Fuzzy DBScan Algorithm

Spatial Data Mining

Naima Farooqi 16803005

Aditya Gupta 16803004

Harsh Vishnoi 16803030

Submitted to: Mr. Mahendra Gurve & Mrs. Ankita Wadhwa

Date : 02/05/2019

Abstract

The DBScan algorithm is a well-known density-based clustering approach particularly useful in spatial data mining for its ability to find objects group with heterogeneous shapes and homogeneous local density distributions in the feature space. But this algorithm suffers from some limitations, mainly the inability to identify clusters with variable density distributions and partially overlapping borders, which is often a characteristic of both scientific data and real-world data. Here we have implemented three different fuzzy extensions of the DBScan algorithm to generate clusters with distinct fuzzy density characteristics, and have proposed a modified version of fuzzy DBScan inspired by the above mentioned fuzzy extensions of DBScan. In the classical DBScan algorithm, they have used two precise parameters (minimum points, epsilon) to define locally dense areas. In the above three fuzzy extensions of DBScan, they have used soft constraints, up to 2-4 parameters to model approximate values of the input parameters. In our modified algorithm we have reduced the parameters to only two. Our algorithm defines the membership with which each point belongs to a cluster.

Objectives

Our main objective was to propose a better density algorithm than the existing ones. Generally, it is never very accurate to say that a particular point belongs to one and only one cluster. Hence we have fuzzified the algorithm and have assigned a membership value to each data point with which it belongs to a particular cluster. We have discussed the evaluation of our proposals on real-world datasets by comparing them w.r.t. state of the art soft clustering approaches. We choose as competitors the Fuzzy C-Means algorithm (FCM) due to its popularity, and the (FN-DBSCAN) (Soft-DBSCAN) as representative of fuzzy density-based approaches extending DBSCAN. We have also compared our results with three different fuzzy DBScan algorithms -: Fuzzy Core DBSCAN (FCore), Fuzzy Border DBSCAN (border) and Fuzzy DBSCAN (Fuzzy DBScan)(both combined).

Background study and findings

We took help from a research paper based on fuzzy DBScan algorithms. That research paper has proposed three different fuzzy extensions of DBScan algorithm: Fuzzy Core DBSCAN (FCore), Fuzzy Border DBSCAN (FBorder) and Fuzzy DBSCAN (FDBScan).

Fuzzy Core DBScan is obtained by considering crisp distances and by introducing an approximate value of the minimum cardinality of the local neighborhood of a point. Here they fave fuzzified the parameter Minimum Points and split them into MAX_minPts and MIN_minPts.

Fuzzy Border DBSCAN (FBorder), can be defined by allowing the specification of an approximate value of the maximum distance instead of asking for a precise numeric parameter and in defining a soft constraint with a monotonic not increasing membership function on the positive real domain of distance values. The soft constraint defines the concept of fuzzy neighborhood size so that a point can

belong to the fuzzy neighborhood of another point to a degree in (0,1]. This allows computing a gradual membership to the clusters.
In Fuzzy DBScan, we introduce how to model fuzziness over both cores and borders in order to subsume the previously proposed approaches into what is named Fuzzy DBSCAN, i.e. FDBScan.

We have used the following formulas to test the accuracy of our algorithm.

Fuzzy Partition Index(FPI):

$$FPI = 1 - \left(\frac{|C|}{|C|-1}\right) \times \left(1 - \sum_{i=1}^{|D|}\sum_{j=1}^{|C|}\frac{\mu_{ij}^2}{|D|}\right)$$

Partition Coefficient (PC):

$$PC = \frac{1}{|D|} \times \sum_{i=1}^{|D|}\sum_{j=1}^{|C|}\mu_{ij}^2$$

F Measure:

$$FMeasure(C_j, cl) = 2 \times \frac{FPrecision(C_j, cl) * FRecall(C_j, cl)}{FPrecision(C_j, cl) + FRecall(C_j, cl)}$$

