More Clustering

CURE Algorithm
Non-Euclidean Approaches
The CURE Algorithm

◆ Problem with BFR/$k$ -means:
  ◆ Assumes clusters are normally distributed in each dimension.
  ◆ And axes are fixed --- ellipses at an angle are not OK.

◆ CURE:
  ◆ Assumes a Euclidean distance.
  ◆ Allows clusters to assume any shape.
Example: Stanford Faculty Salaries
Starting CURE

1. Pick a random sample of points that fit in main memory.
2. Cluster these points hierarchically --- group nearest points/clusters.
3. For each cluster, pick a sample of points, as dispersed as possible.
4. From the sample, pick representatives by moving them (say) 20% toward the centroid of the cluster.
Example: Initial Clusters

- **salary**
- **age**
Example: Pick Dispersed Points

Pick (say) 4 remote points for each cluster.
Example: Pick Dispersed Points

Move points (say) 20% toward the centroid.
Finishing CURE

- Now, visit each point \( p \) in the data set.
- Place it in the “closest cluster.”
  - Normal definition of “closest”: that cluster with the closest (to \( p \)) among all the sample points of all the clusters.
Curse of Dimensionality

- One way to look at it: in large-dimension spaces, random vectors are perpendicular. Why?
  - Argument #1: Lots of 2-dim subspaces. There must be one where the vectors’ projections are almost perpendicular.
  - Argument #2: Expected value of cosine of angle is 0.
Cosine of Angle Between Random Vectors

◆ Assume vectors emanate from the origin (0,0,...,0).
◆ Components are random in range [-1,1].
◆ \((a_1,a_2,...,a_n)(b_1,b_2,...,b_n)\) has expected value 0 and a standard deviation that grows as \(\sqrt{n}\).
◆ But lengths of both vectors grow as \(\sqrt{n}\).
◆ So dot product around \(\sqrt{n}/ (\sqrt{n} * \sqrt{n}) = 1/\sqrt{n}\).
Random Vectors --- Continued

◆ Thus, a typical pair of vectors has an angle whose cosine is on the order of $1/\sqrt{n}$.

◆ As $n \to \infty$, that’s 0; i.e., the angle is about 90°.
Interesting Consequence

▸ Suppose “random vectors are perpendicular,” even in non-Euclidean spaces.

▸ Suppose we know the distance from $A$ to $B$, say $d(A,B)$, and we also know $d(B,C)$, but we don’t know $d(A,C)$.

▸ Suppose $B$ and $C$ are fairly close, say in the same cluster.

▸ What is $d(A,C)$?
Diagram of Situation

Assuming points lie in a plane:
\[ d(A,B)^2 + d(B,C)^2 = d(A,C)^2 \]
Important Point

Why do we assume AB is perpendicular to AC, and not that either of the other two angles are right-angles?

1. AB and AC are not “random vectors”; they each go to points that are far away from A and close to each other.

2. If AB is longer than AC, then it is angle ACB that is right, but both ACB and ABC are approximately right-angles.
Dealing With a Non-Euclidean Space

◆ Problem: clusters cannot be represented by centroids.
◆ Why? Because the “average” of “points” might not be a point in the space.
◆ Best substitute: the *clustroid* = point in the cluster that minimizes the sum of the squares of distances to the points in the cluster.
Representing Clusters in Non-Euclidean Spaces

- Recall BFR represents a Euclidean cluster by $N$, SUM, and SUMSQ.
- A non-Euclidean cluster is represented by:
  - $N$.
  - The clustroid.
  - Sum of the squares of the distances from clustroid to all points in the cluster.
Example of CoD Use

Problem: in non-Euclidean space, we want to decide whether to merge two clusters.

- Each cluster represented by $N$, clustroid, and “SUMSQ.”
- Also, SUMSQ for each point in the cluster, even if it is not the clustroid.

- Merge if SUMSQ for new cluster is “low.”
Estimating SUMSQ

$p$'s clustroid, $c$

other clustroid, $b$
Suppose $p$ Were the Clustroid of Combined Cluster

- It’s SUMSQ would be the sum of:
  1. Old SUMSQ($p$) [for old cluster containing $p$].
  2. SUMSQ($b$) plus $d(p,b)^2$ times number of points in $b$’s cluster.

