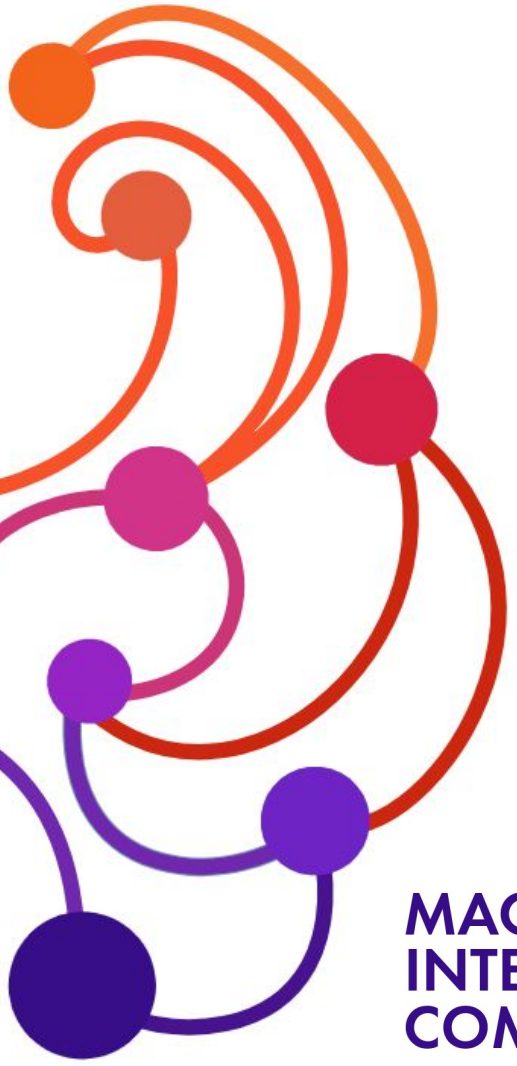


ML Series (Part 2)

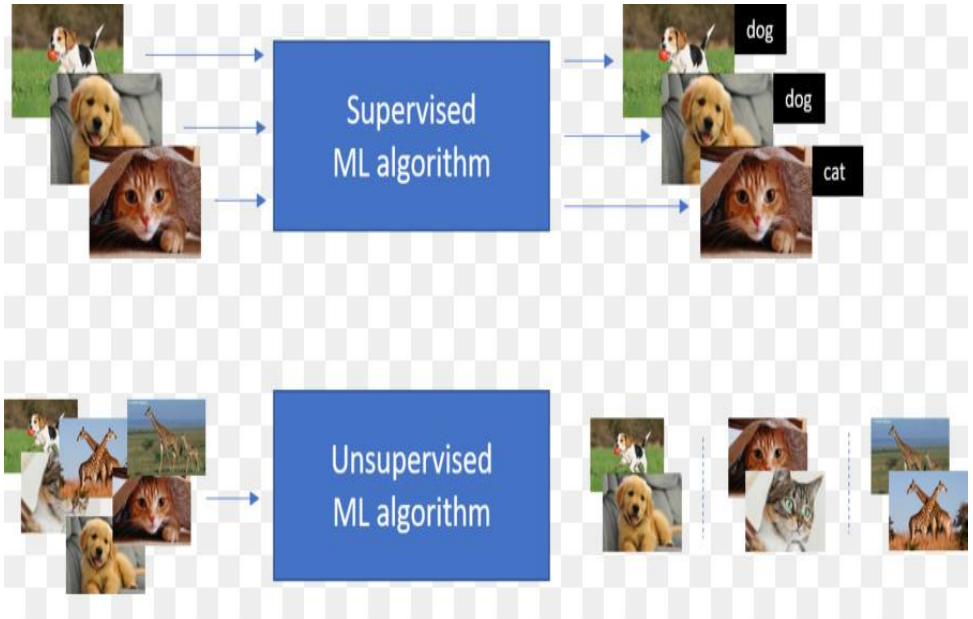
Introduction to Clustering Algorithms

Sign in here: bit.ly/bumicspring2019ws04

MACHINE
INTELLIGENCE
COMMUNITY

What are Clustering Algorithms?

- Clustering algorithms are a form of unsupervised learning.
- What is unsupervised learning?

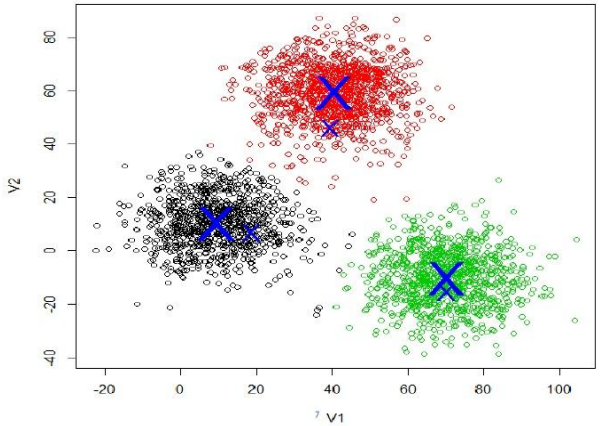


Applications of Clustering

- Search Engines
  - Grouping together similar objects to improve search experiences
- Monitoring Academic Progress
  - Partitioning students into groups based on academic performance
- Biology
  - Classifying plants and animals given features
- Marketing
  - Finding groups of customers with similar behavior given a large database of customer data containing their properties and past buying records
- etc

