Cluster validity index

MINI PROJECT CH4025

ABHINAV T K
CH17BTECH11002

Objective

To develop a Python code for finding the cluster validity index (SSDD index) discussed in the paper: **Cluster validity index for irregular clustering results**

Introduction

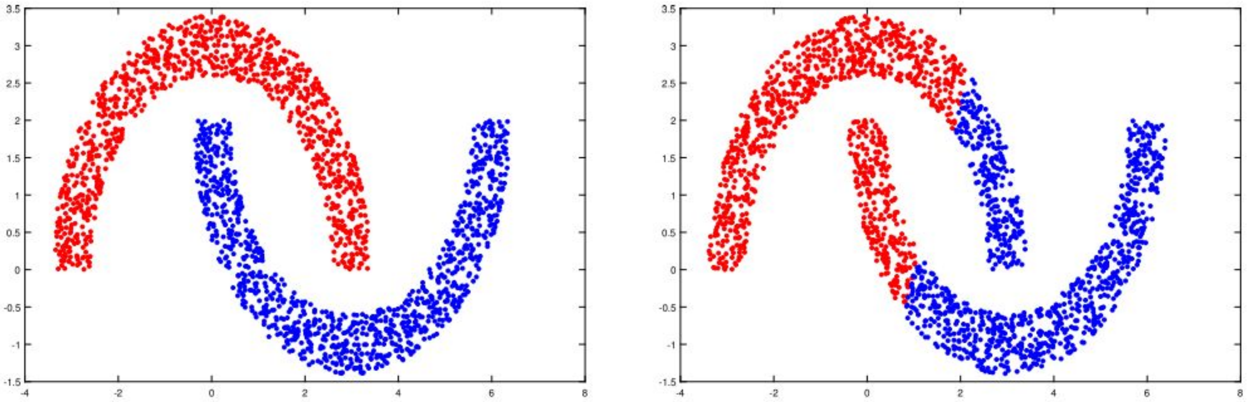
- Clustering is an unsupervised learning process aiming to discover the underlying structure of the input data space, which has been widely applied in image processing, data mining, text information processing, etc.
- It is difficult for users to select the proper clustering algorithm and the parameters for the specific data in advance.
- The **cluster validity index** (CVI) is used to help select the best clustering algorithm that fits the underlying structure of the data.

Different CVIs

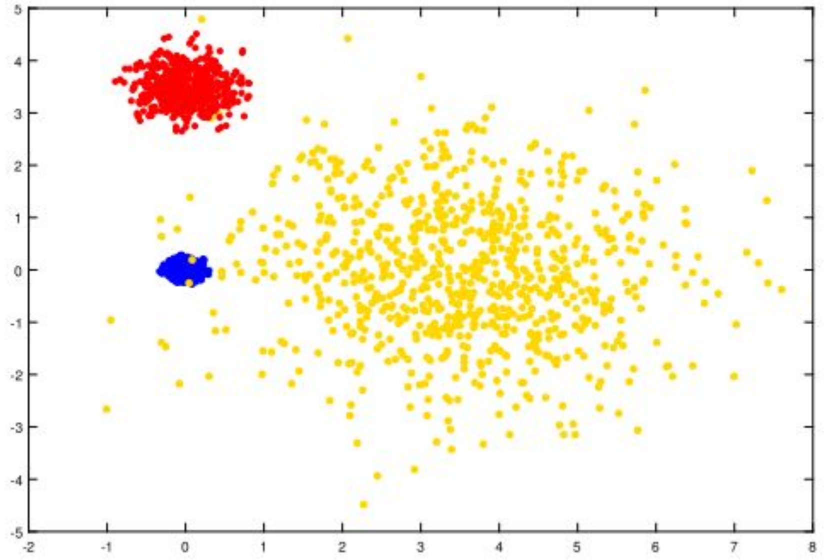
- Dunn index
- Davies–Bouldin Index
- S_Dbw Index
- CDbw Index
- DBCV Index
- Average Normality Index

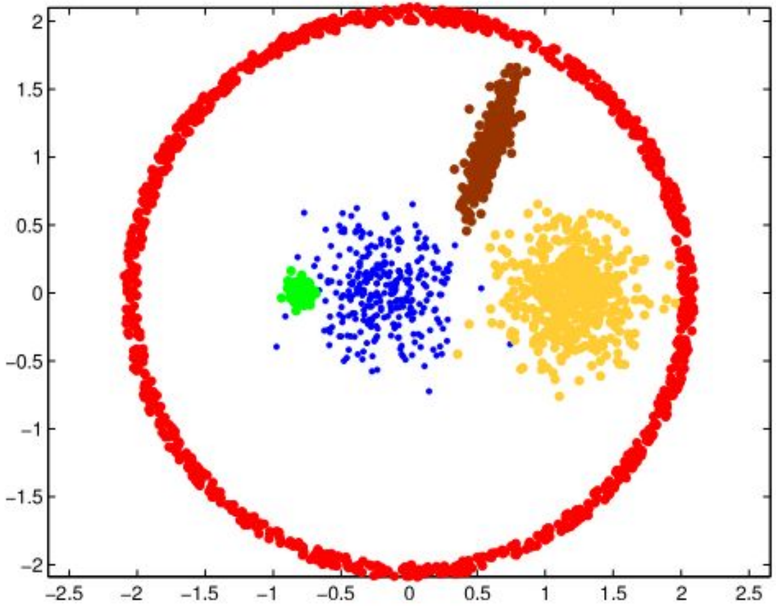
Drawbacks of existing CVIs

- Most traditional CVIs assume that the clusters are spherically distributed, therefore, they cannot handle the density based clustering results with **arbitrarily shaped clusters**.
- When the clustering result becomes more complicated where the clusters are at the same time in non-spherical shapes, different sizes and densities the CVIs fail.



