e-PGPathshala
Subject : Computer Science
Paper: Data Analytics
Module No 27: CS/DA/27 -Clustering - 2
Quadrant 1 – e-text

Learning Objectives

- Understand different distance metrics
- Learn the partitioning method with k-means
- Extensions of k-means-BFR algorithm

1.1 Introduction to Distance metrics

Each clustering problem is based on some kind of "distance" between points. Two major classes of distance measure are Euclidean distance and Non-Euclidean distance. A Euclidean space has some number of real-valued dimensions and "dense" points. There is a notion of "average" of two points. A Euclidean distance is based on the locations of points in such a space. Example: Manhattan distance. A Non-Euclidean distance is based on properties of points, but not their "location" in a space. Example: Jaccard distance, Cosine distance, etc. in this module let us discuss the non-Euclidean case.

1.2 Non-Euclidean Case

Non-Euclidean metrics based clustering algorithms have the benefit of not confining the space shape of data set. Non-Euclidean metrics such as kernel metrics have been introduced into some clustering algorithms, such as k-means. Kernel-based method transforms the data patterns from input space to a feature space by a kernel function. In this way, nonlinearly patterns in the input space can be transformed into linearly patterns in the feature space, without the curse of

dimensionality. Mahalanobis distance is relative with correlations between variables, whereas, Euclidean distance treats them equally.

1.3 Non-Euclidean case?

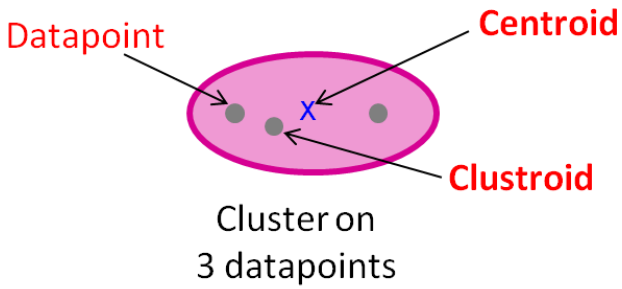
- The only "locations" we can talk about are the points themselves
  - i.e., there is no "average" of two points

Approach 1:

- How to represent a cluster of many points? clustroid = (data) point "closest" to other points
- How do you determine the "nearness" of clusters? Treat clustroid as if it were centroid, when computing inter-cluster distances.

Closest Point

- How to represent a cluster of many points?
  - clustroid = point "closest" to other points
- Possible meanings of "closest":
  - Smallest maximum distance to other points
  - Smallest average distance to other points
  - Smallest sum of squares of distances to other points
  - For distance metric d clustroid c of cluster C is = $\min_c \sum_{x \in C} d(x,c)^2$
- Centroid is the avg. of all (data)points in the cluster. This means centroid is an "artificial" point.
- Clustroid is an existing (data)point that is "closest" to all other points in the cluster.



"Nearness" of Clusters

- **Approach 2:**

  Inter-cluster distance = minimum of the distances between any two points, one from each cluster

- **Approach 3:**

  Pick a notion of "cohesion" of clusters, *e.g.*, maximum distance from the clusters

  - Merge clusters whose union is most cohesive

1.4 Cohesion Metrics

Cohesion metrics measure how well the methods of a class are related to each other. A cohesive class performs one function. A non-cohesive class performs two or more unrelated functions. A non-cohesive class may need to be restructured into two or more smaller classes. The assumption behind the following cohesion metrics is that methods are related if they work on the same class-level variables. Methods are unrelated if they work on different variables altogether. In a cohesive class, methods work with the same set of variables. In a non-cohesive class, there are some methods that work on different data.

- **Diameter** of the merged cluster = maximum distance between points in the cluster
- **Average distance** between points in the cluster

- **Density-based approach -** Take the diameter or avg. distance, e.g., and divide by the number of points in the cluster

Complexity:

- **Naïve implementation of hierarchical clustering:**
  - At each step, compute pairwise distances between all pairs of clusters, then merge
  - $O(N^3)$
- Careful implementation using priority queue can reduce time to $O(N^2 \log N)$
  - Still too expensive for really big datasets that do not fit in memory

1.5 Partitioning method

Construct a partition of a database D of n objects into a set of k clusters. Given a k, find a partition of k clusters that optimizes the chosen partitioning criterion. Example: k-means - Each cluster is represented by the center of the cluster

K- means Algorithm:

K-means clustering is a data mining/machine learning algorithm used to cluster observations into groups of related observations without any prior knowledge of those relationships.