- Critical point: vector $p \to b$ assumed perpendicular to vectors from $b$ to all other points in its cluster --- justifies (2).
Combining Clusters --- Continued

◆ We can thus estimate SUMSQ for each point in the combined cluster. Take the point with the least SUMSQ as the clustroid of the new cluster --- provided that SUMSQ is small enough.
The GRGPF Algorithm

- From Ganti et al. --- see reading list.
- Works for non-Euclidean distances.
- Works for massive (disk-resident) data.
- Hierarchical clustering.
- Clusters are grouped into a tree of disk blocks (like a B-tree or R-tree).
Information Retained About a Cluster

1. $N$, clustroid, SUMSQ.
2. The $p$ points closest to the clustroid, and their values of SUMSQ.
3. The $p$ points of the cluster that are furthest away from the clustroid, and their SUMSQ’s.
At Interior Nodes of the Tree

- Interior nodes have samples of the clustroids of the clusters found at descendant leaves of this node.
- Try to keep clusters on one leaf block close, descendants of a level-1 node close, etc.
- Interior part of tree kept in main memory.
Initialization

◆ Take a main-memory sample of points.
◆ Organize them into clusters hierarchically.
◆ Build the initial tree, with level-1 interior nodes representing clusters of clusters, and so on.
◆ All other points are inserted into this tree.
Inserting Points

- Start at the root.
- At each interior node, visit one or more children that have sample clustroids near the inserted point.
- At the leaves, insert the point into the cluster with the nearest clustroid.
Updating Cluster Data

- Suppose we add point $X$ to a cluster.
- Increase count $N$ by 1.
- For each of the $2p + 1$ points $Y$ whose SUMSQ is stored, add $d(X,Y)^2$.
- Estimate SUMSQ for $X$. 
Estimating SUMSQ($X$)

- If $C$ is the clustroid, SUMSQ($X$) is, by the CoD assumption:
  \[ Nd \,(X,C)^2 + \text{SUMSQ}(C) \]
  - Based on assumption that vector from $X$ to $C$ is perpendicular to vectors from $C$ to all the other nodes of the cluster.

- This value may allow $X$ to replace one of the closest or furthest nodes.
Possible Modification to Cluster Data

◆ There may be a new clustroid --- one of the $p$ closest points --- because of the addition of $X$.

◆ Eventually, the clustroid may migrate out of the $p$ closest points, and the entire representation of the cluster needs to be recomputed.
Splitting and Merging Clusters

- Maintain a threshold for the radius of a cluster $\sqrt{\text{SUMSQ}/N}$.
- Split a cluster whose radius is too large.
- Adding clusters may overflow leaf blocks, and require splits of blocks up the tree.
  - Splitting is similar to a B-tree.
  - But try to keep locality of clusters.
Splitting and Merging --- (2)

- The problem case is when we have split so much that the tree no longer fits in main memory.
- Raise the threshold on radius and merge clusters that are sufficiently close.
Merging Clusters

Suppose there are nearby clusters with clustroids $C$ and $D$, and we want to consider merging them.

Assume that the clustroid of the combined cluster will be one of the $\rho$ furthest points from the clustroid of one of those clusters.
Merging --- (2)

◆ Compute SUMSQ(X) [from the cluster of C] for the combined cluster by summing:

1. SUMSQ(X) from its own cluster.
2. SUMSQ(D) + N \[ d(X,C)^2 + d(C,D)^2 \].

◆ Uses the CoD to reason that the distance from X to each point in the other cluster goes to C, makes a right angle to D, and another right angle to the point.
Merging --- Concluded

- Pick as the clustroid for the combined cluster that point with the least SUMSQ.
- But if this SUMSQ is too large, do not merge clusters.
- Hope you get enough mergers to fit the tree in main memory.
Fastmap

- Not a clustering algorithm --- rather, a method for applying *multidimensional scaling*.
  - That is, mapping the points onto a small-dimension space, so the CoD does not apply.
Fastmap --- (2)

- Assumes non-Euclidean space.
  - But like GRGFP pretends it is working in 2-dimensional Euclidean space when it is convenient to do so.

- Goal: map $n$ points in much less than $O(n^2)$ time.
  - I.e., you cannot compute distances between each pair of points and place points in k-dim. space to minimize error.
Fastmap --- Key Idea

• Create a “dimension” in non-Euclidean space by:
  1. Pick a pair of points A and B that are far apart.
     • Start with random A; pick most distant B.
  2. Treat AB as an “axis” and project all points onto AB, using the law of cosines.
Projecting Point C Onto AB

\[ x = \frac{d^2(A,C) + d^2(A,B) - d^2(B,C)}{2d(A,B)} \]
Revising Distances

- Having computed the position of every point along the *pseudo-axis* AB, we need to lower the distances between points in the "other dimensions."
\[ d_{new}(C,D) = \sqrt{d_{old}(C,D)^2 - (x-y)^2} \]
But …

◆ We can’t afford to compute new distances for each pseudo-dimension.
  ✷ It would take $O(n^2)$ time.

◆ Rather, for each pseudo-dimension, store the position along the pseudo-axis for each point, and adjust the distance between points by square-subtract-sqrt only when needed.
  ✷ I.e., one of the points is an axis-end.
Fastmap --- Summary

◆ Pick a number of dimensions $k$.

FOR $i = 1$ TO $k$ DO BEGIN
Pick a pseudo-axis $A_iB_i$;
Compute projection of each point onto this pseudo-axis;
END;

◆ Each step is $O(ni)$; total $O(nk^2)$. 