(a) ground truth (b) bad partitioning



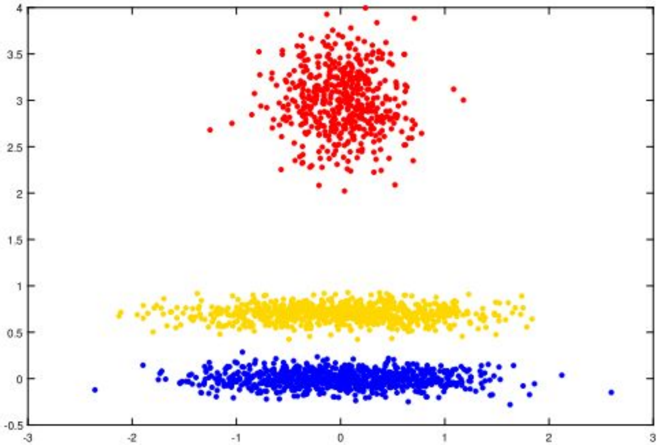


More examples of clusters where commonly used CVIs fail

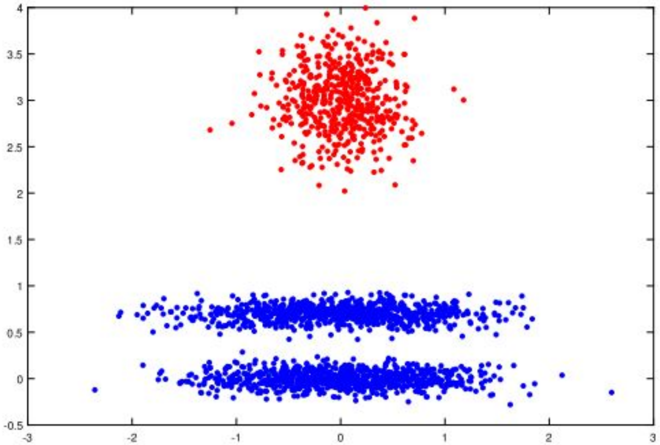
New cluster validity index: SSDD index

- The new index is designed for **hard clustering** with irregular clustering results.
- Here the clusters are in **arbitrary shapes**, **different sizes** & **densities**, and with **small separation distances**.
- In the SSDD index, we assume that good clusters are high density regions surrounded by low density regions and separated from other high density regions.

- The SSDD index is sensitive to the existence of low density regions between multiple density peaks within a cluster and can adaptively determine whether the distances between clusters are big enough to well separate them.



(a) ground truth

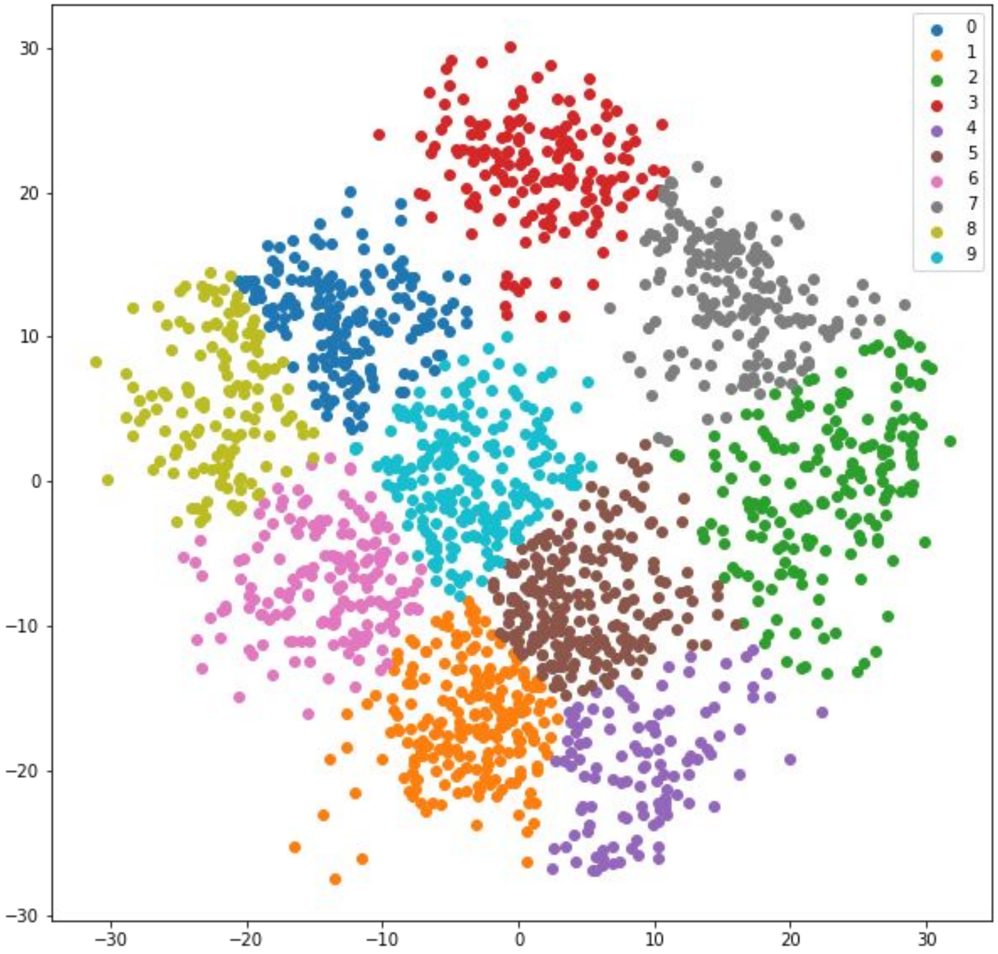


(b) bad partitioning

SSDD index of KMeans clustering on MNIST data

MNIST Data

- Dataset of handwritten digits from 0 to 9.
- Shape: (1797,64)
- Did dimensionality reduction using PCA.
- Applied KMeans clustering with 10 clusters.



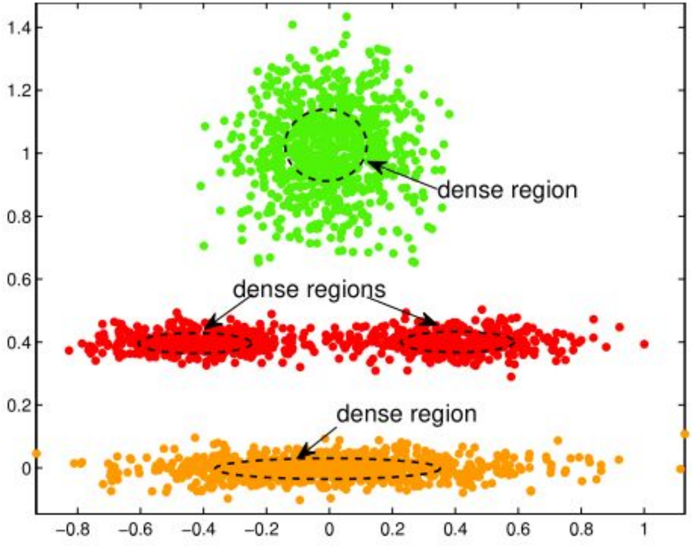
30
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30
0
1
2
3
4
5
6
7
8
9

Applying SSDD algorithm

- First separated data points of each clusters into separate arrays.
- Part 1: **Inner-cluster evaluation**: based on the density changes along the backbone of a cluster.
- Part 2: **Inter-cluster evaluation**: based on the density changes from the inner-cluster region to the adaptively determined inter-cluster regions.

