Requirements for Clustering Data Streams

Daniel Barbará

George Mason University,
ISE Dept MSN 4A4,
Fairfax, VA 22030.

dbarbara@gmu.edu

ABSTRACT

Scientific and industrial examples of data streams abound in astronomy, telecommunication operations, banking and stock-market applications, e-commerce and other fields. A challenge imposed by continuously arriving data streams is to analyze them and to modify the models that explain them as new data arrives. In this paper, we analyze the requirements needed for clustering data streams. We review some of the latest algorithms in the literature and assess if they meet these requirements.

Keywords

Data streams, clustering, outliers, tracking changing models.

1. INTRODUCTION

Organizations today accumulate data at an astonishing rate. This fact brings new challenges for data mining. Finding out when patterns change in the data opens the possibility of making better decisions and discovering new interesting facts. The challenge is to design algorithms that can track changes in an incremental way without making growing demands on memory and processing resources.

The kind of data sets that arise in recent applications is appropriately referred to as *data streams*, i.e., a continuous stream of new data points that makes operating on the past portions of the data repeatedly an impractical proposition.

In this paper we examine the requirements to track changes in clustering models for a data stream. Clustering is a widely used technique that helps uncovering structures in data that were previously not known. Finding changes in clusters as new data is collected can prove fruitful in scenarios like the following:

- Tracking the evolution of the spread of illnesses. As new cases are reported, finding out how clusters evolve can prove crucial in identifying sources responsible for the spread of the illness.
- Tracking the evolution of workload in an e-commerce server, which can help in dynamically fine tune the server to obtain better performance.
- Tracking meteorological data, such as temperatures registered throughout a region, by observing how clusters of spatial-meteorological points evolve in time.
- Tracking network data to study changes in the traffic patterns and possible intrusions.
- Tracking data feeds from sensor applications.

In this paper we explore the requirements that are needed for a clustering algorithm that can successfully process data streams tracking changes on the clustering models. The paper is organized as follows. Section 2 talks about the requirements. Section 3 presents a way of deciding when we are in need for new clusters, as new data arrives. Section 4 takes a look at published algorithms that have been designed with incremental processing of points in mind, and determines if these algorithms meet the requirements posed here. Section 5 points at a few open directions of research for data stream clustering. Finally, Section 6 offers conclusions.

2. REQUIREMENTS FOR CLUSTERING DATA STREAMS

The nature of data streams calls for three basic requirements in clustering algorithms:

- Compactness of representation
- Fast, incremental processing of new data points
- Clear and fast identification of ``outliers''

We shall explain each one of these requirements in turn in the following sub-sections.

2.1 Compactness of representation

Since data streams are continuously arriving to a site, any clustering algorithm that aims to process them cannot afford the luxury of a lengthy description of the clusters found so far. In particular, basing the decision of where to put the next point on the list of clusters found so far is not an option. This list grows unbounded as new points arrive, and therefore would exhaust any main memory resources that the machine possesses. Since we insist on the capacity of processing new points on-line, checking the new point membership against secondary memory representations of the current clusters is not possible. Therefore, a data stream clustering algorithm must provide a representation of the clusters found that is not only compact, but it does not grow appreciably with the number of points processed. (Notice that even a linear growth is intolerable.)

2.2 Fast, incremental processing of new data points

The need for speed and incremental processing is obvious once we consider the on-line nature of the task. However, this is far from being a trivial requirement. The placement of new points has to be based on the evaluation of a function. This function has to meet the following two conditions:

- The placement of new points cannot be decided by a function that requires comparison with all the points that have been processed in the past.
- The function that decides the placement of new points has to exhibit good performance.

The first condition argues again for a compact representation of the current clusters. But it also argues for a function that can be evaluated using that representation. The second condition merely addresses the need for a function with good complexity: e.g., one that is linear on the size of the representation chosen.

2.3 Clear and fast identification of ``outliers''

This requirement is probably the least intuitive of the three. We have placed the word outliers between quotes, since by that term we mean to imply points that do not fit well in any of the clusters that the algorithm has found so far.