How is this done:

Consists on the separation of all information observations of a specific dataset into *k* different clusters which aggregate each one of the data entries. For this are defined *k* center points, one for each cluster, called centroids. Each piece of data is connected to the partition with the nearest mean from that point, that is, the nearest centroid. As a result of this operation we'll get *k* sets of data entries, each one related with one specific centroid. Within these sets, the distance to the respective centroid is minimized.

Algorithm

The process to achieve the result sets of classified data is quite simple. It basically consists on several iterations of a specific process, designed to get a optimal minimum solution for all data points.

First, we need to establish a function of what we want to minimize, in our case the distance between every data point and the correspondent centroid.

So, what we want is:

$$J = \sum_{i=1}^{k} \sum_{j=1}^{n} \|x_j - c_i\|^2$$

With this function well defined, we can split the process in several steps, in order to achieve the wanted result. Our starting point is a large set of data entries and a *k* defining the number of centers.

Step 1 –

The first step is to choose randomly *k* of our points as partition centers.

Step 2

Next, we compute the distance between every data point on the set and those centers and store that information.

Step 3

Supported by the last step calculations, we assign each point to the nearest cluster center. This is, we get the minimum distance calculated for each point, and we add that point to the specific partition set.

Step 4

Update de cluster center positions by using the following formula:

$$c_i = \frac{1}{|k_i|} \sum_{x_{j \in k}}^{k} x_j$$

Step 5

If the cluster centers change, repeat the process from 2. Otherwise you have successfully computed the k means clustering algorithm and got the partition's members and centroids.

The achieved result is the minimum configuration for the selected start points. It is possible that this output isn't the optimal minimum of the selected set of data, but instead a local minimum of the function. To mitigate this problem, we can run process more than one time in order to get the optimal solution.

It is important for you to know that there are some variations of the initial center choice method. Depending on the problem you want to solve, some initial processes might benefit your implementations.

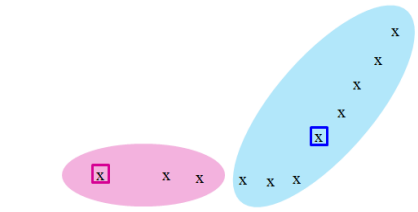
Algorithm with example:

- Assumes Euclidean space/distance
- Start by picking *k*, the number of clusters
- Initialize clusters by picking one point per cluster
    - Example: Pick one point at random, then *k*-1 other points, each as far away as possible from the previous points

Populating Clusters

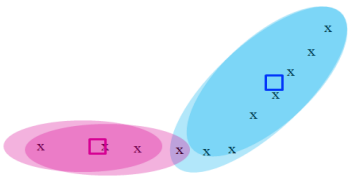
- For each point, place it in the cluster whose current centroid it is nearest
- After all points are assigned, update the locations of centroids of the *k* clusters
- Reassign all points to their closest centroid. Sometimes moves points between clusters
- Repeat 2 and 3 until convergence
    - Convergence: Points don't move between clusters and centroids stabilize