Part 1: Inner-cluster evaluation

- Inner-cluster validation measure is to check whether high density regions are separated by low density regions within a cluster.



Definition 1: Backbone Points & Cluster's Backbone

- Data point x is a backbone point of its cluster if the local density of x, LD(x) exceeds the local density of its neighbors, i.e., **LD(x) > LD(xi)**, where xi ∈ kNN(x, l), where kNN(x, l) represents the l-nearest neighbors of x.

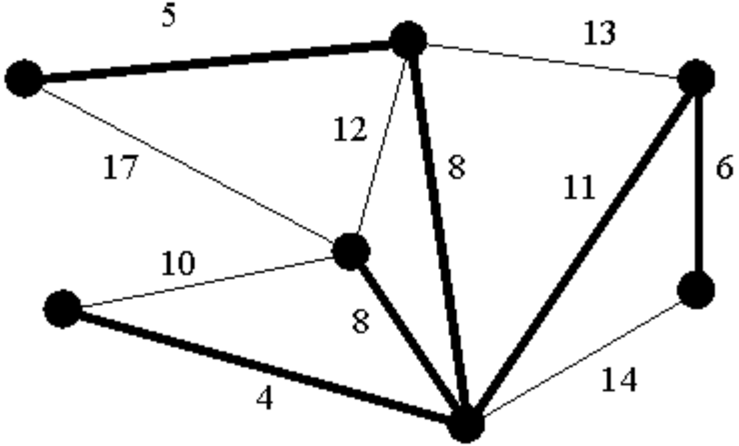
$$LD(x) = \frac{1}{\frac{1}{l}\sum_{x_l \in kNN(x,l)} d(x, x_l)}$$

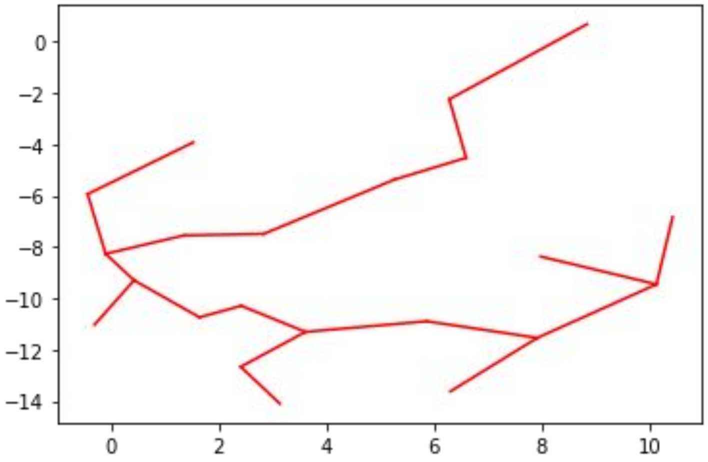
- The backbone points are the data points that have the **maximum local density** in their neighborhood.
- The backbone points form a cluster's backbone, which is consistent with the cluster's structure, no matter how its density or size changes.

```python
def find_backbone_points(C,l):
  """
  C : cluster
  l: no. of nearest neighbours  (user-input)

  Returns the backbone points of the cluster C.
  """


  backbone_points = []  # This list contains the index of the backbone points
  knn_list = [] # This list contains the information of nearest neighbors of each point in the cluster.
  LD = [] # This list contains local density of each point in the cluster.
  knn = NearestNeighbors(n_neighbors=l+1) # Fit the nearest neighbors estimator from the training dataset.
  knn.fit(C)
  for i in range(len(C)):
    a = knn.kneighbors(C[i].reshape(1,-1))  # Finds the K-neighbors of a point and returns a
    knn_list.append(a)                      # list containing indexes of those points  and the distances between the points.
    LD.append(l/a[0].sum()) # Adding Local density of each point to the list
  for i in range(len(C)): # Checking each point whether it is a backbone point
    ind = 0
    for j in range(1,l+1):
      if LD[i]>LD[knn_list[i][1][0][j]]:      # Condition for backbone point  # knn_list[i][1][0][j] gives the index of the nearest neighbor
        ind+=1
    if ind ==l:
      backbone_points.append(i)
  backbone_array = []     # This list contains the backbone points
  for i in backbone_points:
    backbone_array.append(C[i])
  backbone_array = np.array(backbone_array)
  return backbone_array
```

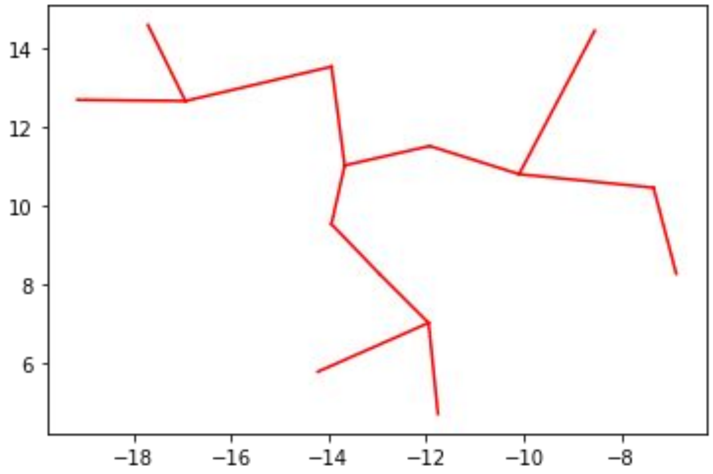
Definition 2: Density Changes (DC) Along Cluster's Backbone

- Find minimum spanning tree (MST) of a cluster with backbone points as vertices and the distances between them as the weight of the edges.
- A **minimum spanning tree** (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected **graph** that connects all the vertices together, without any cycles and with the minimum possible total edge weight.





