaming. The average quality difference of FCM and streaming was 0.0053% and the speed up obtained was 69.65 times. The quality difference and speed up obtained on this 75 million example data set was excellent. There are similar good results available for other data sets [18].

VII. DISCUSSION AND CONCLUSIONS

In this paper, we have outlined a general framework for clustering streaming data using soft clustering algorithms. The algorithms must enable cluster centroids to be extracted with weights based on the number of examples partially assigned them. We have shown the necessary modifications for three types of fuzzy/possibilistic clustering algorithms FCM, GK and PCM.

The algorithm requires that you know the number of cluster centers and that most chunks of data are reasonably representative of the class mixture. It is clearly possible to create scenarios where the data is ordered in a strange way and then the algorithm will have unpredictable results.

History can be maintained by adding one or more centroids from previous applications of the clustering algorithm to chunks of data. An approximation to all the data being clustered at once can be obtained by using one set of weighted centroids which are incrementally updated after every chunk of examples has been clustered. The algorithm allows for an infinite stream by clustering only as much data as can fit in memory at a given time. By keeping a limited history it will adapt to changes over time. The framework can be applied to other types of clustering algorithms that results in centroids.

Results have been shown from a 75 million example data set representative of a magnetic resonance image scan of the human brain using three weighted features. The results were almost exactly same as clustering with all the data using FCM but done about 70 times faster (the time required was just over an hour vs. 5 days).

VIII. ACKNOWLEDGEMENTS

This research was partially supported by the National Institutes of Health under grant number 1 R01 EB00822-01 and by the Department of Energy through the ASCI PPPE Data Discovery Program, Contract number: DE-AC04-76DO00789.

REFERENCES

- [1] E.E. Gustafson and W.C. Kessel. Fuzzy Clustering with a Fuzzy Covariance Matrix. *Proc. 18th IEEE Conference on Decision and Control (IEEE CDC, San Diego, CA)*, page 761766, 1979.
- [2] J.C Bezdek. Pattern Recognition with Fuzzy Objective Function Algorithms. *Plenum Press, New York*, 1981.
- [3] J. B. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press*, pages 281–297.
- [4] R. Krishnapuram and J.M. Keller. The possibilistic c-means: insights and recommendations. *IEEE Trans. Fuzzy Systems*, 4:385–393, 1996.
- [5] Cheng T.W., Goldgof D.B., and Hall L.O. Fast fuzzy clustering. *Fuzzy Sets and Systems*, pages 49–56, 1998.
- [6] David Altman. Efficient Fuzzy Clustering of Multi-spectral Images. *IEEE FUZZ*, 1999.
- [7] N.R. Pal and J.C. Bezdek. Complexity reduction for large image processing. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, pages 598–611, 2002.
- [8] S.Eschrich, J.Ke, L.O.Hall, and D.B. Goldgof. Fast Accurate Fuzzy Clustering through Data Reduction. *IEEE Transactions on Fuzzy Systems*, 11:262–270, 2003.
- [9] P. Hore, Lawrence Hall, and Dmitry Goldgof. A Cluster Ensemble Framework for Large Data sets. *IEEE International Conference on Systems, Man, and Cybernetics*, 2006.
- [10] S. Rahmi, M. Zargham, A. Thakre, and D. Chhillar. A Parallel Fuzzy C-Mean Algorithm for Image Segmentation. *Fuzzy Information*, 1:234–237, 2004.
- [11] Alfredo Petrosino and Mauro Verde. P-AFLC: a parallel scalable fuzzy clustering algorithm. *ICPR*, 1:809–812, 2004.
- [12] S. Asharaf and M. Narasimha Murty. An adaptive rough fuzzy single pass algorithm for clustering large data sets. *Pattern Recognition*, 36, 2003.
- [13] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH : An Efficient Data Clustering Method for Very Large Databases. *Proc. ACM SIGMOD Int'l. Conf. on Management of Data, ACM Press*, pages 103–114, 1996.
- [14] P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. *Proc. 4th Int'l. Conf. Knowledge Discovery and Data Mining, AAAI Press*, pages 9–15, 1998.
- [15] F. Farnstrom, J. Lewis, and C. Elkan. Scalability of Clustering Algorithms Revisited. *SIGKDD Explorations*, pages 51–57, 2000.
- [16] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An Efficient Clustering Algorithm for Large Databases. *In Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 73–84, 1998.
- [17] Raymond T. Ng and Jiawei Han. CLARANS: A Method for Clustering Objects for Spatial Data Mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.
- [18] Prodig Hore, Lawrence Hall, and Dmitry Goldgof. Single Pass Fuzzy C Means. *FUZZ-IEEE (accepted)*, 2007.
- [19] Chetan Gupta and Robert Grossman. GenIc: A Single Pass Generalized Incremental Algorithm for Clustering. *Proceedings of the Fourth {SIAM} International Conference on Data Mining (SDM)*, pages 22–24, 2004.
- [20] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-Data Algorithms for High-Quality Clustering. *Proceedings of IEEE International Conference on Data Engineering*, 2002.
- [21] Traven H. G. C. A neural network approach to statistical pattern classification by semiparametric estimation of probability density functions. *IEEE Transactions on Neural Networks*, 2:366–377, 1991.
- [22] Richard J. Hathaway and James C. Bezdek. Extending Fuzzy and Probabilistic Clustering to Very Large Data Sets. *Journal of Computational Statistics and Data Analysis*, 2006.
- [23] Richard J. Hathaway and James C. Bezdek. Optimization of Clustering Criteria by Reformulation. *IEEE Transactions on Fuzzy Systems*, 3:241–245, 1995.