The reason for including this requirement can be explained by the dynamic nature of data streams. It is highly likely that the data stream will exhibit different trends during its lifetime, and consequently the points received at any given time may not fit well under the clustering model the algorithm has identified so far.

Thus, the algorithm employed must be able to detect this in a clear and speedy manner. In fact, the function that evaluates the point placement must have within its range a value for ``outlier.''

Associated with this requirement is the need to decide what to do with these outliers. We view this as an application dependent decision: in some cases, if sufficient outliers are found we may want to abandon the old clusters for new ones. An example of this could be a data stream of weather data points: sufficient outliers indicate a new trend that needs to be represented by new clusters. In others, we may have to redefine the boundaries of existing clusters. An example of this case could be a data stream of spotted cases of an illness. Outliers may indicate the spread of the epidemics over larger geographical areas.

3. Tracking clustering models

As new points arrive, we need to determine if a new clustering model is needed. Assuming we have a clear way of determining outliers by the algorithm used (as specified in Section 2), we can keep track of how many outliers we have seen in the recent past. To determine if too many of them have been observed to make the current clustering model unfit, we can resort to the use of Chernoff bounds [3], by defining a random variable *Xi* (each *i* corresponding to a new point) whose value is 0 if the point is an outlier and 1 otherwise. (A good analogy is to consider the clusters as ``bulls eyes'' and the new point as a dart; if the dart ``hits'' the clusters, then we assign a value of 1 to Xi, otherwise we assign a 0.)

Since we do not know the value of *p*, the probability of a ``hit'' by a point (i.e., the probability of *Xi* being 1), we have to estimate it by dividing sum of the *Xi* variables, *X*, by the number of points we have tried to cluster, *n*. The estimate will be bound to the real value by using Equation (1).

$$\Pr[|\frac{X}{n} - p| \le \varepsilon] > 1 - \delta \quad (1)$$

In Equation (1), ε is the desired deviation of the estimate with respect to the real value, while δ indicates an arbitrarily small probability. In essence, Equation (1) establishes that the estimate and the real value can be made to be arbitrarily close by the choice of δ.

Using the Chernoff inequality, we can bound the estimate of the success probability, by bounding the probability that the estimate $\frac{X}{n}$ surpasses $(1+\varepsilon)p$, as shown in Equation (2). The variable *n* indicates the number of points that we have tried (successfully or unsuccessfully) to cluster.

$$\Pr[\frac{X}{n} > (1+\varepsilon)p] \le e^{(-pne^2/3)}$$

$$\Pr[\frac{X}{n} > (1+\varepsilon)p] \le e^{(-pn\varepsilon^2/3)} \quad (2)$$

It can be proven (see [11]) that Equation (1) will hold if the number of successful attempts to cluster points, *s* (i.e., number of times that the random variable is 1) is bounded by Equation (3), while *n* is bounded by Equation (4).

$$s > \frac{3(1+\varepsilon)}{\varepsilon^2}\ln(\frac{2}{\delta}) \quad (3)$$

$$n \le \frac{3(1+\varepsilon)}{(1-\varepsilon)\varepsilon^2 p}\ln(\frac{2}{\delta}) \quad (4)$$

These two bounds are all we need to decide whether the current clustering model is valid under the new data we are receiving. If after processing *n* points (given by Equation (4)), we are able to successfully cluster at least *s* of them, the clustering model is still valid. Otherwise, it is time to produce a new model.

Once that it has been established that a new clustering model is needed, the decision on how to proceed is application dependent. More precisely, two actions are possible:

- Re-cluster the entire set of points seen so far, including the last *n* points that prompted the decision to re-cluster. Obviously, this is the most expensive course of action. However, given a good, compact representation of the clusters obtained before the string of unsuccessful attempts, it is possible to use that representation, plus the ``outliers'' to produce new clusters, without having to look again at all the previous points.
- Discard the old clusters, and produce a new set of clusters, by considering only the last *n* processed points.

Following the latter course of action, we have been able to use these bounds to effectively track clusters in data streams (see [2]). Notice that after any of the two paths of action is taken, the need to re-evaluate the number of clusters that the algorithm aims to find (a common input value for most of the algorithms in the literature) exists.