MST of backbone points of cluster 0



MST of backbone points of cluster 1

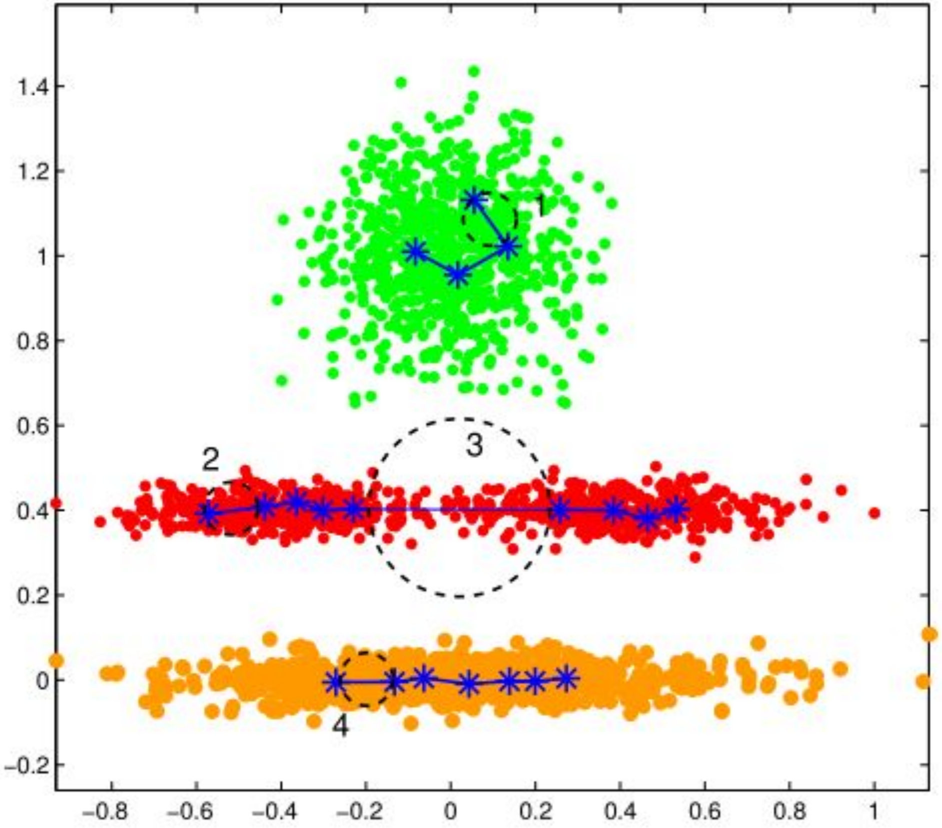
```python
def minimum_spanning_tree(X, copy_X=True):
    """X are edge weights of fully connected graph"""
    if copy_X:
        X = X.copy()

    if X.shape[0] != X.shape[1]:
        raise ValueError("X needs to be square matrix of edge weights")
    n_vertices = X.shape[0]
    spanning_edges = []

    # initialize with node 0:
    visited_vertices = [0]
    num_visited = 1
    # exclude self connections:
    diag_indices = np.arange(n_vertices)
    X[diag_indices, diag_indices] = np.inf

    while num_visited != n_vertices:
        new_edge = np.argmin(X[visited_vertices], axis=None)
        # 2d encoding of new_edge from flat, get correct indices
        new_edge = divmod(new_edge, n_vertices)
        new_edge = [visited_vertices[new_edge[0]], new_edge[1]]
        # add edge to tree
        spanning_edges.append(new_edge)
        visited_vertices.append(new_edge[1])
        # remove all edges inside current tree
        X[visited_vertices, new_edge[1]] = np.inf
        X[new_edge[1], visited_vertices] = np.inf
        num_visited += 1
    return np.vstack(spanning_edges)
```

- BAVDens: Density of the region between two adjacent vertices of a MST.

$$BAVDens = \frac{|inside(v_a, v_b)|}{[d(v_a, v_b)]^{dim}}$$

  - where v_a and v_b are one pair of adjacent vertices in MST,
  - the numerator is the number of datapoints inside the hypersphere with $\overline{v_a v_b}$ as diameter.
  - $d(v_a v_b)$ is the distance between v_a and v_b and dim is the dimensionality of the input data space.

```python
def BAVDens_finder(edge_list, q, backbone_array, C):
  """
  Input
  edge_list: list of edges in the MST of a cluster
  q: array containing distance between backbone points
  backbone_array: array containing backbone points of a cluster
  C : a cluster
  """
  BAVDens = [] # This list contains the BAVDens of a cluster
  for i in range(len(edge_list)):
    point1 = edge_list[i][0]
    point2 = edge_list[i][1]
    radius = q[point1,point2]/2     # radius of the circle with each MST edges as a diameter.
    midpoint = (backbone_array[point1] + backbone_array[point2])/2 # midpoint
    count = 0
    for j in C:
      if (j[0]-midpoint[0])**2+(j[1]-midpoint[1])**2 <= radius**2:    # condition for points inside the circle
        count+=1
    BavDen = count/((2*radius)**2)    # power ==> dim represents the dimensionality of the input data space (2)
    BAVDens.append(BavDen)
  return BAVDens
```



Clusters' backbones and their MSTs.

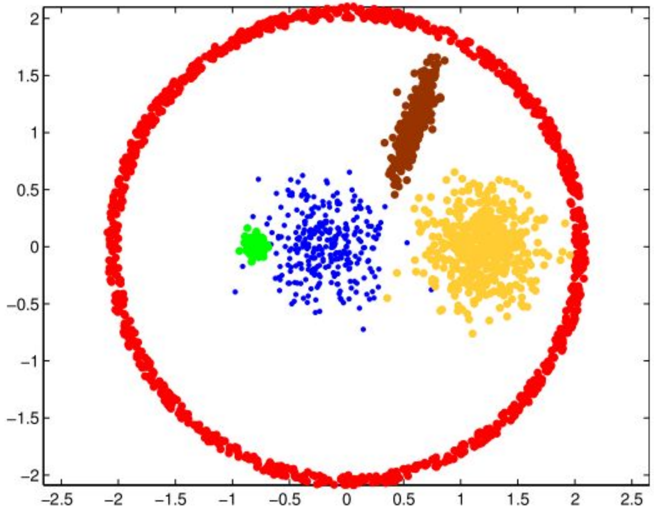
- Given the BAVDens between all pairs of adjacent vertices in MST, the density changes of a cluster ci, DC(ci), along its backbone is defined as:

$$DC(c_i) = \frac{\max(BAVDens) - \min(BAVDens)}{\max(BAVDens)}$$

- In SSDD, DC is used as the inner-cluster validation measure. A **smaller DC(ci)** value indicates a more homogeneous dense region in cluster ci, which represents a better formed cluster.

