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*.
   - All of the k-medoid types of approaches, including PAM, CLARA, and CLARANS, are known not to be scalable and thus are not appropriate in a streaming context.
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 .
- When using Clustering Features to store a small summary of points, the quality of the summary decreases when storing points together in one CF that should be assigned to different centers later on. If we summarize points in a CF and later on get centers where all these points are closest to the same center, then their clustering cost can be computed with the CF without any error. Thus, the idea of BIRCH is to heuristically identify points that are likely to be clustered
- together. For this purpose, they use the insertion process defined in the paper.
- BIRCH works with increasing thresholds when processing the input data. It starts with threshold t = 0 and then increases t whenever the number of CFs exceeds a given space bound, calling a rebuilding algorithm to shrink the tree. This algorithm ensures that the number of CFs is decreased sufficiently. Notice that CFs cannot be split again, so the rebuilding might return a different tree than the one computed directly with the new threshold.
- 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.

RESUMEN BIRCH (STREAM): Birch compresses a large dataset into a smaller one via a CFtree (clustering feature tree). Each leaf of this tree captures sufficient statistics (namely the firrst and second moments) of a subset of points. Internal nodes capture sufficient statistics of the leaves below them. The algorithm for computing a CFtree tree repeatedly inserts points into the leaves provided that the radius of the set of points associated with a leaf does not exceed a certain threshold. If the threshold is exceded then a new leaf is created and the tree is appropriately balanced. If the tree does not fit in main memory then a new threshold is used to create a smaller tree.

3. Streaming-Data Algorithms For High-Quality Clustering (2002):

- Most heuristics, including k-Means, are also infeasible for data streams because they require random access. As a result, several heuristics have been proposed for scaling clustering algorithms. In the database literature, the BIRCH system is commonly considered to provide a competitive heuristic for this problem. There are no guarantees on their SSQ performance.
- The k-Means algorithm and BIRCH are most relevant to their results. Most other previous work on clustering either does not offer the

scalability required for a fast streaming algorithm or does not directly optimize SSQ.

- It mentions CLARANS (1) because choosing a new medoid among all the remaining points is time-consuming and, to address this problem, CLARANS draws a fresh sample of feasible centers before each calculation of SSQ improvement.
- They assume that our data stream is not sorted in any way.
- Their solution for k-Median is obtained via a variant called **facility location**, which does not specify in advance the number of clusters desired, and instead evaluates an algorithm's performance by a combination of SSQ and the number of centers used.
- The streaming algorithm given in Section 3 is shown to enjoy theoretical quality guarantees.
- The SSQ minimization problem is identical to k-Median except that **M = R^b for some integer b**, **d is the Euclidean metric**, **the medians can be arbitrary points in M**, and d2(x; ci) replaces d(x; ci) in the objective function; that is, **they minimize the "sum of squares" (SSQ) rather than the sum of distances**.
- **Facility location** is the same as k-Median except that instead of restricting the number of medians to be at most k they simply impose a cost for each median, or facility. The additive cost associated with each facility is called the **facility cost**. It allows as input a rango of number of centers.
- Previous problems are known to be **NP-hard**, and several theoretical approximation algorithms are known. Their algorithm for clustering streaming data uses a subroutine called **LSEARCH**. The streaming algorithm, STREAM, is as follows: They cluster the ith chunk Xi using LSEARCH, and assign each resulting median a weight equal to the sum of the weights of its members from Xi. They then purge memory, retaining only the k weighted cluster centers, and apply LSEARCH to the weighted centers they have retained from X1,…, Xi, to obtain a set of (weighted) centers for the entire stream X1 U…U Xi.
- The CG algorithm does not directly solve k-Median but could be used as a subroutine to a k-Median algorithm, as follows. They first set an initial range for the facility cost z (between 0 and an easy-to-calculate
- upper bound); they then perform a binary search within this range to find a value of z that gives them the desired number k of facilities; for each value of z that they try, we call Algorithm CG to get a solution.

- Time calling CG as a subroutine of a binary search is prohibitive for large data streams. Therefore, they describe a new local search algorithm that relies on the correctness of the above algorithm but avoids the super-quadratic running time by taking advantage o the structure of local search in certain ways.

- Comparison with BIRCH (2) and k-Means:
  - They give empirical evidence that the clustering algorithm outperforms the commonly-used k-Means algorithm.
  - They also experimentally demonstrate our streaming algorithm's superior performance to Birch.
  - We found that SSQ for k-means was worse than that for LSEARCH, and that LSEARCH typically found near-optimum (if not the optimum) solution.
  - Over the course of multiple runs, there was a large variance in the performance of k-Means, whereas LSEARCH was consistently good. LSEARCH took longer to run for each trial but for most datasets found a near-optimal answer before k-Means found an equally good solution. On many datasets k-Means never found a good solution.
  - K-means is infeasible for data streams because they require random Access.
  - STREAM and BIRCH have a common method of attack: repeated preclustering of the data. However the preclustering of STREAM is bottom up, where every substep is a clustering process, whereas the preclustering in BIRCH is top down partitioning.
  - Overall, our results point to a cluster quality vs. running time tradeoff. In applications where speed is of the essence, e.g., clustering web search results, BIRCH appears to do a reasonable quick and dirty job. In applications like intrusion detection or target marketing where mistakes can be costly our STREAM algorithm exhibits superior SSQ performance.

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.
- A **coreset** is a small weighted set of points S, that ensures that if we compute the weighted clustering cost of S for any given set of centers C, then the result will be a (1 + e)-approximation of the cost of the original input.
- It computes high quality solutions in a time short in practice.
- First, BICO computes a summary S of the data with a provable quality guarantee: For every center set C, S has the same cost as P up to a (1+")-factor, i. e., S is a coreset. Then, it runs k-means++ on S.
- This paper contributes to the field of interlacing theoretical and practical work to develop an algorithm good in theory and practice.
- The main problem with the insertion procedure of BIRCH is that the decisión whether points are added to a CF or not is based on the increase of the radius of the candidate CF.
- Like BIRCH, BICO uses a tree whose nodes correspond to CFs.
- Algorithm 1.
- The aim of the rebuilding algorithm is to create a tree which is similar to the tree which would have resulted from using the new threshold in the first place.
- Comparison with StreamKM++ (4) and StreamLS (3) (approximation algorithms, designed for high quality solutions) 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.
    - For small k, BICOs running time is only beaten by MacQueen's and in particular, BICO is 5-10 times faster than Stream-KM++ (and more for StreamLS). For larger k, BICO needs to maintain a

larger coreset to keep the quality up. However, BICO can trade quality for speed. They do additional testruns showing that with different parameters, BICO still beats the cost of MacQueen and BIRCH in similar running time. They believe that BICO provides the best quality-speed trade-off in practice.

- BIRCH decides heuristically how to group the points into subclusters. We aim at refining this process such that the reduced set is not only small but is also guaranteed to approximate the original point set. More precisely, we aim at constructing a coreset.
- Stream-KM++ also aims at a trade-off between quality and speed which makes it most relevant for our work. BIRCH is the most relevant practical algorithm.
- BIRCH and MacQueen both tend to compute much worse solutions.
- BICO is still practical for large k despite the large running times when adjusted.