F Precision:

$$FPrecision(C_j, cl) = \frac{\sum_{i \in C_j \cap D_{cl}} \mu_{ij}}{|C_j|}$$

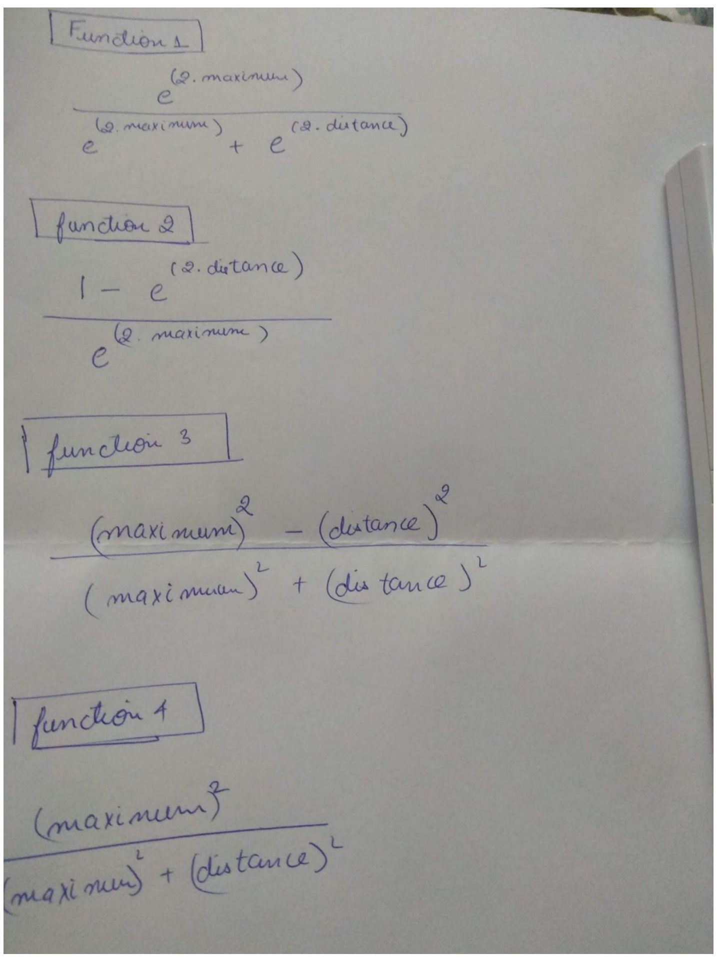
FRecall:

$$FRecall(C_j, cl) = \frac{\sum_{i \in C_j \cap D_{cl}} \mu_{ij}}{|D_{cl}|}$$

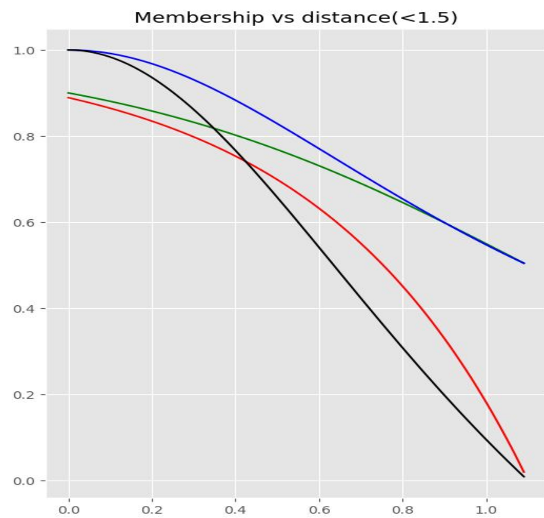
and the final *Fuzzy F-Measure* is defined as:

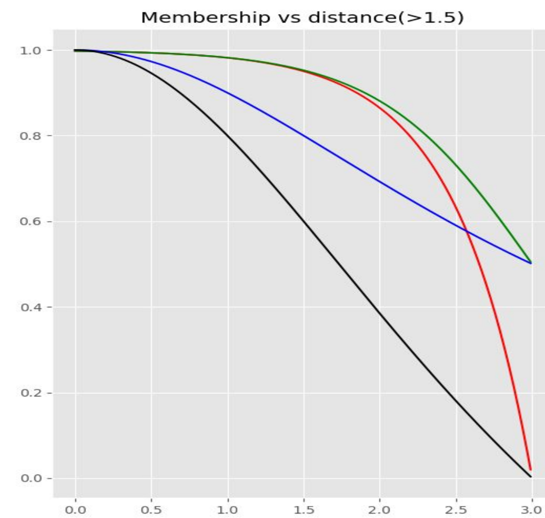
$$\sum_{C_j \in C} \frac{|C_j|}{|D|} \times \text{Fuzzy F-Measure}(C_j, cl)$$

We have tested many membership functions and chose the best among them. The final membership functions are:



The graph below shows the relation between Membership and distance. The first graph shows how membership decreases with respect to distance when the maximum distance is less than 1.5. And the second graph shows the Membership variation when the maximum distance is greater than 1.5.