4. Comparison of Existing Algorithms

This section presents a summary of a series of published algorithms whose aim has been to incrementally cluster points in a data set (not necessarily a data stream). This should not be viewed as an extensive study of all published clustering algorithms, but only of those that have aimed to incrementally cluster data sets. On purpose we have not included any algorithm that uses sampling to scale to large data sets, since sampling would defeat the aim of clustering data streams. We study whether or not these algorithms meet the requirements established in the previous section.

4.1 BIRCH

BIRCH [12] builds a hierarchical data structure, the CF-tree –a height-balanced tree-, to incrementally cluster incoming points. Each node in the tree is defined by a CF-vector of statistical measures representing the set of nodes under that node. BIRCH tries to come up with the best clusters with available main memory, while minimizing the amount of I/O required. The CF-tree can be thought as storing a hierarchical clustering model in an agglomerative fashion, making it possible to cluster sub-clusters represented by their CF-vectors. Results can be improved by allowing several passes over the data set, but in principle one pass suffices to get a clustering, so the complexity of the algorithm is *O(N).* Since each node in the tree has predefined limited capacity, the clusters do not always correspond to natural shapes. (In [8], it is reported that BIRCH does not perform well for non-spherical clusters; our own experience confirms that observation.)

- Compactness: BIRCH is designed to store the cluster representation in secondary memory (the CF-tree, is, after all analogous to a B-Tree). Even though the design of the algorithm stores the initial (seed) clusters in a main-memory CF-tree, it is clear that after that, the CF-tree should be let to grow in secondary memory. Therefore, the representation is not as compact as it is required to process data streams. This fact is aggravated as the dimensionality of the set increases. In our experience, the secondary memory representation grows very fast as the set goes beyond 3 or 4 dimensions.
- Function: Even though it tries to minimize the I/O requirements of clustering a new point, it still takes a considerable amount of time to do so. The processing of points is better served in batches to try to amortize the overall cost.
- Outliers: A number of bytes is reserved in BIRCH to handle outliers that do not fit well the clusters defined by the CF-tree. BIRCH periodically re-evaluates these outliers to see if they can be absorbed in the current tree without making it grow in size. Potentially, this features can be used to track changes in the clustering model, as exposed in Section 3.

4.2 COBWEB

COBWEB [4] is an incremental clustering technique that falls under the class of conceptual clustering algorithms (intended to discover understandable patterns in data). COBWEB uses the category utility function [6] to guide classification and tree formation. COBWEB keeps a hierarchical clustering model in the form of a classification tree. Each node contains a probabilistic description of the concept that summarizes objects classified under that node. For a new point, COBWEB descends the tree along an appropriate path, updating the counts in the interior nodes along the way and looks for the best node to place the point on, using the category utility function.

- Compactness: This classification tree is not height-balanced which often causes the space (and time) complexity to degrade dramatically. This makes COBWEB an ill choice for data streams clustering.
- Function: again, due to the cluster representation problem, the time complexity of adding a new point to the clusters might degrade dramatically.
- Outliers: COBWEB analyzes the result of placing a new point on a new node, created specifically for this point and computes (using the category utility function) whether this is a better choice than placing the point in one of the current clusters. In that way, outliers can be identified.

4.3 STREAM

By this name we denote the algorithms described in [5],[10], which aim to provide guaranteed performance, i.e., whose solution is guaranteed to be no worse than a number of times the optimal. The optimal solution (whose finding is intractable) is understood to be the one that minimizes the sum of square distance measure (SSQ), i.e., the sum of the square of the distances of points to the *k* cluster medians or centroids. This is an objective similar to the one K-Means aims to minimize, following an iterative heuristic. However, K-Means does not guarantee any bounds. Notice that by minimizing SSQ, the algorithm tends to find hyper-spheres as clusters, independently of the true nature of the data clusters, which can be of arbitrary shape.