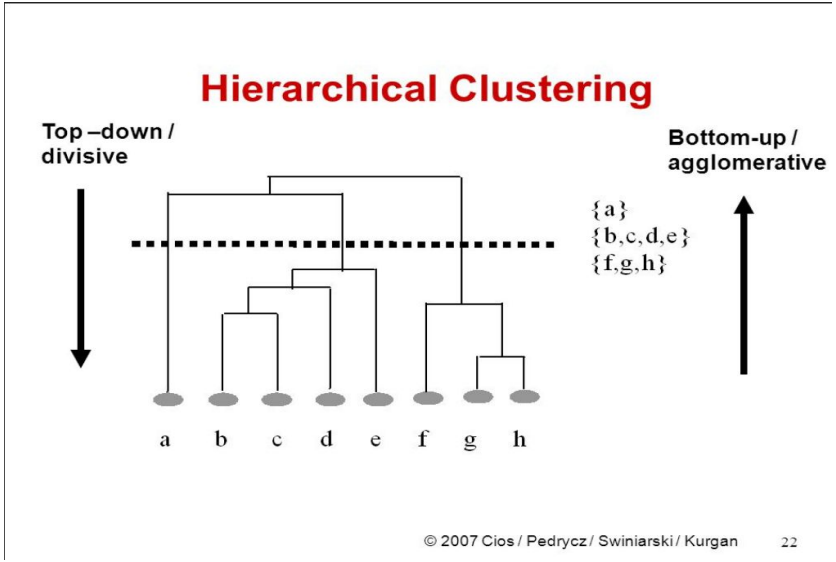
Types of Clustering Algorithms

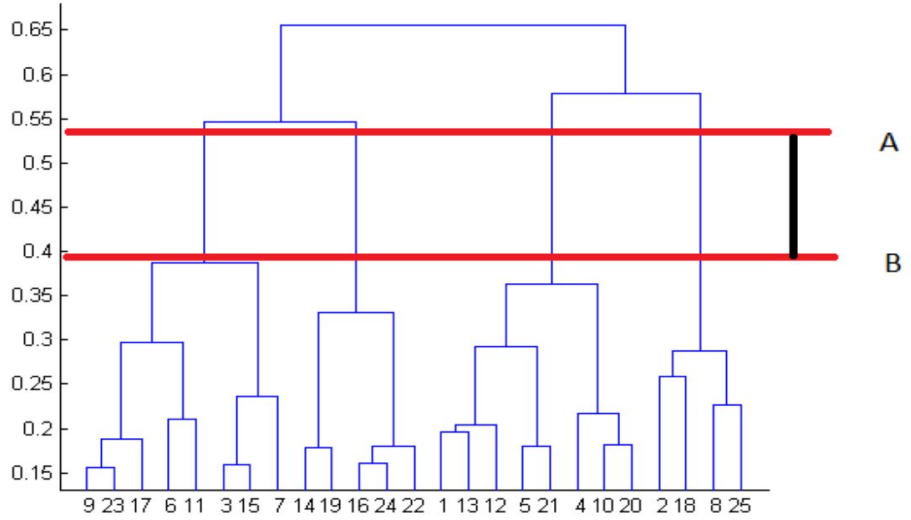
- Hard Clustering
  - Each point belongs either belongs completely to a cluster or it does not.
- Soft Clustering
  - Points can be “shared” by clusters. For example a point will have a probabilistic likelihood of belonging to several various clusters
- Partitional Clustering
  - Carving the space into parts
- Hierarchical Clustering
  - Clusters themselves are further clusterable!!!



Hierarchical Clustering

Agglomerative Clustering (Bottom Up) vs Divisive Clustering (Top Down)

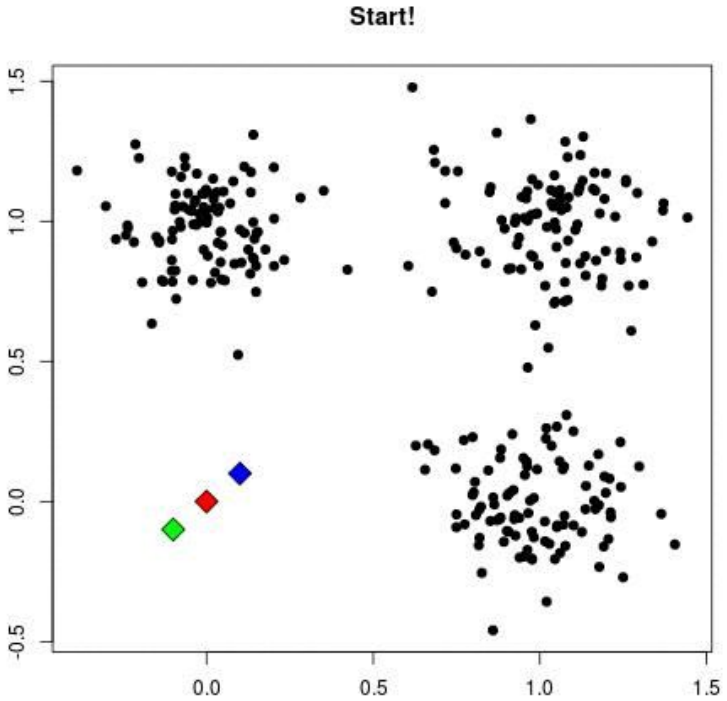




More information at [5]

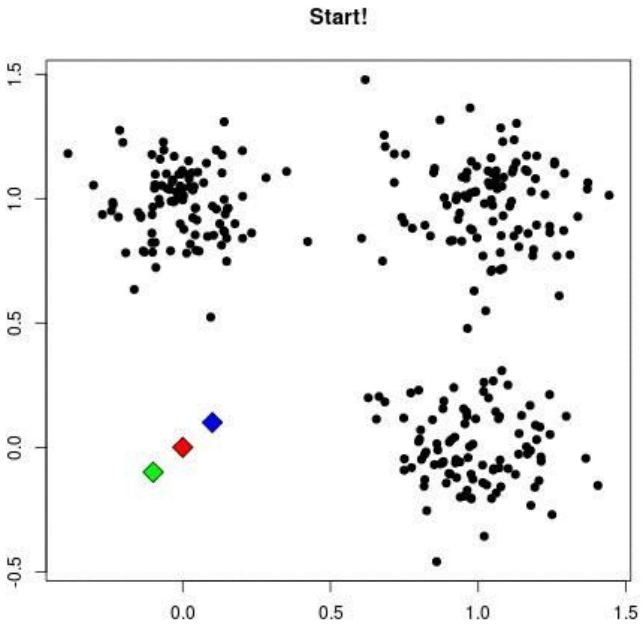
K-Means Clustering

K-means Clustering



K-means Clustering

1. Randomly assign K centroids to the space, which define the initial clusters.

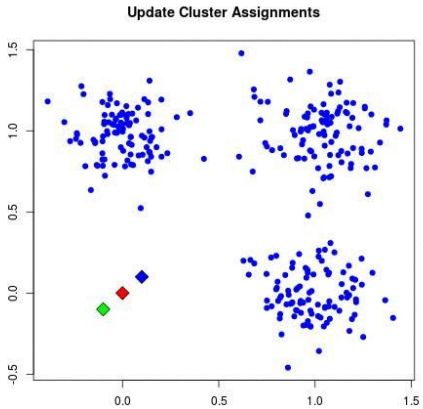


K-means Clustering

2. **Assignment step**: assign each point of training data to the closest centroid,