Assigning Clusters



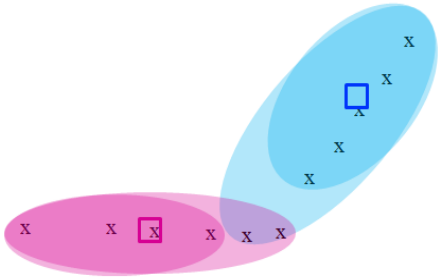
x … data point
… centroid

Clusters after round 1



x … data point
… centroid

Clusters after round 2

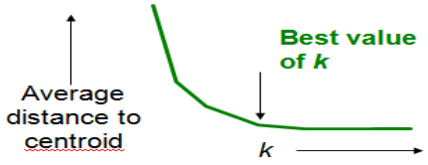


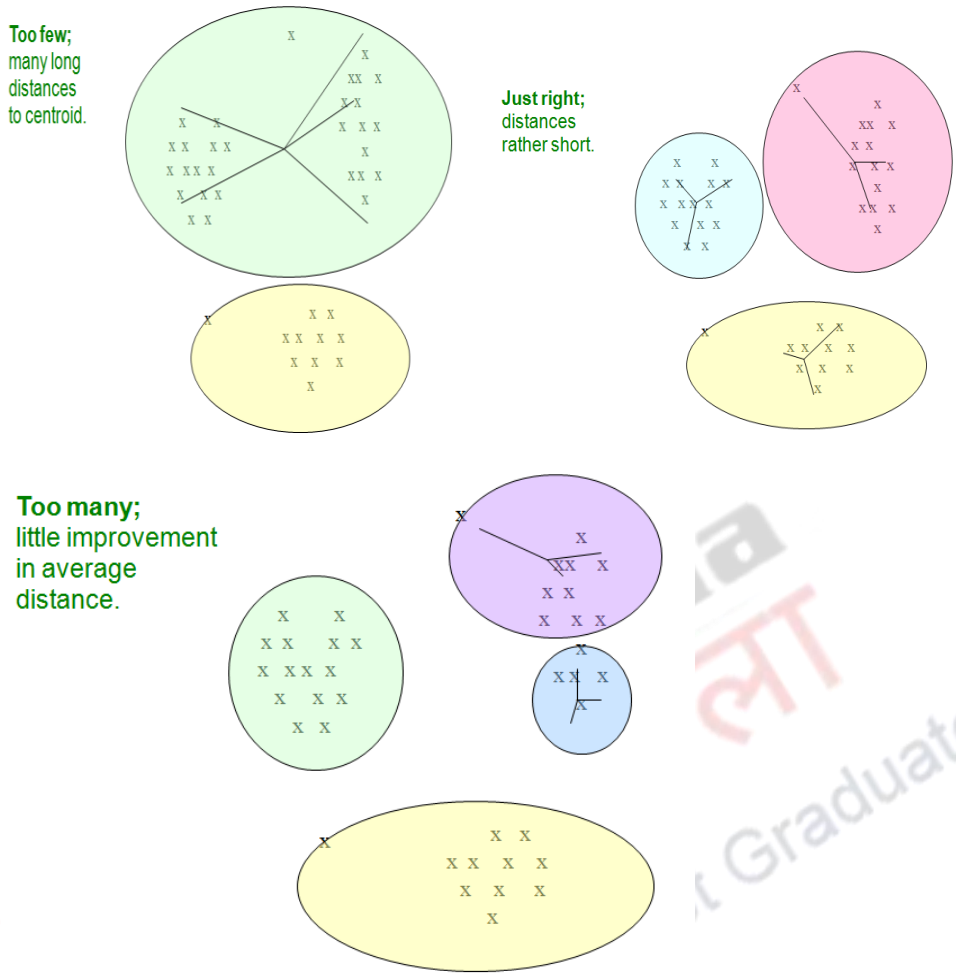
x … data point
… centroid

Clusters at the end

Selecting '*k*'

- Try different **k**, looking at the change in the average distance to centroid as **k** increases
- Average falls rapidly until right **k**, then changes little





1.6 Extensions of *k*-means to large data

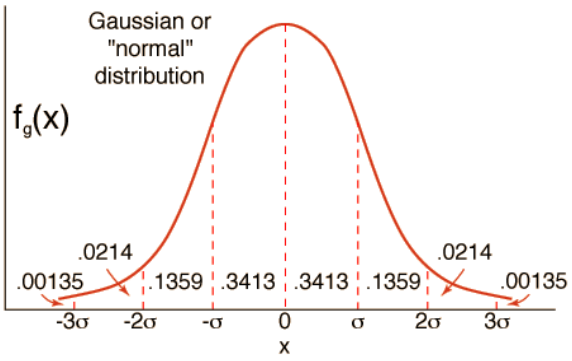
BFR Algorithm

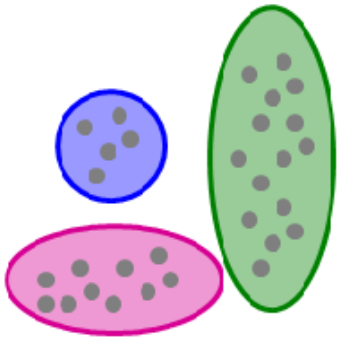
BFR [Bradley-Fayyad-Reina] is a variant of k-means designed for very large (disk resident) datasets.

BFR algorithm is a point assignment clustering algorithm. BFR provides method to scale the clustering algorithm to large databases. It also avoids the multiple scan of the database Designed for high-dimensional spaces. Strong assumption on the clusters shape.

Assumes that clusters arenormally distributed around a centroid in a Euclidean space. Standard deviations in different dimensions may vary clusters are

axis-aligned ellipses. For every point we can quantify the likelihood that it belongs to a particular cluster.





