Clustering on Streams

Suresh Venkatasubramanian
University of Utah

DEFINITION

An instance of a clustering problem (see clustering) consists of a collection of points in a distance space, a measure of the *cost* of a clustering, and a measure of the *size* of a clustering. The goal is to compute a partitioning of the points into clusters such that the cost of this clustering is minimized, while the size is kept under some predefined threshold. Less commonly, a threshold for the cost is specified, while the goal is to minimize the size of the clustering.

A data stream (see data streams) is a sequence of data presented to an algorithm one item at a time. A stream algorithm, upon reading an item, must perform some action based on this item and the contents of its working space, which is sublinear in the size of the data sequence. After this action is performed (which might include copying the item to its working space), the item is discarded.

Clustering on streams refers to the problem of clustering a data set presented as a data stream.

HISTORICAL BACKGROUND

Clustering (see clustering) algorithms typically require access to the entire data to produce an effective clustering. This is a problem for large data sets, where random access to the data, or repeated access to the entire data set, is a costly operation. For example, the well-known k-means heuristic is an iterative procedure that in each iteration must read the entire data set twice. One set of approaches to performing clustering on large data involves sampling: a small subset of data is extracted from the input and clustered, and then this clustering is extrapolated to the entire data set.

The *data stream* paradigm[14] came about in two ways: firstly, as a way to model access to large streaming sources (network traffic, satellite imagery) that by virtue of their sheer volume, cannot be archived for offline processing and need to be aggregated, summarized and then discarded in real time. Secondly, the streaming paradigm has shown itself to be the most effective way of accessing large databases: Google's MapReduce[9] computational framework is one example of the efficacy of stream processing.

Designing clustering algorithms for stream data requires different algorithmic ideas than those useful for traditional clustering algorithms. The online computational paradigm[4] is a potential solution: in this paradigm, an algorithm is presented with items one by one, and using only information learned upto the current time, must make a prediction or estimate on the new item being presented. Although the online computing paradigm captures the sequential aspect of stream processing, it does not capture the additional constraint that only a small portion of the history may be stored. In fact, an online algorithm is permitted to use the entirety of the history of the stream, and is usually not limited computationally in any way. Thus, new ideas are needed to perform clustering in a stream setting.

SCIENTIFIC FUNDAMENTALS

Preliminaries

Let X be a domain and d be a distance function defined between pairs of elements in X. Typically, it is assumed that d is a metric (i.e it satisfies the triangle inequality $d(x, y) + d(y, z) \geq d(x, z) \forall x, y, z \in X$). One of the more common measures of the cost of a cluster is the so-called *median cost*: the cost of a cluster $C \subseteq X$ is the function

$$\mathrm{cost}(C) = \sum_{x \in C} d(x, c^*)$$

where $c^* \in X$, the *cluster center*, is the point that minimizes $\mathrm{cost}(C)$. The *k-median problem* is to find a collection of k disjoint clusters, the sum of whose costs is minimized.
An equally important cost function is the *mean cost*: the cost of a cluster $C \subseteq X$ is the function

$$\mathrm{cost}(C) = \sum_{x \in C} d^2(x, c^*)$$

where c^* is defined as before. The *k-means problem* is to find a collection of clusters whose total *mean cost* is minimized. It is useful to note that the median cost is more robust to outliers in the data; however, the mean cost function, especially for points in Euclidean spaces, yields a very simple definition for c^*: it is merely the centroid of the set of points in the cluster. Other measures that are often considered are the k-center cost, where the goal is to minimize the maximum radius of a cluster, and the diameter cost, where the goal is to minimize the maximum diameter of a cluster (note that the diameter measure does not require one to define a cluster center).
A data stream problem consists of a sequence of items $x_1, x_2, \ldots x_n$, and a function $f(x_1, \ldots x_n)$ that one wishes to compute. The limitation here is that the algorithm is only permitted to store a *sublinear* number of items in memory, because n is typically too large for all the items to fit in memory. Further, even random access to the data is prohibitive, and so the algorithm is limited to accessing the data in sequential order.
Since most standard clustering problems (including the ones described above) are NP-hard in general, one cannot expect solutions that minimize the cost of a clustering. However, one can often show that an algorithm comes close to being optimal: formally, one can show that the cost achieved by an algorithm is within some multiplicative factor c of the optimal solution. Such an algorithm is said to be a c-approximation algorithm. Many of the methods presented here will provide such guarantees on the quality of their output. As usual, one should keep in mind that these guarantees are *worst-case*, and thus apply to any possible input the algorithm may encounter. In practice, these algorithms will often perform far better than promised.