Part 2: Inter-cluster evaluation

- Inter-cluster separation measure check whether the density of the region between ci and all other clusters is significantly lower than the inner-cluster's density of ci itself.

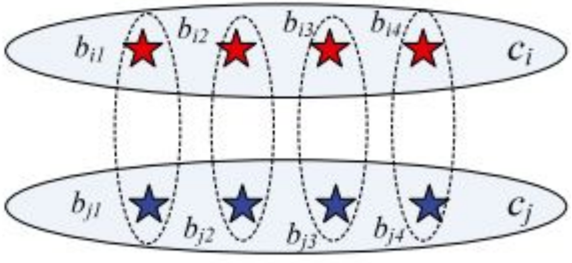


Here the distance between the green and the blue cluster is small, but they are still well separated because the density of the region between the two clusters is lower than the density of the inner region of the green cluster.

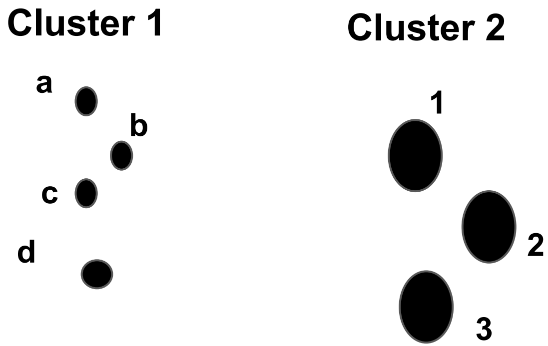
Definition 3: Inter-Cluster Nearest Data Pair

- We find nearest backbone points pairs and inter-cluster nearest data points.

Nearest backbone points pairs:



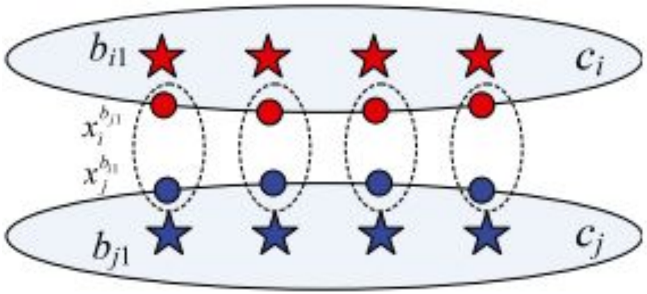
nearest backbone points pairs



Backbone point pairs between cluster 1 and 2 are: (b,1), (d,3)

```python
def backbone_point_pairs(backbone_array1,backbone_array2):
  """
  To find nearest backbone point pairs based on the condition
  Input: backbone arrays of two clusters
  Return: nearest backbone_point_pairss between two clusters based on the condition
  """
  backbone_point_pairss = []
  m = backbone_array1.shape[0]  # no of backbone points in cluster 1
  n = backbone_array2.shape[0]  # no of backbone points in cluster 2
  distance_matrix = np.zeros(shape=(m,n)) # array containing distance between two backbone points
  for i in range(m):
    for j in range(n):
      distance_matrix[i,j] = round(math.sqrt((backbone_array1[i][0] - backbone_array2[j][0])**2 + (backbone_array1[i][1] - backbone_array2[j][1])**2),2)

  # finding nearest backbone_point_pairs using given condiion
  for i in range(m):
    row_min = min(distance_matrix[i,:])
    itemindex = np.where(distance_matrix[i,:]==row_min)
    col = itemindex[0][0]
    col_min = min(distance_matrix[:,col])
    if row_min == col_min:
      backbone_point_pairss.append((i,col))
  return backbone_point_pairss
```

Inter-cluster nearest data pair: For a nearest backbone points pair (b1,b2) between clusters 1 and 2, assume that xi is the data point in C1 that is closest to b2 and xj is the data point in C2 that is closest to b1, then (xi,xj) is called a inter-cluster nearest data pair between C1 and C2.



inter-cluster nearest data pairs

```
# List of inter-cluster nearest data pairs
data_pairs_list = [0]*n
for i in range(n):
  data_pairs_list[i]= [0]*n
  for j in range(n):
    data_pairs_list[i][j]=[]
    if backbone_point_pairs_list[i][j]== None:
      data_pairs_list[i][j].append(None)
    else:
      for k in backbone_point_pairs_list[i][j]:
        index1 = find_index(backbone_array_list[j][k[1]], cluster_list[i])
        index2 = find_index(backbone_array_list[i][k[0]], cluster_list[j])
        data_pairs_list[i][j].append((index1,index2))
```

```python
def find_index(backbone_point, cluster):
  """
  Finds the point in the cluster that is nearest to the backbone point
  """
  min_dist = 1000000
  index = None
  for i in range(len(cluster)):
    dist = find_distance(cluster[i],backbone_point)
    if dist < min_dist:
      min_dist = dist
      index = i
  return index
```

Definition 4: Density of Region Between One Cluster and All Other Clusters

- Define region between clusters: Suppose (p_a, p_b) is one inter-cluster nearest data pair between cluster ci and cj. The region enclosed by the hyper-sphere centered at the middle point of $p_a p_b$ with a certain radius Rij is defined as one of the intercluster regions between ci and cj.
- Here, $$R_{ij} = \frac{\min\{ICD(c_i), ICD(c_j)\}}{2}$$
- ICD represents the Inner-cluster Core Distance which is defined as the average length of all the edges in MST built for all b in cluster ci.
- Now the density of region between clusters i and j: $$BNPDens_{p_a, p_b} = \frac{|inside_{R_{ij}}(p_a, p_b)|}{(2R_{ij})^{dim}}$$
- Where numerator is the number of datapoints inside the hypersphere centred at the midpoint of $p_a \mathbf{p_b}$ with radius Rij.
- Given all the inter-cluster nearest data pairs between a cluster ci and all the other clusters in the partitioning, the **density of the region between ci and other clusters** is defined as the average of all the BNPDens calculated (i.e., **mean(BNPDens)**).

```python
mean_BNPDens = [0]*n # This list contains mean density of the region between Ci and all other clusters
count_list = []
for c in range(n):
  BNPDens_list=[]
  for d in range(n):
    if c!=d:
      for i in data_pairs_list[c][d]:
        if i==None: continue
        else:
          point1 = cluster_list[c][i[0]]
          point2 = cluster_list[d][i[1]]
          midpoint = (point1 + point2)/2
          radius = min(ICD_list[c],ICD_list[d])/2
          count = 0
          for k in cluster_list[c]:
            if (k[0]-midpoint[0])**2+(k[1]-midpoint[1])**2 <= radius**2:
              count+=1
          for k in cluster_list[d]:
            if (k[0]-midpoint[0])**2+(k[1]-midpoint[1])**2 <= radius**2:
              count+=1
          BNPD = count/((2*radius)**2)
          BNPDens_list.append(BNPD)
  mean_BNPDens[c] = mean(BNPDens_list)
```

