Clustering from Data Streams

João Gama; University of Porto

abstract

Clustering is one of the most popular data mining techniques. In this article, we review the relevant methods and algorithms for designing cluster algorithms under the data streams computational model, and discuss research directions in tracking evolving clusters.

Definition

Clustering is the process of grouping objects into different groups, such that the common properties of data in each subset is high, and between different subsets is low. The data stream clustering problem is defined as *to maintain a continuously consistent good clustering of the sequence observed so far, using a small amount of memory and time.* The issues are imposed by the continuous arriving data points, and the need to analyze them in real time. These characteristics requires incremental clustering, maintaining cluster structures that evolve over time. Moreover, the data stream may evolve over time, and new clusters might appear, other disappears, reflecting the dynamics of the stream.

Main Techniques

Clustering data streams requires a process able to continuously cluster objects within memory and time restrictions (Gama, 2010). Following de Andrade Silva et al. (2013), algorithms for clustering data streams should ideally fulfil the following requirements: (i) provide timely results by performing fast and incremental processing of data objects; (ii) rapidly adapt to changing dynamics of the data, which means algorithms should detect when new clusters may appear, or others disappear; (iii) scale to the number of objects that are continuously arriving; (iv) provide a model representation that is not only compact, but that also does not grow with the number of objects processed (notice that even a linear growth should not be tolerated); (v) rapidly detect the presence of outliers and act accordingly; and (vi) deal with different data types, *e.g.*, XML trees, DNA sequences, GPS temporal and spatial information. Although these requirements are only partially fulfilled in practice, it is instructive to keep them in mind when designing algorithms for clustering data streams.

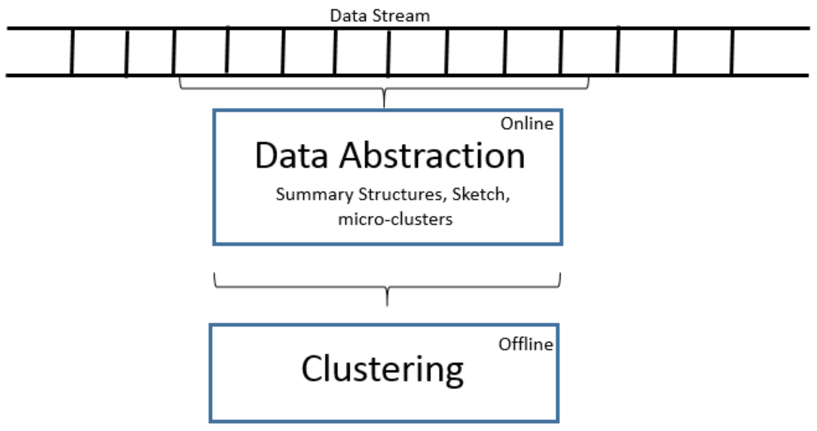

Figure 1: A generic schema for clustering data streams.

Major clustering approaches in data stream cluster analysis include:

- *Partitioning* algorithms: construct a partition of a set of objects into k clusters, that minimize some objective function (e.g. the sum of squares distances to the centroid representative). Examples include k-means (Farnstrom et al., 2000), and k-medoids (Guha et al., 2003);

- *Micro-clustering* algorithms: divide the clustering process into two phases, where the first phase is online and summarizes the data stream in local models (micro-clusters) and the second phase generates a global cluster model from the micro-clusters. Examples of these algorithms include BIRCH (Zhang et al., 1996), CluStream (Aggarwal et al., 2003), and Clustree (Kranen et al., 2011).

Basic Concepts

Data stream clustering algorithms can be summarized into two main steps: data summarization step and clustering step, as illustrated in Figure 1. The online abstraction step summarizes the data stream with the help of particular data structures in order to deal with space and memory constraints of stream applications. These data structures summarize the stream in order to preserve the meaning of the original objects without the need of storing them. Among the commonly-employed data structures, we highlight the feature vectors (Zhang et al., 1996; Aggarwal et al., 2003), prototype arrays (Guha et al., 2003), coreset trees (Ackermann et al., 2012), and data grids (Gama et al., 2011).

A powerful idea in clustering from data streams is the concept of *cluster feature - CF*. A cluster feature, or *micro-cluster*, is a compact representation of a set of points. A CF structure is a triple (N, LS, SS), used to store the sufficient statistics of a set of points:

- N is the number of data points;
- LS is a vector, of the same dimension of data points, that store the linear sum of the N points;
- SS is a vector, of the same dimension of data points, that store the square sum of the N points.