$$\underset{c_i \in C}{\operatorname{argmin}}\, dist(c_i, x)^2$$

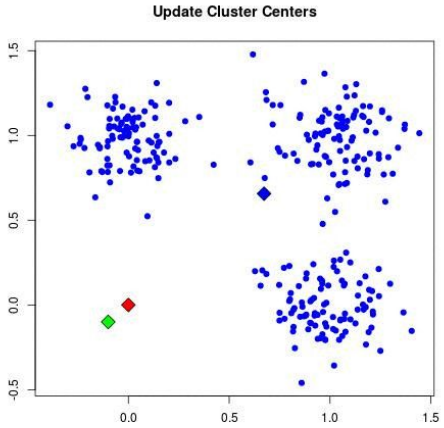
where c_i is the current centroid, and C is the set of all centroids. Usually Euclidean (L2) distance is used as the distance function.



K-means Clustering

3. **Update step:** Find the new centroids by taking the average of all the points assigned to that cluster,

$$c_i = \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i$$

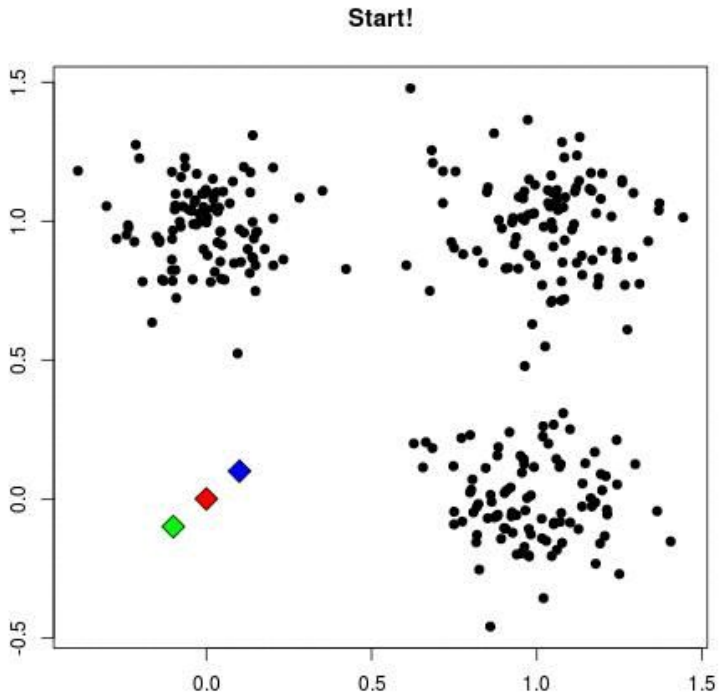


K-means Clustering: Repeat until convergence

number of clusters
number of cases
centroid for cluster *j*
case *i*

objective function $\leftarrow J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2$

Distance function



K-means Clustering: Algorithm

1. Initialize **cluster centroids** $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n$ randomly.
2. Repeat until convergence: {

   For every i, set

   $$c^{(i)} := \arg\min_j \|x^{(i)} - \mu_j\|^2.$$

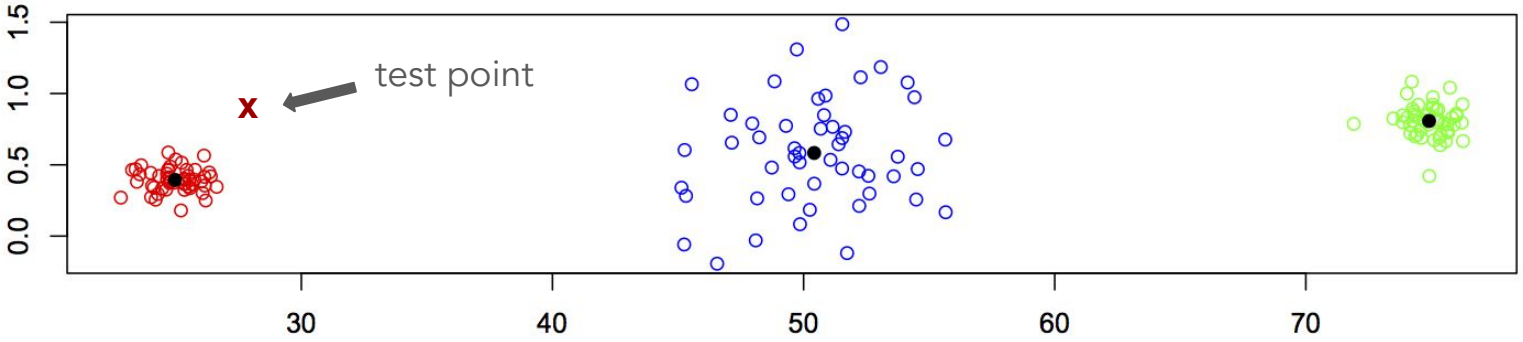
   For each j, set

   $$\mu_j := \frac{\sum_{i=1}^{m} 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)} = j\}}.$$