STREAM processes data streams in batches of points, each fitting in main memory. For each batch B_i, STREAM clusters the points in it, and then, retaining the weighted cluster centers C_i (the centroids weighted by the number of points attracted to them). C_i is a set of centroids corresponding to the *i*-th batch of points. Then STREAM clusters the weighted centroids retained for each batch examined so far, i.e., $C_0,\ldots,C_i$, obtaining a clustering model. In both instances (the clustering of the batch and the clustering of the batches' centroids), STREAMS uses an algorithm, LOCALSEARCH, which runs in asymptotically linear time with respect to the number of points. The algorithm LOCALSEARCH insures that the solution is bounded by no more than a constant to the optimal one (in terms of SSQ).

- Compactness: The representation of current clusters is compact, as it requires only the list of the C_i sets, i.e., the set of centroids corresponding to each batch B_i. The size of this representation, however, increases as more batches are processed, since the number of sets of centroids is always increasing.
- Function: the function to incrementally cluster the items runs in time asymptotically linear with respect to the number of points. For the first iteration, to find the clusters in the current batch, this time is always the same (if the batches are of equal size). However, the time taken for the second iteration, i.e., the clustering of centroid sets increases without bound as more batches of the data stream arrive. Moreover, even though LOCALSEARCH is an asymptotically linear algorithm, it is recognized by its authors [10] to take longer than K-Means to find a bounded solution.
- Outliers: There are no provisions in [10] for outliers. If a sufficient number of them occur in a new batch, the centroids found for that batch will be substantially different from the centroids for previous batches. If the same number of clusters is input to LOCALSEARCH, the procedure will find hyper-spheres that capture both the old and the new centroid sets. This, in fact, may be a distortion of the real clusters in the data, and will effectively mask the emerging trend of clusters.

4.4 Fractal Clustering

This algorithm, presented in [1] defines clusters as sets of points that exhibit self-similarity [7]. Fractal Clustering (FC) clusters points incrementally, placing them in the cluster in which they have the minimal *fractal impact*. That is, the cluster that changes its fractal dimension in the least when the point is placed in it. FC has the capacity of finding clusters of arbitrary shape.

- Compactness: FC defines a grid of boxes (hyper-cubes whose dimensionality is equal to that of the points in the data set) at the lowest range of measurements (defined by the size of each box) and keeps count of the population of each box as points are being clustered. FC does not need to keep the actual points after they are clustered, since the box populations suffice to compute the fractal dimension (for example, by using a box-counting plot). Populations of the boxes at higher ranges are simply calculated by aggregating those on the lowest range. As such, the size of this representation does not depend on the number of points already processed, but rather on the dimensionality of the set and the size of the smallest box (at the lowest range). Our experiments have shown that as little as 2 Megabytes are enough to perform clustering for a 10-dimensional data set. Of course, as the number of dimensions grows, so does the number of boxes involved in the representation. However, it is also true that the number of populated boxes in high dimensional spaces is small compared to the total number of boxes. By keeping only the populated boxes, one can save a considerable amount of space. Our experiments show that even discarding boxes that have small populations (e.g., 1 point), the results remain consistently good.
- Function: whenever a new point needs to be clustered, the function that decides where this point is placed is based on the computation of the new fractal dimension of each cluster considering that the point is placed on it. Again, since FC uses box populations (as opposed to the set of points already clustered), the function takes time proportional to the number of boxes (which in turn is dependent on the dimensionality of the set). Again, the higher the dimensionality, the more boxes there exist, but since the computation involves only populated boxes, the complexity remains very acceptable. Moreover, recent work has shown that it is possible to compute this function in linear time with respect to the boxes (see [9]).
- Outliers: FC defines as outliers those points whose fractal impact exceeds a certain threshold. Intuitively, if a point changes the fractal dimension of every cluster too much, it is considered an outlier.