The properties of cluster features are:

- **Incrementality**
  If a point x is added to a cluster A, the sufficient statistics are updated as follows:
$$LS_A \leftarrow LS_A + x; SS_A \leftarrow SS_A + x^2; N_A \leftarrow N_A + 1$$
- **Additivity**
  if A and B are disjoint sets, merging them is equal to the sum of their parts. The additive property allows us to merge sub-clusters incrementally.
$$LS_C \leftarrow LS_A + LS_B; SS_C \leftarrow SS_A + SS_B; N_C \leftarrow N_A + N_B.$$

A CF entry has sufficient information to calculate the norms

$$L_1 = \sum_{i=1}^{n} |LS_{a_i} - LS_{b_i}| \text{ and } L_2 = \sqrt{\sum_{i=1}^{n} (LS_{a_i} - LS_{b_i})^2}$$

and basic measures to characterize a cluster:

- **Centroid**, defined as the gravity center of the cluster:
$$\vec{X}0 = \frac{LS}{N}$$
- **Radius**, defined as the average distance from member points to the centroid:
$$R = \sqrt{\frac{SS}{N} - \frac{LS}{N}^2}$$
  .
- **Diameter**, defined as the largest distance between member points:
$$R = \sqrt{\frac{2N \times SS - 2 \times LS^2}{N \times (N-1)}}$$
  .

When processing and summarizing continuously-arriving stream data, the most recent observations are more important because they reflect the current state of the process generating the data. A popular approach in data stream clustering consists of defining a time window that covers the most recent data. The window models that have been used in the literature are: the landmark model, sliding-window model, and damped model (Gama, 2010).

Partitioning Clustering

K-means is the most widely used clustering algorithm. It constructs a partition of a set of objects into k clusters, that minimize some objective function, usually a squared error function, which imply round-shape clusters. The input parameter k is fixed and must be given in advance that limits its real applicability to streaming and evolving data.

Farnstrom et al. (2000) proposes a *Single pass k-Means* algorithm. The main idea is to use a buffer where points of the dataset are kept in a compressed way. The data stream is processed in blocks. All available space on the buffer is filled with points from the stream. Using these points, find k centres such that the sum of distances from data points to their closest centre is minimized. Only the k-centroids (representing the clustering results) are retained, with the corresponding k-cluster features. In the next iterations, the buffer is initialized with the k-centroids, found in the previous iteration and the incoming data points from the stream. The *Very Fast k-means* algorithm (VFKM) (Domingos e Hulten, 2001) uses the Hoeffding bound to determine the number of examples needed in each step of a k-means algorithm. VFKM runs as a sequence of k-means runs, with increasing number of examples until the Hoeffding bound is satisfied.

Guha et al. (2003) present a analytical study on k-median clustering data streams. The proposed algorithm makes a single pass over the data stream and uses small space. It requires $O(nk)$ time and $O(n\epsilon)$ space where k is the number of centers, n is the number of points and $\epsilon < 1$. They have proved that any k-median algorithm that achieves a constant factor approximation cannot achieve a better run time than $O(nk)$.

Micro Clustering

The idea of dividing the clustering process into two layers, where the first layer generate local models (micro-clusters) and the second layer generates global models from the local ones, is a powerful idea that has been used elsewhere.

The BIRCH system (Zhang et al., 1996) builds a hierarchical structure of data, the CF-tree (see Figure 2), where each node contains a set of cluster features. These CF's contain the sufficient statistics describing a set of points in the data set, and all information of the cluster features below in the tree. The system requires two user defined parameters: b the branch factor or the

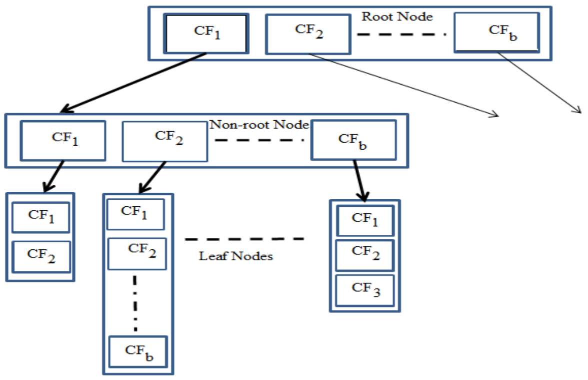

Figure 2: The Clustering Feature Tree in BIRCH. **B** is the maximum number of CFs in a level of the tree.

maximum number of entries in each non-leaf node; and T the maximum diameter (or radius) of any CF in a leaf node. The maximum diameter T defines the examples that can be *absorbed* by a CF. Increasing T, more examples can be absorbed by a micro-cluster and smaller CF-Trees are generated.

When an example is available, it traverses down the current tree from the root, till finding the appropriate leaf. At each non-leaf node, the example follow the *closest*-CF path, with respect to norms L_1 or L_2. If the closest-CF in the leaf cannot absorb the example, make a new CF entry. If there is no room for new leaf, split the parent node. A leaf node might be expanded due to the constrains imposed by B, and T. The process consists of taking the two farthest CFs and creates two new leaf nodes. When traversing backup the CFs are updated.

Monitoring the Evolution of the Cluster Structure

The *CluStream* Algorithm (Aggarwal et al., 2003) is an extension of the BIRCH system designed for data streams. Here, the CFs includes temporal information: the time-stamp of an example is treated as a feature. For each incoming data point, the distance to the centroids of existing CFs, are computed. The data point is absorbed by an existing CF if the distance to the centroid falls within the *maximum boundary* of the CF. The *maximum boundary* is defined as a factor t of the *radius* deviation of the CF; Otherwise, the data point starts a new micro-cluster.