   }

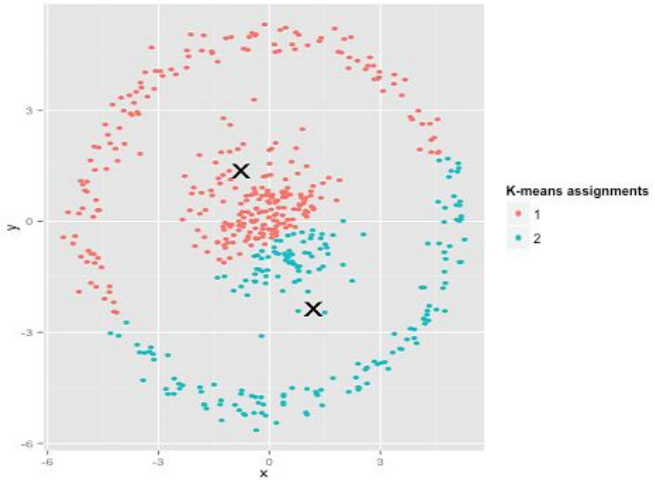
K-means Clustering: Prediction

1. Calculate the Euclidean distance (or whichever distance metric used) of the test points to each of the centroids.
2. Select the cluster corresponding to the shortest distance. This is the predicted cluster.



Characteristics of K-means

- Fixed number of clusters
- Often produces clusters of uniform size
- **Sensitive**: Order of data traversal affects final results, inconsistent due to random initialization
- Fast, easy to implement



DP-means Clustering

DP-means Clustering

1. Initialize only 1 cluster (k=1, $\ell_c = \{x_1 \dots x_n\}$ initially), where the value of the centroid is the global mean,

$$\boldsymbol{\mu_c} = \frac{1}{|\ell_c|} \sum_{\boldsymbol{x} \in \ell_c} \boldsymbol{x}.$$

2. **Assign** each point x_i to the closest centroid. Usually Euclidean (L2) distance is used as the distance function.
3. **Update**: If the distance from x_i > λ (cluster penalty parameter), add a new cluster (k=k+1), and set $\mu_c = x_i$. Otherwise, place the point into the nearest cluster.
4. Find the new centroid by taking the average of all points assigned to each cluster ℓ_j:

$$\boldsymbol{\mu_j} = \frac{1}{|\ell_j|} \sum_{\boldsymbol{x} \in \ell_j} \boldsymbol{x}.$$

5. Repeat steps 2-4 until clusters become stable (they no longer reassign with each iteration).

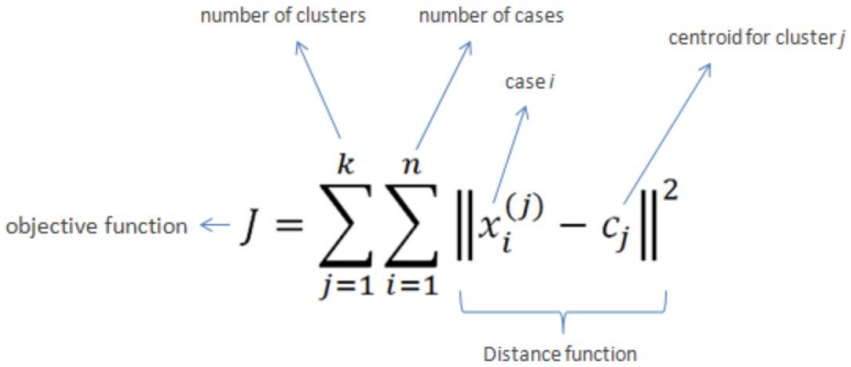
DP-means Clustering: Algorithm

Input: $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$: input data, λ : cluster penalty parameter
Output: Clustering $\ell_1, ..., \ell_k$ and number of clusters k

1. Init. $k = 1, \ell_1 = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ and $\boldsymbol{\mu}_1$ the global mean.
2. Init. cluster indicators $z_i = 1$ for all $i = 1, ..., n$.
3. Repeat until convergence

   - For each point $\boldsymbol{x}_i$
     - Compute $d_{ic} = \|\boldsymbol{x}_i - \boldsymbol{\mu}_c\|^2$ for $c = 1, ..., k$
     - If $\min_c d_{ic} > \lambda$, set $k = k + 1$, $z_i = k$, and $\boldsymbol{\mu}_k = \boldsymbol{x}_i$.
     - Otherwise, set $z_i = \text{argmin}_c d_{ic}$.
   - Generate clusters $\ell_1, ..., \ell_k$ based on $z_1, ..., z_k$: $\ell_j = \{\boldsymbol{x}_i \mid z_i = j\}$.
   - For each cluster ℓ_j, compute $\boldsymbol{\mu}_j = \frac{1}{|\ell_j|} \sum_{\boldsymbol{x} \in \ell_j} \boldsymbol{x}$.

K-means vs. DP-means objectives



$$\min_{\{\ell_c\}_{c=1}^{k}} \quad \sum_{c=1}^{k} \sum_{\boldsymbol{x} \in \ell_c} \|\boldsymbol{x} - \boldsymbol{\mu}_c\|^2 + \boxed{\lambda k}$$

$$\text{where} \qquad \boldsymbol{\mu}_c = \frac{1}{|\ell_c|} \sum_{\boldsymbol{x} \in \ell_c} \boldsymbol{x}.$$