Implementation

We have implemented three different fuzzy DBScan algorithms from the research paper (mentioned below). The third algorithm (4.3 Fuzzy DBScan) is been used to compare with our modified algorithm.

Algorithm of 4.3 Fuzzy DBScan algorithm is given below:

Algorithm 7 ***Fuzzy DBSCAN***$(P, \epsilon_{Max}, \epsilon_{Min}, Mpts_{Max}, Mpts_{Min})$

Require: P: dataset of points
Require: $\epsilon_{Min}, \epsilon_{Max}$: soft constraint on the distance around a point defining the point fuzzy neighbourhood size
Require: $Mpts_{Min}, Mpts_{Max}$: soft constraint on the density around a point to be considered as fuzzy core point

$C = \emptyset$
$Clusters = \emptyset$
for all $p \in P$ s.t. p is unvisited **do**
  mark p as visited
  nPts = regionQuery(p,ϵ_{Max})
  dens(p) = as in equation (5)
  if $(\mu_{minP}(dens(p)) == 0)$ **then**
    mark p as NOISE
  else
    C = next cluster
    $Clusters = Clusters \cup expandClusterFuzzy(p, nPts, C, \epsilon_{Max}, \epsilon_{Min}, Mpts_{Max}, Mpts_{Min})$
  end if
end for
return $Clusters$

Algorithm 8 $expandClusterFuzzy(p, nPts, C, \epsilon_{Max}, \epsilon_{Min}, Mpts_{Max}, Mpts_{Min})$

Require: p: the point just marked as visited
Require: $nPts$: the points in the fuzzy neighbourhood of p
Require: C: the actual cluster
Require: $\epsilon_{Max}, \epsilon_{Min}$: soft constraint on the distance around a point to compute its fuzzy neighbourhood
Require: $Mpts_{Min}, Mpts_{Max}$: soft constraint on the density around a point to be considered a fuzzy core point

add p to C (as core) with membership $\mu_{MinP}(dens(p))$
for all $p' \in nPts$ **do**
  mark p' as visited
  if $\mu_{minP}(dens(p')) > 0$ **then**
    $nPts'$ = regionQuery(p',ϵ_{Max})
    $nPts = nPts \cup nPts'$
    add p' to C (as core) with membership $\mu_{MinP}(dens(p'))$
  else
    add p' to C (as border point) with membership computed by equation (6)
  end if
end for
return C

The snapshot of our modified algorithm is given below:

Fuzzzy DBScan:

```
Require: P: dataset of points
Require: Epsilon: distance around a point which define fuzzy neighbourhood size.
Require: Points: Minimum number of points require to declare current point as core.

For all p Ɛ P do
        If len(Membership[p]) == 0 then
        Npts = regionQuery (p, Epsilon)
        Max_dist = max (p-Npts)
        BorderPts = p' Ɛ Npts where (p'-p) == Max_dist
        If (len (Npts) < Points) then
                Mark p as noise
        Else
                C = next Cluster
                Membership[p][C] = 1
                Cluster = Cluster U
                ExpandCluster (DistanceMatrix, p, Epsilon, Points, Npts, BorderPts,
                                Membership, Max_dist)
        End if
End for
Return Membership
```

ExpandCluster:

```
Require: P:dataset of points
Require: Epsilon: distance around a point which define fuzzy neighbourhood size.
Require: Points: Minimum number of points require to declare current point as core.
Require: Npts: Neighbourhood of Point to expand
Require: C: current cluster Index
Add p to C as core with membership = 1
For all p' Ɛ Npts do
        If p' not in Borderpts then
                Assign Membership to p' according to the proposed Membership function
        End if
End for
For all b' Ɛ BorderPts do
        Npts' = regionQuery(b', Epsilon)
        Max_dist = max(b' – Npts')
        If (Npts' > Points) then
                Membership[b'][C] = 1
                For p' in Npts' do
                        Assign membership to all Npts' according to the proposed membership function
                End for
        Else
                Membership[b'][C] = 0
        End if
End for
return
```

Results

Following is the comparison of our algorithm to all the other algorithms/functions. The comparison is based on three different measures. Algorithm 4.3 is given above. The rest of the functions are defined below.

