MACHINE LEARNING FOR DATA STREAMS

# CLUSTERING

1. **Efficient and Effective Clustering Methods for Spatial Data Mining (1994):**
   * A cluster is represented by its **medoid**, or the most, centrally located data point in the cluster.
   * The clustering process is formalized as **searching a graph in which each node is a K-partition represented by a set of *K* medoids**, and two nodes are neighbors if the only differ by one medoid.
   * Starts with a randomly selected node. For the current node, it checks at most the *maxneighbor* number of neighbors randomly, and if a better neighbor is found, it moves to the neighbor and continues; otherwise it records the current node as a *local mínimum*, and restarts with a new randomly selected node to search for another local mínimum. It stops after the *numlocal* number of the so-called *local mínima* have been found, and returns the best of these.
   * Suffers from the same drawbacks as the above IO method wrt. efficiency.
   * It may not find a real local minimum due to the searching trimming controlled by *maxneighbor*.
2. **BIRCH: An Efficient Data Clustering Method for Very Large Databases (1996):**
   * Balanced Iterative Reducing and Clustering using Hierarchies.
   * Appropiate for **very large datasets**, by making the time and memory constraints explicit.
   * **Incrementaly and dynamically clusters incoming multidimensional metric data points** to try to produce the best quality clustering with the available resources (i.e., available memory and time constraints).
   * It is **local** (as opposed to global) in that each clustering decision is made without scanning all data points or all currently existing clusters.
   * It exploits the observation that the data space is **usually not uniformly occupied**, and hence **not every data point is equally important for clustering purposes**. A dense región of points is treated collectively as a **single cluster**. Points in sparse regions are treated as **outliers** and removed optionally.
   * It makes full use of available memory to derive the finest posible subclusters (to ensure accuracy) while minimizing I/0 costs (to ensure efficiency).
   * Can typically find a good clustering with a **single scan of the data**, and improve the quality further with a few additional scans.
   * First clustering algorithm proposed in the database area to **handle “noise”** (data points that are not part of the underlying pattern) effectively.
   * Considers **metric** atributes, as in most of the Statistics literature (atributes whose values satisfy the requirements of Euclidean space, i.e., self identity and triangular inequality.
   * Offers opportunities for **parallelism**, and for **interactive** or **dynamic performance tuning** based on knowledge about the dataset, gained over the course of the execution.
   * **The clustering and reducing process is organized and characterized by the use of an in-memory, height-balanced and highly-occupied tree structure.** Due to these features, its running time is **linearly scalable**.
   * **Centroid**, **radius** and **diameter** as properties of a single cluster, and **Euclidean distance**, **Manhattan distance**, **average inter-cluster distance**, **average intra-cluster distance** (diameter of the merged cluster) and **variance increase distance** as properties between two clusters and state them separately.
   * The concepts of **Clustering Feature** and **CF** tree are at the core of *BIRCH*’s incremental clustering. A **Clustering Feature** is a triple summarizing the information that we maintain about a cluster (summarizes a set of points by the sum of the points, the number of points and the sum of the squared lengths of all points).
   * Given the **CF** vectors of clusters, the corresponding distances, as well as the usual quality metrics, can all be calculated easily.
   * **CF** summary is not only **efficient** because it stores much less tan all the data points in the cluster, but also **accurate** because it is sufficient for calculating all the measurements that we need for making clustering decisions in BIRCH.
   * A **CF** tree is a height-balanced tree with two parameters: branching factor *B* and threshold *T*.
   * **BIRCH algorithm**:
     + Phase 1: Scan all the data and build an initial in-memory **CF** tree using the given amount of memory and recycling space on disk (an initial memory summary of the data).
     + Phase 2 (optional): Scan the leaf entries in the initial **CF** tree to rebuild a smaller **CF** tree, while removing more outliers and grouping crowded subclusters into larger ones.
     + Phase 3: A global or semi-global algorithm is used to cluster all leaf entries. They adapt an agglomerative hierarchical clustering algorithm by applying it directly to the subclusters represented by their CF vectors. The undesirable effect of the skewed input order, and splitting triggered by page size causes to be unfaithful to the actual clustering patterns in the data. After this pase, we obtain a set of clusters that captures the major distribution pattern in the data.
     + Phase 4 (optional): Entails the cost of additional passes over the data to correct minor and localized inacuracies because of the rare misplacement problema and refine the clusters further .
     + BIRCH has no theoretical quality guarantees and does indeed sometimes perform badly in practice**.**
   * Comparison with CLARANS (1)
     + CLARANS assumes that the memory is enough for holding the whole dataset, so it needs much more memory than BIRCH does.
     + For all three datasets of the base workload, CLARANS is at least 15 times slower than BIRCH, and is sensitive to the pattern of the dataset.
     + The D (with a hyphen on the top) value for the CLARANS clusters is much larger than that for the BIRCH clusters.
     + In conclusión, for the base workload, BIRCH uses much les memory, but is faster, more accurate, and les order-sensitive compared with CLARANS.
3. **Streaming-Data Algorithms For High-Quality Clustering (2002):**
4. **StreamKM++: A Clustering Algorithms for Data Streams (2010):**
   * Computes very good solutions and is reasonably fast for small *k*.
   * Its running time is still far too large for big datasets, especially for large *k*.
   * Comparison with StreamLS (3):
     + StreamLS is significantly outperformed by Stream-KM++ which computes a coreset and then solves the k-means problem on the coreset by applying k-means++.
     + StreamLS uses a batch approach, and Stream-KM++ uses merge & reduce.
5. **BICO: BIRCH meets Coresets for k-means clustering\* (2013):**
   * A data stream algorithm for the k-means problem that combines the data structure of the SIGMOD Test of Time award winning algorithm **BIRCH** with the theoretical concept of **coresets** for clustering problems.
   * It computes high quality solutions in a time short in practice.
   * This paper contributes to the field of interlacing theoretical and practical work to develop an algorithm good in theory and practice.
   * Comparison with StreamKM++ (4) and StreamLS (3) (approximation algorithms) and comparison with BIRCH (2) and MacQueen’s proposal (very fast heuristics):
     + They achieve the same quality as the approximation algorithms mentioned with a much shorter running time, and they get much better solutions than the heuristics at the cost of only a moderate increase in running time.
     + Approximation algorithms are rather slow in practice. Algorithms fast in practice are usually heuristics and known to compute bad solutions on occasions. The best known one is BIRCH. It also computes a summary of the data, but without theoretical quality guarantee.
     + Stream-KM++ computes a coreset and then solves the k-means problem on the coreset by applying k-means++. It computes very good solutions and is reasonably fast for small k. However, especially for large k, its running time is still far too large for big data sets. BICO also computes a coreset and achieves very good quality in practice, but is significantly faster than Stream-KM++.
     + BIRCH and BICO algorithms compute a summary of the data, but while the summary computed by BIRCH can be arbitrarily bad, they show that BICO computes a coreset S, so for every set of centers C, the cost of the input point set P can be approximated by computing the cost of S.