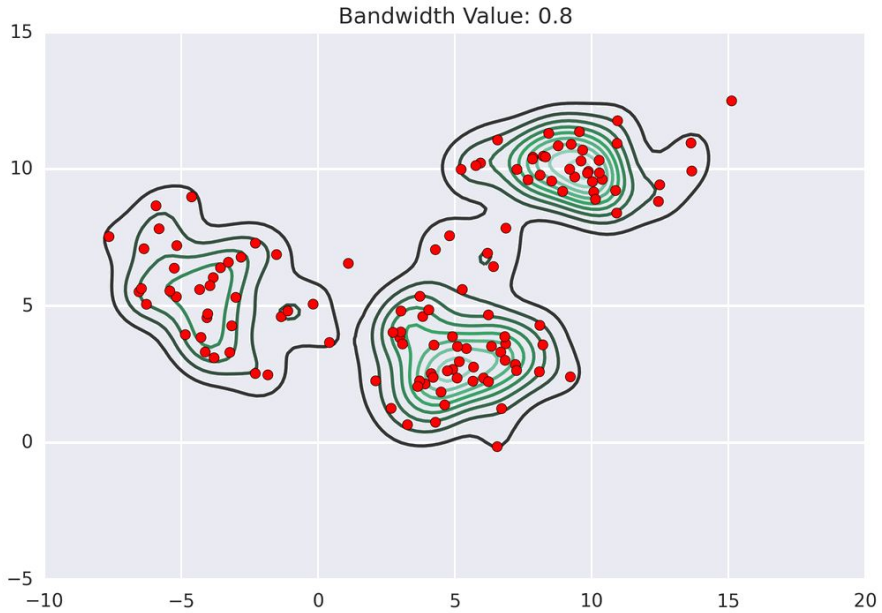
Characteristics of DP-means

- Number of clusters is not fixed
- Prediction is the same as k-means
- Seems like a simple addition to k-means, but actually comes from an intensive derivation from Bayesian nonparametrics (Gaussian mixture models and Dirichlet process) [2]
- Can minimize the number of clusters due to the cluster penalty

Mean Shift

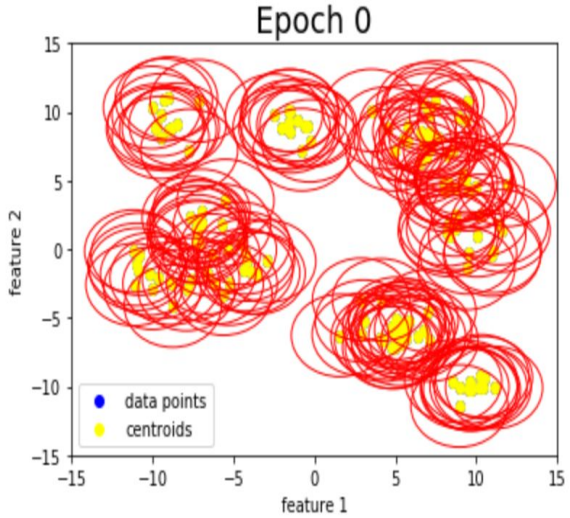
Mean Shift

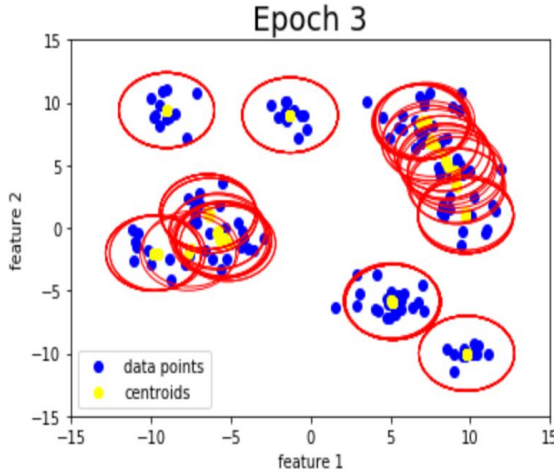
1. Initialize centroids (aka kernels) at every training point
2. Initialize **kernel bandwidth** as a hyperparameter
3. For every iteration, every centroid is repositioned to the mean of all training data points within its kernel bandwidth.
    a. This causes centroids to move towards areas of higher density; thus, kernel density estimation can be used to estimate probability density functions based on the final centroids.

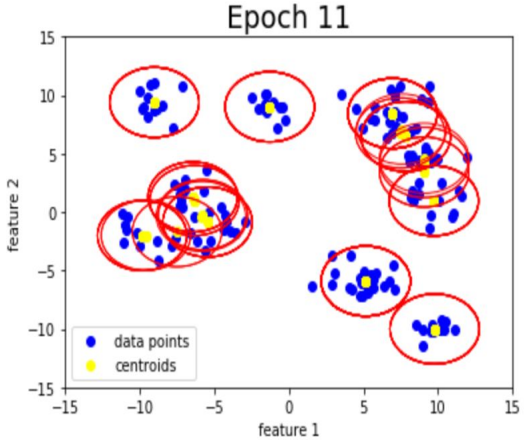


Characteristics of Mean Shift

- Centroids converge to local high-density areas within their kernel bandwidth
- Kernel bandwidth is often chosen using some preexisting understanding of your data.
    - For example, pixels







Mean Shift

Pseudo Code

- Find_neighbourhood_points gets points within centroids' bandwidth

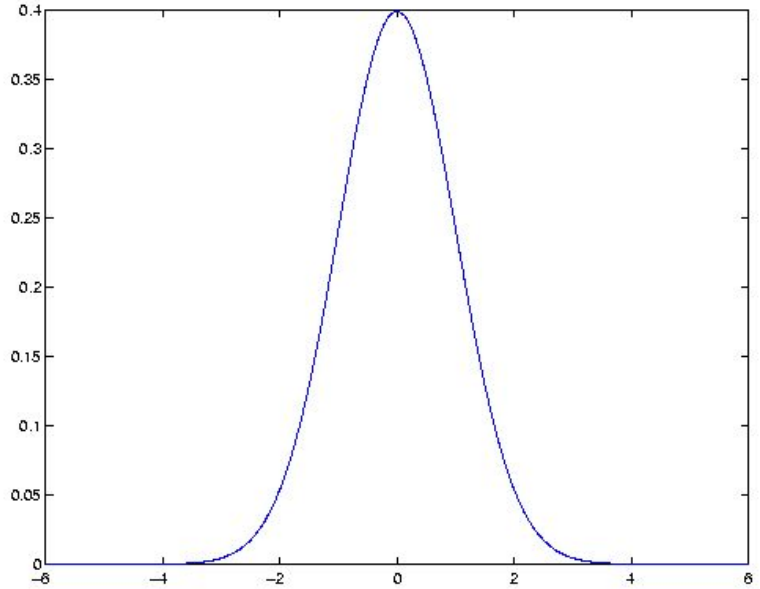
```
def update_centroid(points, bandwidth, centroid):
    neighbourhood_points = find_neighbourhood_points(points, centroid, bandwidth)
    numerator = 0
    denominator = 0
    for point in neighbourhood_points:
        numerator += point
        denominator += 1
    centroid = numerator/denominator
    return centroid
```

Mean Shift

A more powerful implementation of mean shift uses gaussian kernels.