Definition 5: Ratio between Inter-Cluster Density and Inner-Cluster Density (DR)

$$DR(c_i) = \frac{mean(BNPDens)}{\max\{mean(BNPDens), mean(BAVDens_{ci})\}}$$

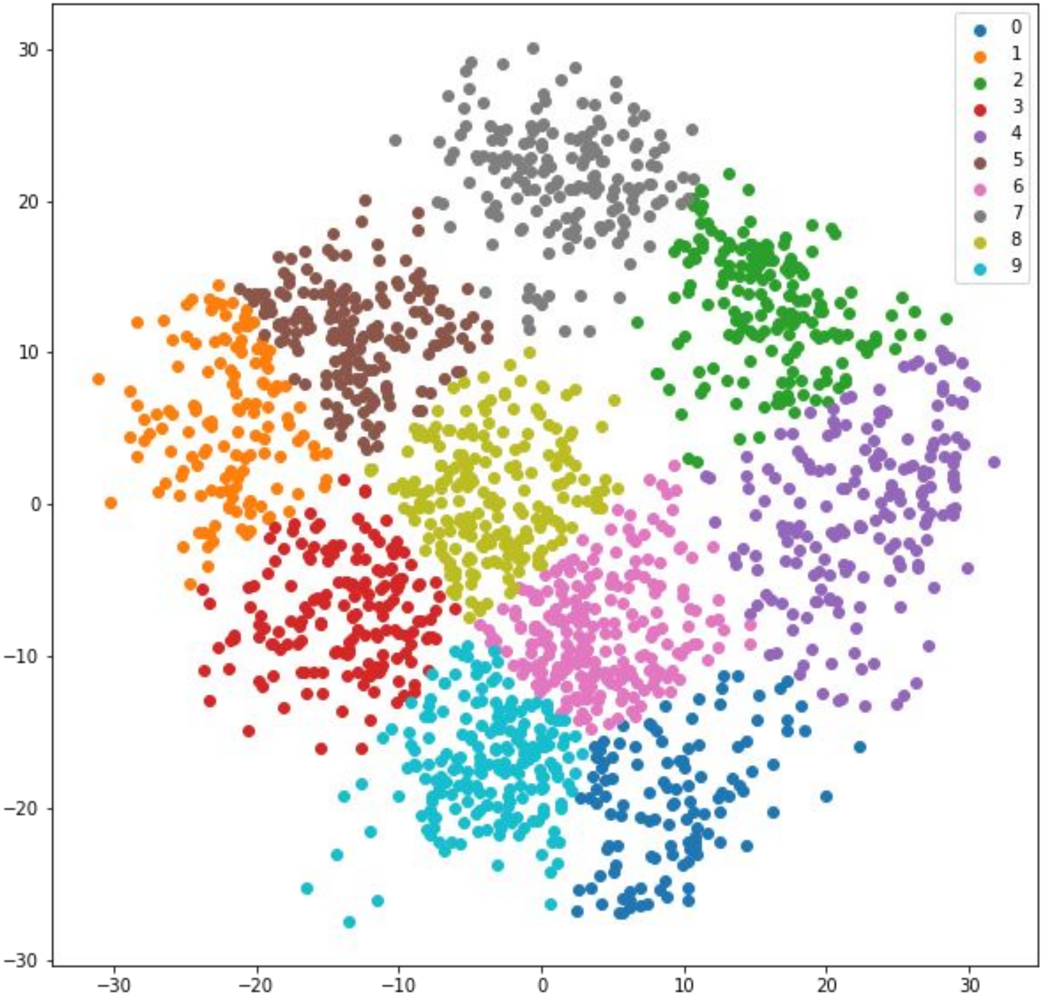
- mean(BNPDens) is the inter-cluster density between Ci and all other clusters.
- mean(BAVDens) is the average of the density of region between two backbone points in a MST.
- In SSDD, DR is used as the inter-cluster validation measure.
- A smaller value of DR(ci) indicates a more significant difference between the density of ci's inner region and that of ci's intercluster regions between other clusters, which represents that the cluster ci is better separated in the partition.

SSDD index

- Based on the inner and the inter-cluster validation measure, the SSDD index of a partition C is defined as:

$$SSDD(C) = \sum_{c_i \in C} [\alpha \cdot DC(c_i) + \beta \cdot DR(c_i)]$$

- DC and DR quantify the inner and the inter-cluster qualities respectively.
- Their contributions to the overall SSDD index are defined by α and β.
- $\alpha, \beta \in [0, 1]$, and $\alpha + \beta = 1$, which normalize the SSDD index to [0,1].
- A **smaller value** of SSDD(C) represents a better clustering result.