The Process

- Points are read one main-memory-full at a time.
- Most points from previous memory loads are summarized by simple statistics .
- To begin, from the initial load we select the initial k centroids by some sensible approach,

  example:
  - Take k random points.
  - Take a sample; pick a random point, and then k–1 more points, each as far from the previously selected points as possible (works better than taking the k random points)

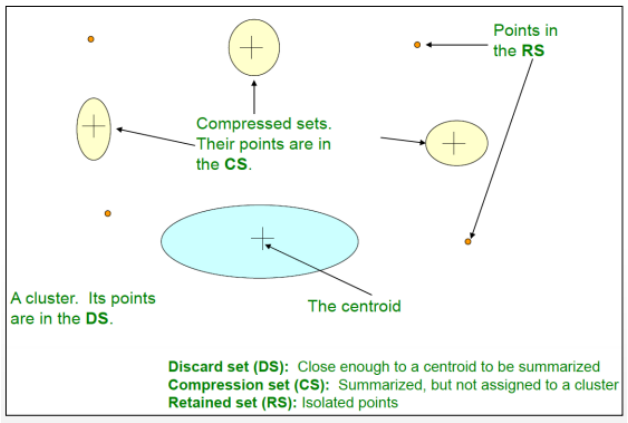
Three Classes of Points

3 sets of points which we keep track of:

Discard set (DS): Points close enough to a centroid to be summarized.

Compression set (CS): Groups of points that are close together but not close to any existing centroid. These points are summarized, but not assigned to a cluster.

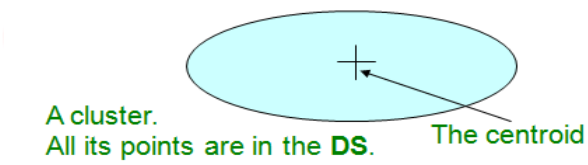
Retained set (RS): Isolated points waiting to be assigned to a compression set.



"Galaxies" Picture

Summarizing Sets of Points

- For each cluster, the discard set (DS) is summarized by:
  - The number of points, N
  - The vector SUM, whose i^{th} component is the sum of the coordinates of the points in the i^{th} dimension
  - The vector SUMSQ:
    - i^{th} component = sum of squares of coordinates in i^{th} dimension



Comments:

- **$2d + 1$** values represent any size cluster

  d = number of dimensions

- Average in **each dimension** (**the centroid**) can be calculated as **SUM_i / N .**

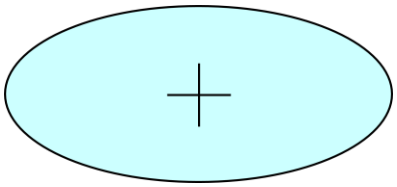
  SUM_i = i^{th} component of SUM

- Variance of a cluster's discard set in dimension *i* is:

  $(SUMSQ_i / N) – (SUM_i / N)^2$

  And standard deviation is the square root of that

- **Next step: Actual clustering**



Note:

Dropping the "axis-aligned" clusters assumption would require storing full covariance matrix to summarize the cluster. So, instead of **SUMSQ** being a ***d***-dim vector, it would be a ***dx d*** matrix, which is too big!

Example:

Suppose a cluster consists of the points $(5,1)$, $(6,-2)$, and $(7,0)$. Then $N = 3$, SUM $= [18,-1]$, and SUMSQ $= [110,5]$. The centroid is SUM$/N$, or $[6,-1/3]$. The variance in the first dimension is $110/3 - (18/3)^2 = 0.667$, so the standard deviation is $\sqrt{0.667} = 0.816$. In the second dimension, the variance is $5/3 - (-1/3)^2 = 1.56$, so the standard deviation is 1.25. □

1.7 The "Memory-Load" of Points

Processing the "Memory-Load" of points (1):

- Find those points that are "**sufficiently close**" to a cluster centroid and add those points to that cluster and the **DS**
  - These points are so close to the centroid that they can be summarized and then discarded
- Use any main-memory clustering algorithm to cluster the remaining points and the old **RS.** Clusters go to the **CS**; outlying points to the **RS.**

Processing the "Memory-Load" of points (2):

- **DS set:** Adjust statistics of the clusters to account for the new points
  - Add ***N***s, ***SUM***s, ***SUMSQ***s
- Consider merging compressed sets in the **CS**
- If this is the last round, merge all compressed sets in the **CS** and all **RS** points into their nearest cluster.