5. Directions

These are the research directions that we consider fruitful for future work in the area of clustering data streams:

- Finding appropriate ways of representing and dealing with the evolution of clusters in a data stream: as new points indicate the need for other clusters to be formed, it is important to have an automatic way to decide what to do with the clusters we have found so far. Although, it is likely that the decision will depend on the nature of the data at hand, automating the process as much as possible is a very desirable goal. Moreover, a way to present the evolution of the clusters to the user would be extremely useful, since that information is likely to be usable knowledge.
- Finding tighter bounds for the process of tracking cluster changes: Chernoff bounds are known to be conservative in their estimates. Finding alternative,

tighter bounds that can do the job is an important area of research.

- Successfully applying the techniques to real data sets: This last point requires the collaboration of data mining researchers with domain experts, in order to carefully evaluate the results and determine if they produce usable knowledge for the application at hand.

6. Conclusions

In this paper we have summarized a simple set of conditions for data stream clustering algorithms. An important requirement, often ignored by algorithm designers, is the need for a clear separation of outliers in the data stream, as sufficient number of these might indicate that a change in the clustering model is needed. We have provided analytical tools to effectively track these changes. Out of the algorithms reviewed, FC is the only one so far that completely fulfills the requirements. We are currently engaged in developing a similar algorithm to process categorical data (which uses entropy as the optimizing criteria). We also aim to produce a technique that can cope with mixed (numerical and categorical) data, by merging results of an algorithm (like FC) that deals with numerical data with those of an algorithm for categorical data.

7. Acknowledgments

I would like to acknowledge Ping Chen, Julia Couto, and Yi Li, who have taken an active role in designing and implementing FC and the upcoming categorical clustering algorithm. My interaction with them has helped enormously in shaping the ideas expressed in this paper.

8. REFERENCES

[1] Barbará D., and Chen, P. Using the Fractal Dimension to Cluster Datasets. Proceedings of the ACM-SIGKDD International Conference on Knowledge and Data Mining, Boston, August 2000.

[2] Barbará D., and Chen, P. Tracking Clusters in Evolving Data Sets. Proceedings of FLAIRS'2001, Special Track on Knowledge Discovery and Data Mining , Key West, May 2001.

[3] Chernoff, H. A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations. Annals of Mathematical Statistics, Vol. 23, pages 493-509, 1952.

[4] Fisher D.H. Iterative Optimization and Simplification of Hierarchical Clusterings. Journal of AI Research, Vol. 4, pages 147-180, 1996.

[5] Guha S., Mishra N., Motwani R., and O'Callaghan L. Clustering data streams. Proceedings of the Annual Symposium on Foundations of Computer Science November 2000.

[6] Gluck M.A., and Corter J.E. Information, uncertainty, and the utility of categories. Proceedings of the Seventh Annual Conference of the Cognitive Science Society, Irvine, CA, 1985.

[7] Schroeder M. Fractal, Chaos, Power Laws: Minutes from an Infinite Paradise. W.H. Freeman and Company, 1991.

[8] Sheikholeslami G., Chatterjee S., and Zhang A. WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. Proceedings of the 24th Very Large Data Bases Conference, 1998.

[9] Traina, A., Traina, C., Papadimitriou S., and Faloutsos, C. Tri-Plots: Scalable Tools for Multidimensional Data Mining. Proceedings of the 7^{th} ACM SIGKDD International Conference on Knowledge and Data Mining, San Francisco, August 2001.

[10] O'Callaghan L., Mishra N., Meyerson A., Guha S., and Motwani R. High-Performance Clustering of Streams and Large Data Sets. International Conference on Data Engineering (ICDE) 2002 (to appear).

[11] Watanabe, O. Simple Sampling Techniques for Discovery Science. IEICE Transactions on Inf. & Syst., Vol. E83-D, No. 1, January, 2000.

[12] Zhang T., Ramakrishnan R., and Livny M. "BIRCH: A Efficient Data Clustering Method for Very Large Databases. Proceedings of the ACM SIGMOD Conference on Management of Data, Montreal, Canada, 1996.

About the authors:

Daniel Barbará is an Associate Professor in the Information and Software Engineering Department at George Mason University. His current research interests are Data Mining and Data Warehousing. Previously, he has worked in Bell Communication Research and Panasonic Laboratories. He earned a Ph.D. from the Computer Science Department of Princeton University in 1985.