SSDD index of each cluster of MNIST dataset with α=0.5, β=0.5

```
[0.584380529001188,
 0.5185499531684143,
 0.5537794753829844,
 0.47979846913416524,
 0.49027701805954593,
 0.563447477208875,
 0.6855195640350977,
 0.5448139163250539,
 0.5437680824501967,
 0.6408876546783006]
```

SSDD index of the partition = 0.561

Algorithm

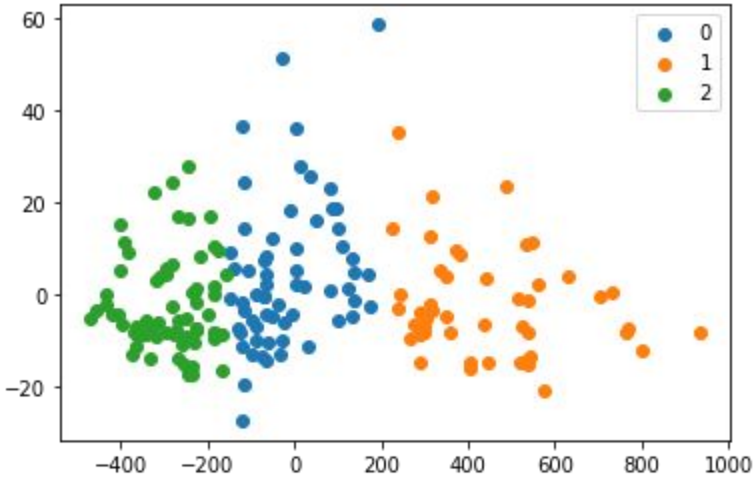
Pseudocode of SSDD.

Input: The partition C containing k clusters. The user defined neighborhood size l.
Output: The SSDD evaluation on C.

Begin:
For $i = 1$: k
  find c_i's backbone points
  build the MST_{BPs} based on the found backbone points
  calculate the *BAVDens* for all pairs of adjacent vertices in the MST_{BPs}
  calculate the density changes $DC(c_i)$
  find all the inter-cluster nearest data pairs between c_i and the other clusters
  calculate the *BNPDens* between all pairs of inter-cluster nearest data pairs
  calculate c_i's inner-cluster density and inter-cluster density ratio $DR(c_i)$
End of i
  calculate $SSDD(C)$ based on $DC(c_i)$ and $DR(c_i)$ according to Eq. (22).
End

Validation of SSDD

- Used Wine dataset which has 3 classes.
- Applied different CVIs on the dataset and selected the “best” partition founded by each CVI on the dataset.
- Find the ARI score of those best partitions.



Ground truth

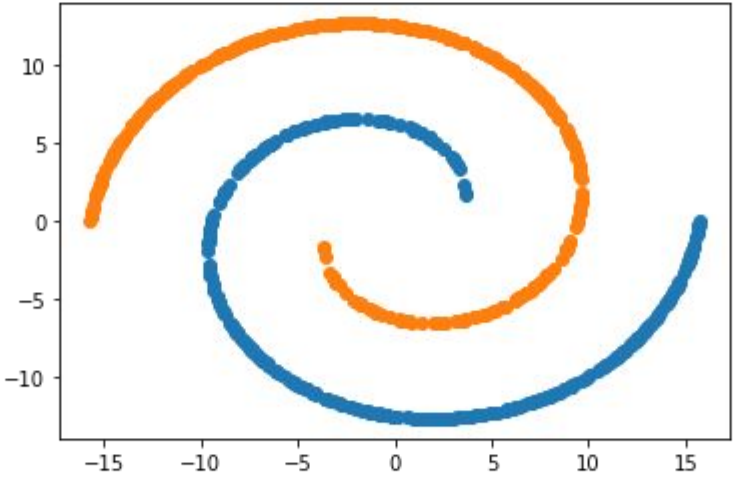
No. of clusters	ARI score	Index			
		DB (↓)	SDBW (↓)	CDBW (↑)	SSDD (↓)
2	0.369	**0.48**	0.56	0.00005	0.815
3	**0.371**	0.532	0.376	0.0035	**0.4595**
4	0.303	0.543	0.35	0.003	0.493
5	0.311	0.5431	**0.239**	**0.0056**	**-**

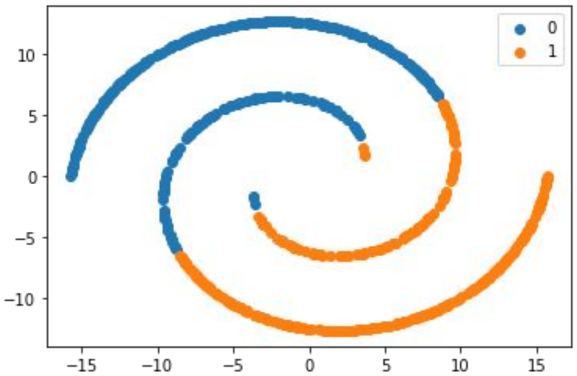
- Indices used:
  - Davies–Bouldin Index (DB): smaller value means better clustering.
  - S_Dbw index: smaller value means better clustering.
  - CDbw index: bigger value represent better clustering.
  - SSDD: smaller value represent better clustering.

- ARI score is the similarity between predicted labels after clustering to the original labels.
- Larger ARI value means better clustering.
- We can see that larger ARI value is when the no. of clusters is 3 and SSDD has found it as the best partition.

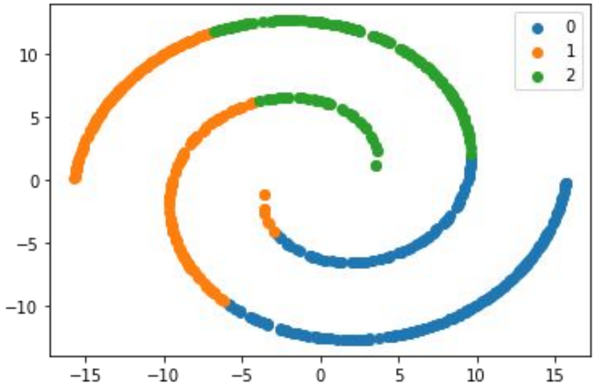
Validation on synthetic dataset

- Spiral dataset: Two intertwined spirals representing two clusters.
- Non-spherical clusters for testing SSDD.