A Few Details:

- Question:
  - How do we decide if a point is "close enough" to a cluster that we will add the point to that cluster?
  - How do we decide whether two compressed sets (CS) deserve to be combined into one?

1.8 Mahalanobis distance:

Mahalanobis distance is the distance between a data point and a multivariate space's centroid (overall mean). Use the Mahalanobis distance in principle components analysis to identify outliers. It is a more powerful multivariate method for detecting outliers than examining one variable at a time because it considers the different scales between variables and the correlations between them.

The Mahalanobis distance has the following properties:

- It accounts for the fact that the variances in each direction are different.
- It accounts for the covariance between variables.
- It reduces to the familiar Euclidean distance for uncorrelated variables with unit variance.

For uni-variate normal data, the uni-variate z-score standardizes the distribution (so that it has mean 0 and unit variance) and gives a dimensionless quantity that specifies the distance from an observation to the mean in terms of the scale of the data. For multivariate normal data with mean μ and covariance matrix Σ, you can decorrelate the variables and standardize the distribution by applying the Cholesky transformation $z = L^{-1}(x - \mu)$, where L is the Cholesky factor of Σ, $\Sigma = LL^T$.After transforming the data, you can compute the standard Euclidian

distance from the point z to the origin. In order to get rid of square roots, I'll compute the square of the Euclidean distance, which is $\text{dist}^2(z,0) = z^T z$. This measures how far from the origin a point is, and it is the multivariate generalization of a z-score.

You can rewrite $z^T z$ in terms of the original correlated variables. The squared distance $\text{Mahal}^2(x,\mu)$ is

- $z^T z$
- $(L^{-1}(x - \mu))^T (L^{-1}(x - \mu))$
- $(x - \mu)^T (LL^T)^{-1} (x - \mu)$
- $(x - \mu)^T \Sigma^{-1} (x - \mu)$

The last formula is the definition of the squared Mahalanobis distance. The derivation uses several matrix identities such as $(AB)^T = B^T A^T$, $(AB)^{-1} = B^{-1}A^{-1}$, and $(A^{-1})^T = (A^T)^{-1}$. Notice that if Σ is the identity matrix, then the Mahalanobis distance reduces to the standard Euclidean distance between x and μ.

The Mahalanobis distance accounts for the variance of each variable and the covariance between variables. Geometrically, it does this by transforming the data into standardized uncorrelated data and computing the ordinary Euclidean distance for the transformed data. In this way, the Mahalanobis distance is like a uni-variate z-score: it provides a way to measure distances that takes into account the scale of the data.

We need a way to decide whether to put a new point into a cluster (and discard)

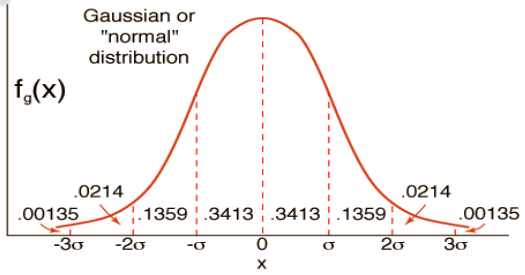
- BFR suggests two ways:
  - The Mahalanobis distance is less than a threshold
  - High likelihood of the point belonging to currently nearest centroid

- Normalized Euclidean distance from centroid
- For point $(x_1, \ldots x_d)$ and centroid $(c_1, \ldots c_d)$
  - Normalize in each dimension : $y_i=(x_i-c_i)/\sigma_i$
  - Take some of the squares of the y_i
  - Take the square root

$$d(x,c)=\sqrt{\sum_{i-1}^{d}((x_i - c_i)/\sigma i)^2}$$

- σ_i - standard deviation of points in the cluster in the i^{th} dimension.

- If clusters are normally distributed in d dimensions, then after transforation, one stand deviation = $\sqrt{d}$
- i.e., 68% of the pointsof the cluster will have a Mahalanobis distance < $\sqrt{d}$
- Accept a point for a cluster if its M.D is < some threshold , e.g. 2 standard deviations



Conditions for Combining Clusters

- Compute the variance of the combined sub-cluster
  - ***N***, ***SUM***, and ***SUMSQ*** allow us to make that calculation quickly
  - Combine if the combined variance is below some threshold

Summary

- Introduction to distance metrics with non-Euclidean distance

- k-means the most popular and commonly used clustering algorithm
- Various extensions of k-means for handling very large data, e.g., BFR algorithm