- Centroids = Kernels
- K(distance) = Kernel weight
- m(x) = summation of weights * points divided by summation of weights

$$m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x) x_i}{\sum_{x_i \in N(x)} K(x_i - x)}$$



Mean Shift

- The most popular is the **Gaussian kernel**.

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u^2}{2}\right)$$

Pseudo Code

```
def gaussian_kernel(distance, bandwidth):
    # Fancy kernel function :D
    val = (1/(bandwidth*math.sqrt(2*math.pi))) * np.exp(-0.5*((distance / bandwidth))**2)
    return val

def euclidean_distance(x1, x2):
    return np.linalg.norm(x1-x2)

def naive_update_centroid(points, bandwidth, centroid):
    neighbourhood_points = find_neighbourhood_points(points, centroid, bandwidth)
    numerator = 0
    denominator = 0

    for point in neighbourhood_points:
        distance = euclidean_distance(point, centroid)
        weight = kernel(distance, bandwidth)
        numerator += (point * weight)
        denominator += weight # 1 * weight
    centroid = numerator/denominator
    return centroid
```

Mean Shift

We can use our kernels (otherwise known as centroids) to create soft clustering labels through kernel density estimation.

Spectral Clustering

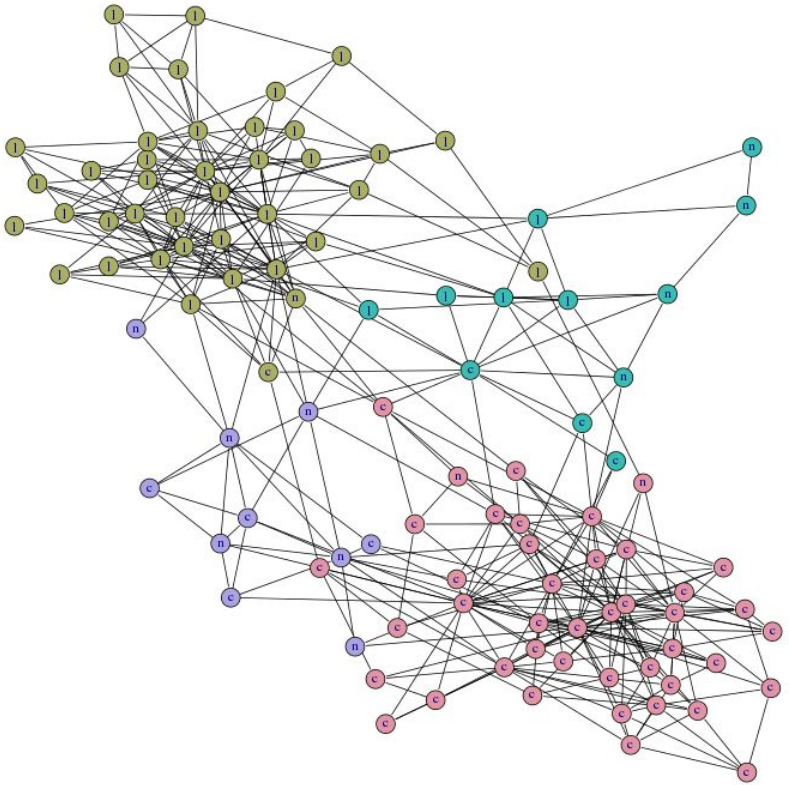
Spectral Clustering

- The goal of hard clustering is to create a cluster indicator matrix, where we assign each point to a cluster via minimization of the objective, i.e.

$$c_1\ [0, 1, 1, 0, 0, 0, 0, 1]$$
$$c_2\ [1, 0, 0, 1, 0, 0, 0, 0]$$
$$c_3\ [0, 0, 0, 0, 1, 1, 1, 0]$$

- This is an NP-hard problem, which is why we compute it iteratively with the goal of local convergence [3].
- It is proven that the k-means objective function can be relaxed to a problem where the globally optimal solution can be computed via eigenvectors of a graph matrix [1].

Spectral Clustering: Goal

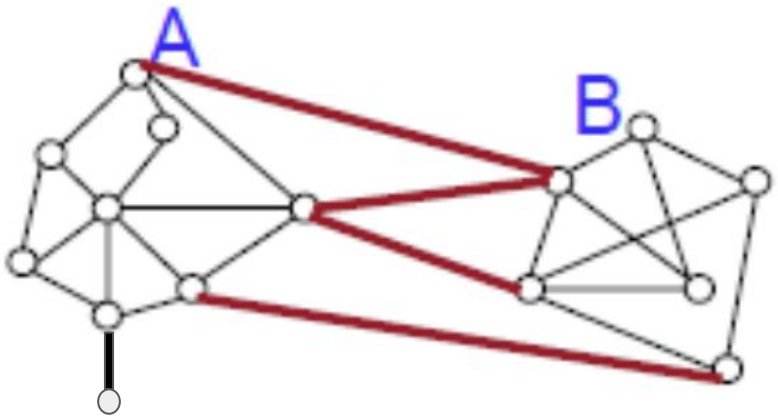


Spectral Clustering: Goal

$$\text{cut}(A, B) := \sum_{i \in A, j \in B} w_{ij}$$

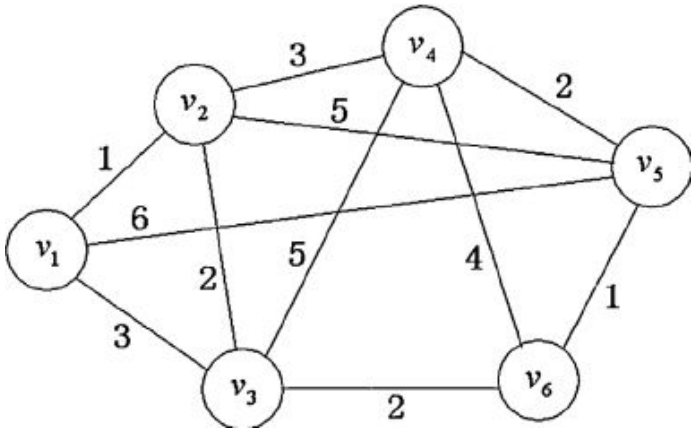
$$\text{Ncut}(A, B) := \text{cut}(A, B)\left(\frac{1}{\text{vol}(A)} + \frac{1}{\text{vol}(B)}\right)$$