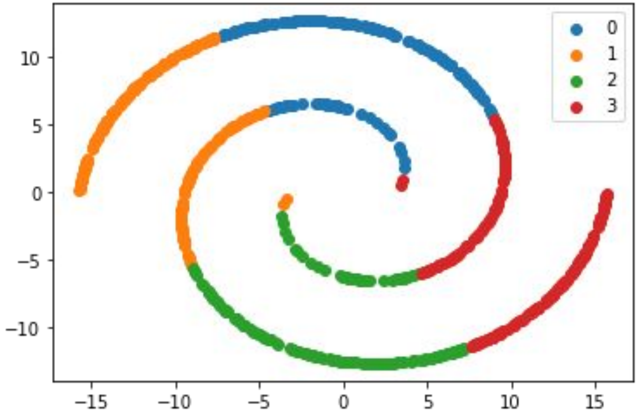




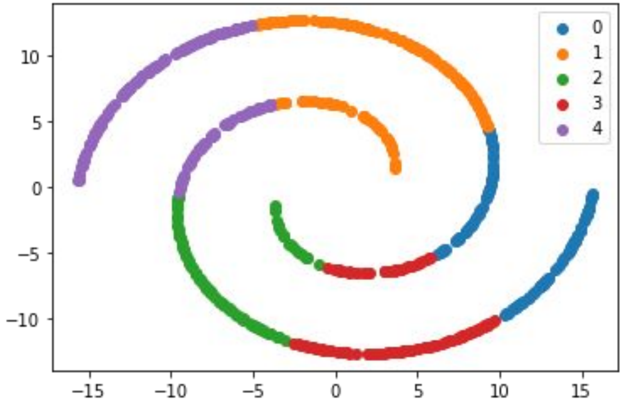
1. Kmeans, k=2



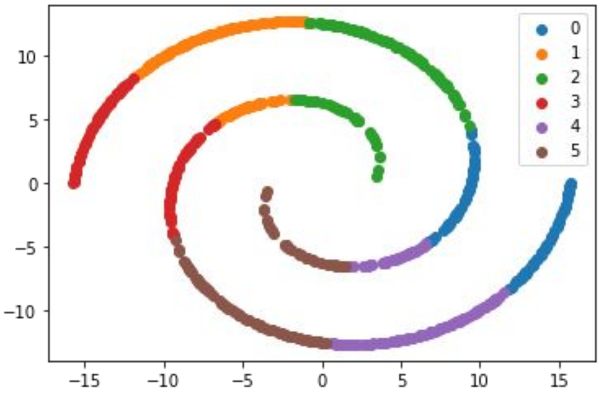
2. Kmeans, k=3



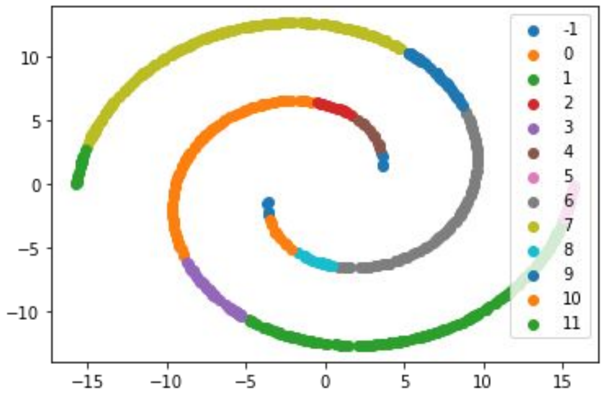
3. Kmeans, k=4



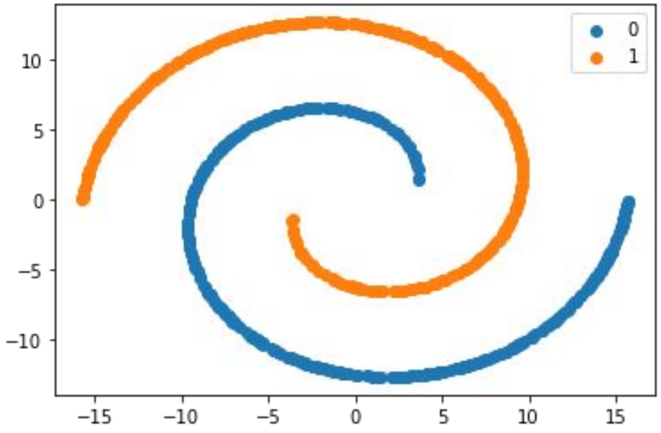
4. Kmeans, k=5



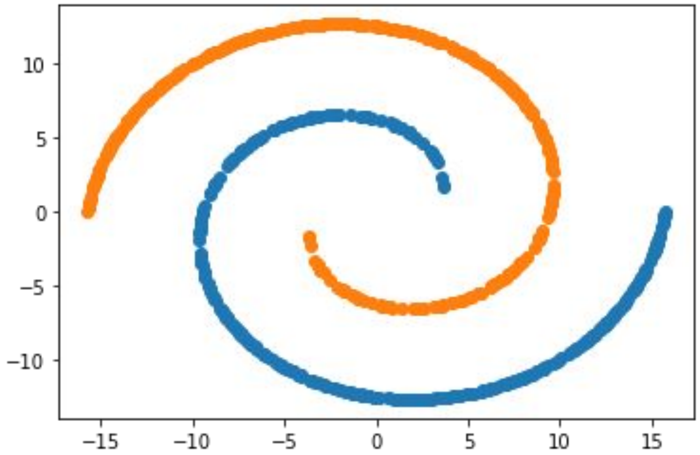
5. Kmeans, k=6



6. DBSCAN: eps=.6, minPts = 5



7. DBSCAN: eps=2, minPts = 5



Ground truth

No.	ARI score	Index					
		DB (↓)	SDBW (↓)	CDBW (↑)	Silhoutte (↑)	CH (↑)	SSDD (↓)
1	0.151	0.945	0.712	0.0007	**0.447**	775.75	0.851
2	0.11	0.805	0.552	0.0006	0.442	832.36	0.77
3	0.069	0.812	0.446	0.0019	0.439	1059.48	0.619
4	0.07	**0.785**	0.485	0.0127	0.423	1027.63	0.623
5	0.051	0.841	0.441	0.021	0.406	**1113.92**	0.595
6	0.32	0.92	**0.22**	**0.31**	0.15	270.7	**-**
7	**1**	2.155	0.909	0.0002	0.148	166.8	**0.492**

- The best partition found by SSDD has the best ARI value.

CONCLUSION

- A new cluster validity index for hard clustering called SSDD is proposed.
- SSDD characterizes a cluster by a backbone along which data points are more densely distributed.
- It implements the inner-cluster evaluation based on the density changes along the backbone of a cluster, and the inter-cluster evaluation based on the adaptive density estimation between different clusters.
- Experimental results and comparisons with other CVIs has demonstrated the effectiveness of the SSDD index for irregular clustering results.

References

- Main Paper: Cluster validity index for irregular clustering results; Shaoyi Liang, Deqiang Han, Yi Yang https://www.sciencedirect.com/science/article/abs/pii/S1568494620305214