General Principles

Stream clustering is a relatively new topic within the larger area of stream algorithms and data analysis. However, there are some general techniques that have proven their usefulness both theoretically as well as practically, and are good starting points for the design and analysis of stream clustering methods. This section reviews these ideas, as well as pointing to examples of how they have been used in various settings.

Incremental Clustering The simplest way to think about a clustering algorithm on stream data is to imagine the stream data arriving in chunks of elements. Prior to the arrival of the current chunk, the clustering algorithm has computed a set of clusters for all the data seen so far. Upon encountering the new chunk, the algorithm must update the clusters, possibly expanding some and contracting others, merging some clusters and splitting others. It then requests the next chunk, discarding the current one. Thus, a core component of any stream clustering algorithm is a routine to incrementally update a clustering when new data arrives. Such an approach was developed by Charikar *et al*[6] for maintaining clusterings of data in a metric space using a diameter cost function. Although their scheme was phrased in terms of incremental clusterings, rather than stream clusterings, their approach generalizes well to streams. They show that their scheme yields a provable approximation to the optimal diameter of a k-clustering.

Representations One of the problems with clustering data streams is choosing a representation for a cluster. At the very least, any stream clustering algorithm stores the location of a cluster center, and possibly the number of items currently associated with this cluster. This representation can be viewed as a *weighted point*, and can be treated as a single point in further iterations of the clustering process. However, this representation loses information about the geometric size and distribution of a cluster. Thus, another standard representation of a cluster consists of the center and the number of points augmented with the sum of squared distances from the points in the cluster to the center. This last term informally measures the variation of points within a cluster, and when viewed in the context of density estimation via Gaussians, is in fact the sample variance of the cluster.

Clusters reduced in this way can be treated as weighted points (or weighted balls), and clustering algorithms should be able to handle such generalized points. One notable example of the use of such a representation is the one-pass clustering algorithm of Bradley *et al*[5], which was simplified and improved by Farnstrom *et al*[11]. Built around the well known k-means algorithm (that iteratively seeks to minimize the k-means measure described above), this technique proceeds as follows.

Algorithm 1 Clustering with representations

```
Initialize cluster centers randomly
while chunk of data remains to be read do
    Read a chunk of data (as much as will fit in memory), and cluster it using the $k$-means algorithm.
    For each cluster, divide the points contained within it into the core (points that are very close to the center
    under various measures), and the periphery.
    Replace the set of points in the core by a summary as described above. Discard all remaining points.
    Use the current cluster list as the set of centers for the next chunk.
```

It is important that representations be *linear*. Specifically, given two chunks of data c, c', and their representations r, r', it should be the case that the representation of $c \cup c'$ be formed from a linear combination of r and r'. This relates to the idea of *sketching* in stream algorithms, and is important because it allows the clustering algorithm to work in the (reduced) space of representations, rather than in the original space of data. Representations like the one described above are linear, and this is a crucial factor in the effectiveness of these algorithms.

Hierarchical Clustering Viewing a cluster as a weighted point in a new clustering problem quickly leads to the idea of *hierarchical clustering*: by thinking of a point as a single-element cluster, and connecting a cluster and its elements in a parent-child relationship, a clustering algorithm can represent multiple levels of merges as a tree of clusters, with the root node being a single cluster containing all the data, and each leaf being a single item. Such a tree is called a Hierarchical Agglomerative Clustering (HAC), since it can be viewed bottom-up as a series of agglomerations. Building such a hierarchy yields more general information about the relationship between clusters, and the ability to make better judgments about how to merge clusters.
The well-known clustering algorithm BIRCH[15] makes use of a hierarchy of cluster representations to cluster a large database in a few passes. In a first pass, a tree called the CF-tree is constructed, where each internal node represents a cluster of clusters, and each leaf represents a cluster of items. This tree is controlled by two parameters: B, the branching factor, and T, a diameter threshold that limits the size of leaf clusters. In further passes, more analysis is performed on the CF-tree to compress clusters further. The tree is built much in the way a B+-tree is built: new items are inserted in the deepest cluster possible, and if the threshold constraint is violated, the cluster is split, and updates are propagated up the tree.
BIRCH is one of the best-known large-data clustering algorithms, and is generally viewed as a benchmark to compare other clustering algorithms against. However, BIRCH does not provide formal guarantees on the quality of the clusterings thus produced. The first algorithm that computes a hierarchical clustering on a stream while providing formal performance guarantees is a method for solving the k-median problem developed by Guha *et al*[12, 13]. This algorithm is best described by first presenting it in a non-streaming context:

Algorithm 2 Small space

```
Divide the input into $\ell$ disjoint parts.
Cluster each part into $O(k)$ clusters. Assign each point to its nearest cluster center.
Cluster the $O(\ell k)$ cluster centers, where each center is weighted by the number of points assigned to it.
```

Note that the total space required by this algorithm is $O(\ell k + n/\ell)$. The value of this algorithm is that it propagates good clusterings: specifically, if the intermediate clusterings are computed by algorithms that yield constant-factor approximations to the best clustering (under the k-median cost measure), then the final output will also be a (larger) constant factor approximation to the best clustering. Also note that the final clustering step may itself be replaced by a recursive call to the algorithm, yielding a hierarchical scheme.
Converting this to a stream algorithm is not too difficult. Consider each chunk of data as one of the disjoint parts the input is broken into. Suppose each part is of size m, and there exists a clustering procedure that can

cluster these points into $2k$ centers with reasonable accuracy. The algorithm reads enough data to obtain m centers ($m^2/2k$ points). These m "points" can be viewed as the input to a second level streaming process, which performs the same operations. In general, the i^{th}-level stream process takes $m^2/2k$ points from the $(i-1)^{\text{th}}$-level stream process and clusters them into m points, which are appended to the stream for the next level.
The guarantees provided by the method rely on having accurate clustering algorithms for the intermediate steps. However, the general paradigm itself is useful as a heuristic: the authors show that using the k-means algorithm as the intermediate clustering step yields reasonable clustering results in practice, even though the method comes with no formal guarantees.

On relaxing the number of clusters If one wishes to obtain guarantees on the quality of a clustering, using at least k clusters is critical; it is easy to design examples where the cost of a $(k-1)$-clustering is much larger than the cost of a k-clustering. One interesting aspect of the above scheme is how it uses weaker clustering algorithms (that output $O(k)$ rather than k clusters) as intermediate steps on the way to computing a k-clustering. In fact, this idea has been shown to be useful in a formal sense: subsequent work by Charikar *et al*[7] showed that if one were to use an extremely weak clustering algorithm (in fact, one that produces $O(k \log n)$ clusters), then this output can be fed into a clustering algorithm that produces k clusters, while maintaining overall quality bounds that are better than those described above. This idea is useful especially if one has a fast algorithm that produces a larger number of clusters, and a more expensive algorithm that produces k clusters: the expensive algorithm can be run on the (small) output of the fast algorithm to produce the desired answer.

Clustering Evolving Data

Stream data is often temporal. Typical data analysis questions are therefore often limited to ranges of time ("in the last three days", "over the past week", "for the period between Jan 1 and Feb 1", and so on). All of the above methods for clustering streams assume that the goal is to cluster the entire data stream, and the only constraint is the space needed to store the data. Although they are almost always incremental, in that the stream can be stopped at any time and the resulting clustering will be accurate *for all data seen upto that point*, they cannot correctly output clusterings on *windows* of data, or allow the influence of past data to gradually wane over time. Even with non-temporal data, it may be important to allow the data analysis to operate on a subset of the data to capture the notion of *concept drift*[10], a term that is used to describe a scenario when natural data characteristics change as the stream evolves.