Accuracy	Algo 4.3	Proposed Algo
Compound	0.83	**0.88**
Jain	**1**	0.99
Aggregation	0.98	**1**
Path_based	0.88	**0.98**
Spiral	1	**1**
d31	0.48	**0.8**
r15	0.993	**0.993**
Flame	0.98	**0.99**

Time Required	Algo 4.3	Proposed Algo
Compound	0.087	**0.0625**
Jain	0.109	**0.046**
Aggregation	0.156	**0.094**
Path_based	0.078	**0.031**
Spiral	**0.015**	0.062
d31	**1.876**	6.95
r15	**0.093**	0.1186
Flame	0.03	**0.01**

F measure	Algo 4.3	Function 1	Function 2	Function 3	Function 4
Compound	**0.87**	0.78	0.76	0.64	0.75
Jain	**1**	0.93	0.92	0.77	0.88
Aggregation	0.71	**0.99**	0.98	0.74	0.87
Path_based	**0.96**	0.93	0.91	0.67	0.93
Spiral	**1**	0.98	0.97	0.86	0.93
d31	0.58	0.64	0.54	0.59	**0.72**
r15	**0.98**	0.71	0.57	0.76	0.89
Flame	0.98	**1**	**1**	0.73	0.91

PC	Algo 4.3	Function 1	Function 2	Function 3	Function 4
Compound	**0.89**	0.733	0.7	0.53	0.69
Jain	**1**	0.87	0.86	0.64	0.79
Aggregation	0.71	**0.96**	0.95	0.56	0.76
Path_based	**1**	0.88	0.85	0.51	0.71
Spiral	**1**	0.95	0.94	0.77	0.87
d31	0.69	0.57	0.44	0.53	**0.72**
r15	**0.97**	0.53	0.36	0.62	0.79
Flame	0.99	**1**	**1**	0.52	0.8

FPI	Algo 4.3	Function 1	Function 2	Function 3	Function 4
Compound	**0.87**	0.73	0.7	0.52	0.68
Jain	**1**	0.87	0.86	0.64	0.78
Aggregation	0.64	**0.96**	0.95	0.55	0.76
Path_based	**1**	0.88	0.85	0.5	0.71
Spiral	**1**	0.95	0.94	0.77	0.87
d31	0.67	0.57	0.44	0.53	**0.72**
r15	**0.97**	0.52	0.35	0.61	0.78
Flame	0.99	**1**	**1**	0.44	0.76

Function 1:

Function2:

Function3:

Function4:

Source Code of our modified algorithm is given below:

File Edit Selection Find View Goto Tools Project Preferences Help

Fuzzy_DBSCAN_1.py | new_algo.py | Corresponding_values_of_dataset.txt | graph_plot.py

```python
import  numpy  as  numpy
import  scipy  as  scipy
from  sklearn  import  cluster
import  matplotlib.pyplot  as  plt
import time

def sequence_cluster(Output):
    l  =  list(set(Output))
    for i in range(len(l)):
        for k in range(len(Output)):
            if Output[k] == l[i]:
                Output[k] = i+1
    return Output

def  set2List(NumpyArray):
    list  =  []
    for  item  in  NumpyArray:
        list.append(item.tolist())
    return  list

def  member(value,MinumumPoints,Max_Points):
    if  value<=MinumumPoints:
        return  0
    elif  value<Max_Points:
        return  (value  -  MinumumPoints)/(Max_Points  -  MinumumPoints)
    else:
        return  1
def Max_Distance(i,PointNeighbors,DistanceMatrix):
    maximum = 0
    for k in PointNeighbors:
        if DistanceMatrix[i][k] > maximum:
            maximum = DistanceMatrix[i][k]
    return maximum

def calculate_Membership(maximum,distance,Epsilon):
    # return ((2.718**(maximum*2))/(2.718**(maximum*2)+2.718**(distance*2)))   #new1
    # return (1 - 2.718**(2*distance)/2.718**(2*maximum))                      #new2
    # return ((maximum**2 - distance**2)/(maximum**2 + distance**2))            #new3
```