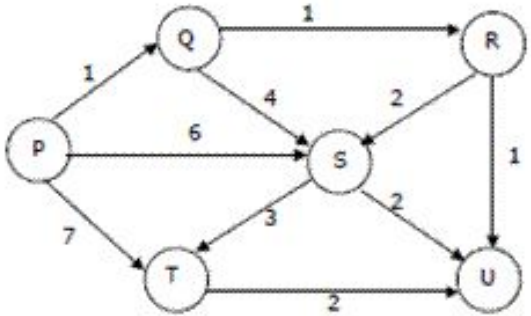
$$\text{vol}(A) = \sum_{i \in A} d_i$$



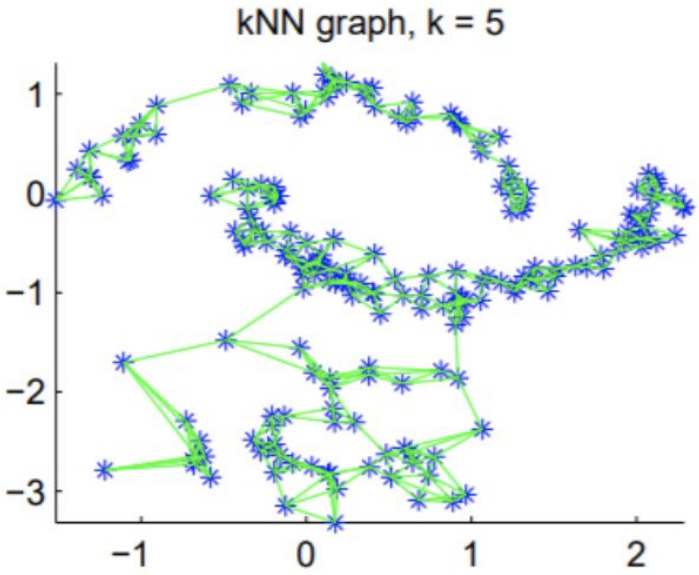
Spectral Clustering: Graph Terminology

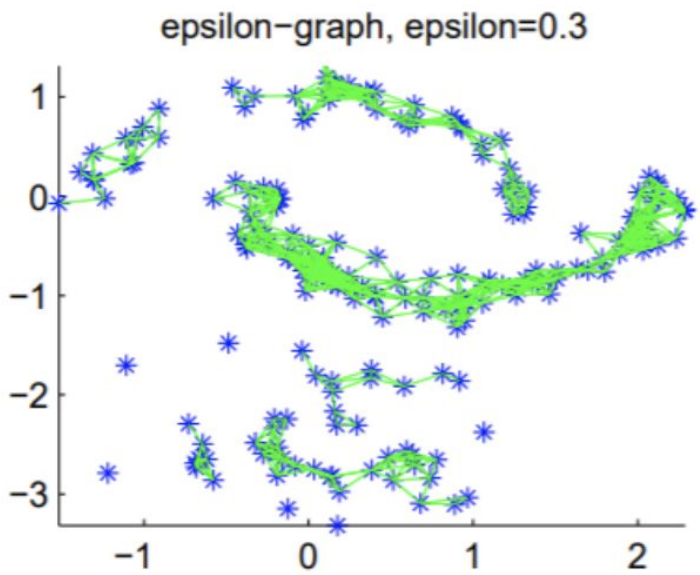
- Assumption: we can represent the data as an **undirected graph** (as opposed to a directed graph), where each edge holds a certain **weight**
- This can be done in multiple ways (k-nearest-neighbor, ε-neighborhood)





Spectral Clustering: Similarity Graph Example



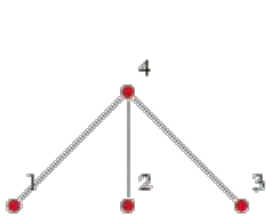


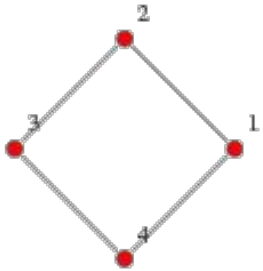
https://towardsdatascience.com/spectral-clustering-for-beginners-d08b7d25b4d8

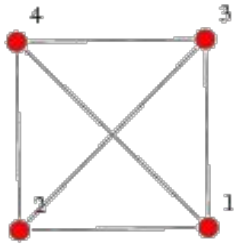
Spectral Clustering: Graph Terminology

- A key element in spectral clustering is finding the **Laplacian matrix** of the graph that represents the data.
- What we need:
  - A (adjacency matrix)
  - W (weight matrix)
  - D (degree matrix)

Spectral Clustering: Graph Terminology (Adjacency)



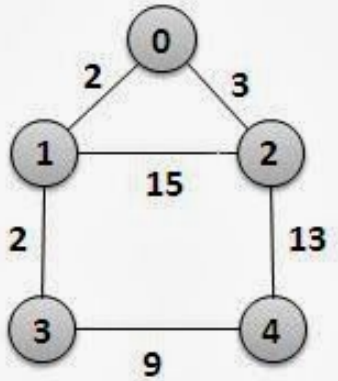




$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \end{array} \qquad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Spectral Clustering: Graph Terminology (Weight)