CluStream can generate approximate clusters for any user defined time granularity. This is achieved by storing the CF at regular time intervals, referred to as snapshots. Suppose the user wants to find clusters in the stream based on a history of length h. The off-line component can analyse the snapshots stored at time t, the current time, and $(t - h)$ by using the addictive property of CF. An important problem is when to store the snapshots of the current set of micro-clusters. For example, the natural time frame stores snapshots each

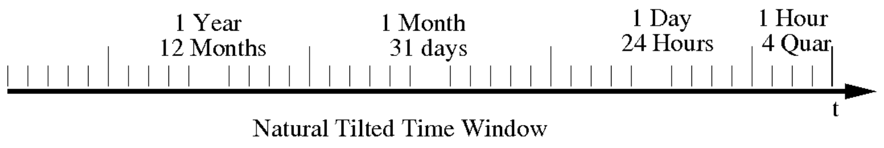

Figure 3: The figure presents a *natural tilted time window*. The most recent data is stored with high-detail, older data is stored in a compressed way. The degree of detail decreases with time.

quarter, 4 quarters are aggregated in hours, 24 hours are aggregated in days, etc (Figure 3). The aggregation level is domain dependent and explores the addictive property of CF. Along similar ideas, Kranen et al. (2011) presents *ClusTree* that uses a weighted CF-vector, which is kept into a hierarchical tree. *ClusTree* provides strategies for dealing with time constraints for anytime clustering, i.e., the possibility of interrupting the process of inserting new objects in the tree at any moment.

Tracking the Evolution of the Cluster Structure

Promising research lines are tracking change in clusters. Spiliopoulou et al. (2006) presents system `MONIC`, for detecting and tracking change in clusters. MONIC assumes that a cluster is an object in a geometric space. It encompasses changes that involve more than one cluster, allowing for insights on cluster change in the whole clustering. The transition tracking mechanism is based on the degree of overlapping between the two clusters. The concept of *overlap* between two clusters X and Y, is defined as the normed number of common records weighted with the age of the records. Assume that cluster X was obtained at time t_1 and cluster Y at time t_2. The degree of overlapping between the two clusters is given by: $overlap(X,Y) = \frac{\sum_{a \in X \cap Y} age(a,t_2)}{\sum_{x \in X} age(x,t_2)}$ The degree of overlapping allows inferring properties of the underlying data stream. Cluster transition at a given time point is a change in a cluster discovered at an earlier timepoint. MONIC consider *internal* and *external* transitions, that reflect the dynamics of the stream. Examples of cluster transitions include: the cluster survives, the cluster is absorbed; a cluster disappears; a new cluster emerges.

References

Ackermann, M. R., Märtens, M., Raupach, C., Swierkot, K., Lammersen, C., e Sohler, C. (2012). Streamkm++: A clustering algorithm for data streams. *ACM Journal of Experimental Algorithmics*, 17(1).

Aggarwal, C. C., Han, J., Wang, J., e Yu, P. S. (2003). A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very Large Data Bases*, pages 81–92. VLDB Endowment.

de Andrade Silva, J., Faria, E. R., Barros, R. C., Hruschka, E. R., de Carvalho, A. C. P. L. F., e Gama, J. (2013). Data stream clustering: A survey. *ACM Comput. Surv.*, 46(1):13.

Domingos, P. e Hulten, G. (2001). A general method for scaling up machine learning algorithms and its application to clustering. In Brodley, C., editor, *Machine Learning, Proceedings of the 18th International Conference*, pages 106–113, Williamstown, USA. Morgan Kaufmann.

Farnstrom, F., Lewis, J., e Elkan, C. (2000). Scalability for clustering algorithms revisited. *SIGKDD Explorations*, 2(1):51–57.

Gama, J. (2010). *KnowledgeDiscovery from Data Streams.* Data Mining and Knowledge Discovery. Chapman & Hall CRC Press, Atlanta, US.

Gama, J., Rodrigues, P. P., e Lopes, L. M. B. (2011). Clustering distributed sensor data streams using local processing and reduced communication. *Intell. Data Anal.*, 15(1):3–28.

Guha, S., Meyerson, A., Mishra, N., Motwani, R., e O'Callaghan, L. (2003). Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528.

Kranen, P., Assent, I., Baldauf, C., e Seidl, T. (2011). The clustree: Indexing micro-clusters for anytime stream mining. *Knowl. Inf. Syst.*, 29(2):249–272.

Spiliopoulou, M., Ntoutsi, I., Theodoridis, Y., e Schult, R. (2006). Monic: modeling and monitoring cluster transitions. In *Proceedings ACM International Conference on Knowledge Discovery and Data Mining*, pages 706–711, Philadelphia, USA. ACM Press.

Zhang, T., Ramakrishnan, R., e Livny, M. (1996). BIRCH: an efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 103–114, Montreal, Canada. ACM Press.