Line 11, Column 34 | Spaces: 4 | Python

File Edit Selection Find View Goto Tools Project Preferences Help

Fuzzy_DBSCAN_1.py | new_algo.py | Corresponding_values_of_dataset.txt | graph_plot.py

```python
def calculate_Membership(maximum,distance,Epsilon):
    # return ((2.718**(maximum*2))/(2.718**(maximum*2)+2.718**(distance*2)))   #new1
    # return (1 - 2.718**(2*distance)/2.718**(2*maximum))                      #new2
    # return ((maximum**2 - distance**2)/(maximum**2 + distance**2))            #new3
    if Epsilon<1.8:
        return (maximum**2/(maximum**2+distance**2))                          #new4
    else:
        return ((2.718**(maximum*2))/(2.718**(maximum*2)+2.718**(distance*2))) #new1
def  DBSCAN(Dataset,Epsilon, Points,DistanceMethod  =  'euclidean'):
    m,n = Dataset.shape
    Type = numpy.zeros(m)
    Membership = []
    for i in range(m):
        Membership.append({})
#      -1  noise    outlier
#     0  border
#     1  core
    ClustersList=[]
    Cluster=[]
    PointClusterNumber=numpy.zeros(m)
    PointClusterNumberIndex=1
    PointNeighbors=[]
    DistanceMatrix  =  scipy.spatial.distance.squareform(scipy.spatial.distance.pdist(Dataset,DistanceMethod))

    BorderPoint = []
    for  i  in  range(m):
        BorderPoint = []
        if  len(Membership[i]) == 0:
            PointNeighbors = numpy.where(DistanceMatrix[i]<=Epsilon)[0]
            PointNeighbors = set2List(PointNeighbors)

            Maximum = Max_Distance(i,PointNeighbors,DistanceMatrix)

            for k in PointNeighbors:
                if DistanceMatrix[i][k] == Maximum:
                    BorderPoint.append(k)

            if len(PointNeighbors) >= Points:
```

Line 11, Column 34 | Spaces: 4 | Python

File Edit Selection Find View Goto Tools Project Preferences Help

Fuzzy_DBSCAN_1.py | new_algo.py | Corresponding_values_of_dataset.txt | graph_plot.py

```python
    BorderPoint = []
    for  i  in  range(m):
        BorderPoint = []
        if  len(Membership[i]) == 0:
            PointNeighbors = numpy.where(DistanceMatrix[i]<=Epsilon)[0]
            PointNeighbors = set2List(PointNeighbors)

            Maximum = Max_Distance(i,PointNeighbors,DistanceMatrix)

            for k in PointNeighbors:
                if DistanceMatrix[i][k] == Maximum:
                    BorderPoint.append(k)

            if len(PointNeighbors) >= Points:
                Membership[i][PointClusterNumberIndex] = 1
                Type[i] = 1
                ExpandCluster(DistanceMatrix,i,Epsilon,Points,PointNeighbors,BorderPoint,Membership,PointClusterNumberIndex,Type,Maximum)
                PointClusterNumberIndex += 1
            else:
                Type[i] = -1
    # print(PointClusterNumberIndex)
    return Membership,Type,PointClusterNumberIndex-1

def ExpandCluster(DistanceMatrix,PointtoExpand,Epsilon,Points,PointNeighbors,BorderPoint,Membership,PointClusterNumberIndex,Type,Maximum):
    Neighbors = []
    for i in PointNeighbors:
        if i not in  BorderPoint:
            Membership[i][PointClusterNumberIndex] = round(calculate_Membership(Maximum,DistanceMatrix[i][PointtoExpand],Epsilon),3)
    for i in BorderPoint:
        for j in Neighbors:
            if (j not in BorderPoint) and (PointClusterNumberIndex not in Membership[j]):
                Membership[j][PointClusterNumberIndex] = round(calculate_Membership(Maximum,DistanceMatrix[j][PointtoExpand],Epsilon),3)
            elif j not in BorderPoint:
                Membership[j][PointClusterNumberIndex] = round(max(Membership[j][PointClusterNumberIndex],calculate_Membership(Maximum,Distanc

        Neighbors = numpy.where(DistanceMatrix[i]<=Epsilon)[0]
        Neighbors = set2List(Neighbors)

        Maximum = Max_Distance(i,Neighbors,DistanceMatrix)
```

Line 11, Column 34 Spaces: 4 Python

File Edit Selection Find View Goto Tools Project Preferences Help

Fuzzy_DBSCAN_1.py | new_algo.py | Corresponding_values_of_dataset.txt | graph_plot.py

```python

        Maximum = Max_Distance(i,Neighbors,DistanceMatrix)

        for k in Neighbors:
            if DistanceMatrix[i][k] == Maximum and k != PointtoExpand:
                try:
                    BorderPoint.index(k)
                except ValueError:
                    BorderPoint.append(k)
        if len(Neighbors) >= Points:
            Membership[i][PointClusterNumberIndex] = 1
            Type[i] = 1
        else:
            Membership[i][PointClusterNumberIndex] = 0
            Type[i] = 0
        PointtoExpand = i
    return

def  calculate_PC(Data_len,cluster_size,Membership_value):
    pc  =  0
    for  i  in  range(Data_len):
        for j in range(cluster_size):
            if (j+1) in Membership_value[i]:
                pc  +=  Membership_value[i][j+1]**2
    return  pc/Data_len

def  calculate_FPI(Data_len,cluster_size,Membership_value):
    value  =  0
    for  i  in  range(Data_len):
        for j in range(cluster_size):
            if (j+1) in Membership_value[i]:
                value  +=  (Membership_value[i][j+1]**2)/Data_len
    if cluster_size == 1:
        return 0
    else:
        FPI  =  1  -  (cluster_size/(cluster_size-1))*(1-value)
        return  FPI

def  F_recall(Output_Membership_final,k):
```