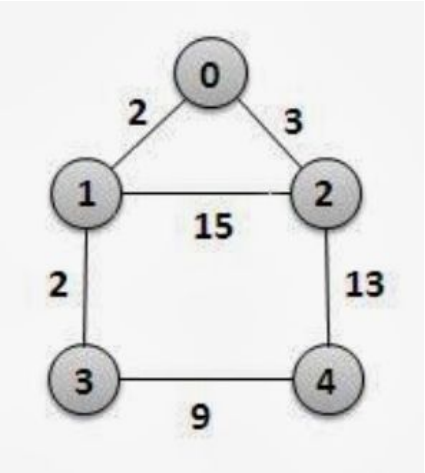
$$A_G(i,j) = \begin{cases} w(i,j) & \text{if } (i,j) \in E, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$



	0	1	2	3	4
0	0	2	3	0	0
1	2	0	15	2	0
2	3	15	0	0	13
3	0	2	0	0	9
4	0	0	13	9	0

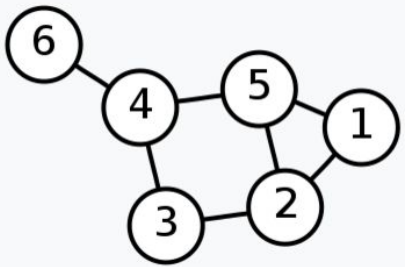
Spectral Clustering: Graph Terminology (Degree)

$$D_G(i,i) = \sum_j A_G(i,j).$$



	0	1	2	3	4
0	5	0	0	0	0
1	0	19	0	0	0
2	0	0	31	0	0
3	0	0	0	11	0
4	0	0	0	0	21

Spectral Clustering: Unnormalized Graph Laplacian

Labeled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$

- Spectral graph theory explains the importance and use of Laplacian matrices

Spectral Clustering: Graph Terminology (Eigenvectors)

- We will also use eigenvectors to uncover the connectivity of the graph
- The **spectrum** in spectral clustering refers to the eigenspace of the graph Laplacian
- For two similar graphs, the Laplacian matrices and eigenvalues will be similar or the same

Proposition 2 (Number of connected components) *Let G be an undirected graph with non-negative weights. Then the multiplicity k of the eigenvalue 0 of L equals the number of connected components $A_1, \ldots, A_k$ in the graph. The eigenspace of eigenvalue 0 is spanned by the indicator vectors $\mathbb{1}_{A_1}, \ldots, \mathbb{1}_{A_k}$ of those components.*

Unnormalized Spectral Clustering

1. Find an appropriate representation of data as an undirected graph (k-nearest-neighbor, ε-neighborhood).
2. Compute the graph Laplacian using the weighted degree matrix and weighted adjacency matrix.
3. Compute the **first (lowest) k eigenvectors** of the Laplacian matrix $\{v_1 \ldots v_k\}$. (This creates a more linearly separable eigenvector space.)
4. Represent these eigenvectors as a matrix $\mathcal{V}$, where each v_i is a column.
5. Perform k-means on the rows of $\mathcal{V}$, the new representation of the data points in similarity space (acts similarly to dimensionality reduction)

Unnormalized Spectral Clustering: Algorithm

Unnormalized spectral clustering

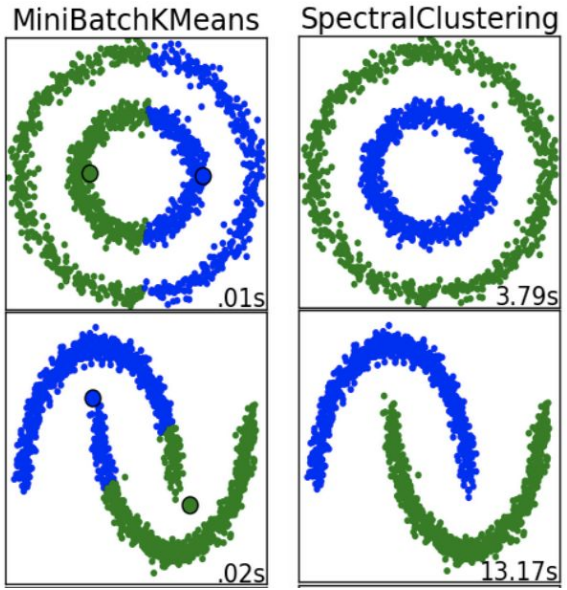
Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$, number k of clusters to construct

- Construct a similarity graph by one of the ways described in Section 2. Let W be its weighted adjacency matrix.
- Compute the unnormalized Laplacian L.
- **Compute the first k eigenvectors $v_1, \ldots, v_k$ of L.**
- Let $V \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $v_1, \ldots, v_k$ as columns.
- For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of V.
- Cluster the points $(y_i)_{i=1,\ldots,n}$ in $\mathbb{R}^k$ with the k-means algorithm into clusters $C_1, \ldots, C_k$.

Output: Clusters $A_1, \ldots, A_k$ with $A_i = \{j | \, y_j \in C_i\}$.

Characteristics of Spectral Clustering

- Slow (eigenvector computation is a bottleneck)
- Better than k-means with getting the underlying structure of data (connectivity-focused, not necessarily linearly separable)
- Fixed number of clusters



http://cobweb.cs.uga.edu/~squinn/mmd_s15/lectures/lecture10_feb4.pdf

Code Example!

References & Further Reading

1. Zha, H., He, X., Ding, C., Simon, H., and Gu, M. Spectral relaxation for k-means clustering. In Proc. NIPS, 2001.
2. Kulis, Brian, and Jordan, Michael. Revisiting k-means: New algorithms via Bayesian nonparametrics. In Proc. ICML, 2012.
3. Vattani, Andrea. The hardness of k-means clustering in the plane. Manuscript, accessible at http://cseweb.ucsd.edu/avattani/papers/kmeans_hardness.pdf 617 (2009).
4. Von Luxburg, Ulrike. A tutorial on spectral clustering. Statistics and computing 17.4 (2007): 395-416.
5. Sasirekha, K., and P. Baby. Agglomerative hierarchical clustering algorithms - a review. International Journal of Scientific and Research Publications 83 (2013): 83.