Sliding Windows A popular model of stream analysis is the *sliding window* model, which introduces a new parameter W. The goal of the stream analysis is to produce summary statistics (a clustering, variance estimates or other statistics), on the *most recent W items only*, while using space that is sublinear in W. This model can be thought of as represented by a *sliding window* of length W with one end (the sliding end) anchored to the current element being read. The challenge of dealing with sliding windows is the problem of deletion. Although not as general as a fully dynamic data model where arbitrary elements can be inserted and deleted, the sliding window model introduces with the problem of updating a cluster representation under deletions, and requires new ideas. One such idea is the *exponential histogram*, first introduced by Datar *et al*[8] to estimate certain statistical properties of sliding windows on streams, and used by Babcock *et al*[3] to compute an approximate k-median clustering in the sliding window model. The idea here is to maintain a set of buckets that together partition all data in the current window. For each bucket, relevant summary statistics are maintained. Intuitively, the smaller the number of items assigned to a bucket, the more accurate the summary statistics (in the limit, the trivial histogram has one bucket for each of the W items in the window). The larger this number, the fewer the number of buckets needed. Balancing these two conflicting requirements yields a scheme where each bucket stores the items between two timestamps, and the bucket sizes increase exponentially as they store items further in the past. It requires more detailed analysis to demonstrate that such a scheme will provide accurate answers to queries over windows, but the use of such exponentially increasing bucket sizes allows the algorithm to use a few buckets, while still maintaining a reasonable approximation to the desired estimate.

Hierarchies of Windows The sliding window model introduces an extra parameter W whose value must be justified by external considerations. One way of getting around this problem is to maintain statistics for multiple values of W (typically an exponentially increasing family). Another approach, used by Aggarwal *et al*[1] is to maintain *snapshots* (summary representations of the clusterings) at time steps at different levels of resolution. For example, a simple two level snapshot scheme might store the cluster representations computed after times $t, t+1, \ldots t+W$, as well as $t, t+2, t+4, \ldots t+2W$ (eliminating duplicate summaries as necessary). Using the linear structure of representations will allow the algorithm to extract summaries for time intervals: they show that such a scheme uses space efficiently while still being able to detect evolution in data streams at different scales.

Decaying Data For scenarios where such a justification might be elusive, another model of evolving data is the *decay model*, in which one can think of a data item's influence waning (typically exponentially) with time. In other words, the value of the i^{th} item, instead of being fixed at x_i, is a function of time $x_i(t) = x_i(0)\exp(-c(t-i))$. This reduces the problem to the standard setting of computing statistics over the entire stream, while using the decay function to decide which items to remove from the limited local storage when computing statistics. The use of exponentially decaying data is quite common in temporal data analysis: one specific example of its application in the clustering of data streams is the work on HPStream by Aggarwal *et al*[2].

KEY APPLICATIONS*

Systems that manage large data sets and perform data analysis will require stream clustering methods. Many modern *data cleaning systems* require such tools, as well as large scientific databases. Another application of stream clustering is for network traffic analysis: such algorithms might be situated at routers, operating on packet streams.

EXPERIMENTAL RESULTS*

Most of the papers cited above are accompanied by experimental evaluations and comparisons to prior work. BIRCH, as mentioned before, is a common benchmarking tool.

CROSS REFERENCE*

- Data Clustering
- Visualization
- Information Retrieval

RECOMMENDED READING

Between 5 and 15 citations to important literature, e.g., in journals, conference proceedings, and websites.

[1] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *VLDB*, pages 81–92, 2003.

[2] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for projected clustering of high dimensional data streams. In *vldb'2004: Proceedings of the Thirtieth international conference on Very large data bases*, pages 852–863. VLDB Endowment, 2004.

[3] Brain Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. Maintaining variance and k-medians over data stream windows. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 234–243, New York, NY, USA, 2003. ACM.

[4] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.

[5] Paul S. Bradley, Usama M. Fayyad, and Cory Reina. Scaling clustering algorithms to large databases. In *KDD*, pages 9–15, 1998.

[6] Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004.

[7] Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 30–39, New York, NY, USA, 2003. ACM.

[8] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows: (extended abstract). In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 635–644, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[9] Jeffrey Dean and Sanjay Ghemaway. MapReduce: Simplified Data Processing on Large Clusters . In *Sixth Symposium on Operating System Design and Implementation (OSDI)*, 2004.

[10] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, New York, NY, USA, 2000. ACM.

[11] Fredrik Farnstrom, James Lewis, and Charles Elkan. Scalability for clustering algorithms revisited. *SIGKDD Explor. Newsl.*, 2(1):51–57, 2000.

[12] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 359, Washington, DC, USA, 2000. IEEE Computer Society.

[13] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, 2003.

[14] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[15] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: A new data clustering algorithm and its applications. *Data Min. Knowl. Discov.*, 1(2):141–182, 1997.