Line 11, Column 34 Spaces: 4 Python

File Edit Selection Find View Goto Tools Project Preferences Help

Fuzzy_DBSCAN_1.py | new_algo.py | Corresponding_values_of_dataset.txt | graph_plot.py

```python
def  F_recall(Output,Membership_final,k):
    membership_sum = 0
    c = 0
    for i in  range(len(Membership_final)):
        if k in Membership_final[i] and Output[i] == k:
            membership_sum += Membership_final[i][k]
    for i in Output:
        if k == i:
            c += 1
    if membership_sum == 0:
        return 0,c
    return membership_sum/c,c

def  F_Precision(Membership_final,k):
    membership_sum = 0
    c = 0
    for i in  range(len(Membership_final)):
        if k in Membership_final[i] and Output[i] == k:
            membership_sum += Membership_final[i][k]
    for i in Membership_final:
        if k in i:
            c += 1
    if c == 0:
        return 0,c
    return membership_sum/c,c

def Combine_clusters(Membership_value,size,Points):
    cluster_size = size
    m = 1
    l = 1
    for j in range(1,size+1):
        for k in range(1,j):
            count = 0
            flag = 0
            for i in Membership_value:
                if (j in i) and (k in i):
                    count += 1
                if count == Points:
                    flag = 1
```

Line 11, Column 34 Spaces: 4 Python

File Edit Selection Find View Goto Tools Project Preferences Help

Fuzzy_DBSCAN_1.py | new_algo.py | Corresponding_values_of_dataset.txt | graph_plot.py

```python
            if flag == 1:
                cluster_size -= 1
                l = min(j,k)
                for i in Membership_value:
                    if (j in i) and (k in i):
                        i[l] = max(i[j],i[k])
                        if l!=k:
                            del i[k]
                        if l!=j:
                            del i[j]
                    elif k in i:
                        i[l] = i[k]
                        if l!=k:
                            del i[k]
                    elif j in i:
                        i[l] = i[j]
                        if l!=j:
                            del i[j]

    return Membership_value,cluster_size
def membership_set(Membership,r1):
    set_value = []
    for i in range(len(Membership)):
        for j in range(r1+1):
            if j in Membership[i]:
                set_value.append(j)
    return set_value

def mapping_cluster(Membership,set_result,Output,set_output):
    count = {}
    Result = []
    c =0
    for i in range(len(Membership)):
        Result.append(0)
    for i in set_result:
        # for m in set_output:
        #         count[m] = 0
        max1 = 0
        pos = 0
```

Line 11, Column 34 Spaces: 4 Python

File Edit Selection Find View Goto Tools Project Preferences Help

Fuzzy_DBSCAN_1.py | new_algo.py | Corresponding_values_of_dataset.txt | graph_plot.py

```python
    for i in range(len(Membership)):
        Result.append(0)
    for i in set_result:
        # for m in set_output:
        #         count[m] = 0
        max1 = 0
        pos = 0
        for j in set_output:
            c = 0
            for k in range(len(Membership)):
                if i in Membership[k] and j == Output[k]:
                    c += 1
            if c>max1:
                max1 = c
                pos = j
        for j in range(len(Membership)):
            if i in Membership[j]:
                Result[j] = pos
                Membership[j][pos] = Membership[j][i]
                if pos != i:
                    del Membership[j][i]
    return Membership,Result

def accuracy(Membership,Output):
    count = 0
    for i in range(len(Output)):
        if Output[i] in Membership[i]:
            count+=1
    return count/len(Output)
def Plot_cluster(Membership,Data,Output,file_name):
    m = max(Output)
    color = {1:'red',2:'blue',3:'yellow',4:'brown',5:'green',6:'orange',7:'cyan',8:'olive',9:'gray',10:'pink',11:'lime',12:'blueviolet',13:'g
    plt.style.use('ggplot')
    for i in range(len(Output)):
        if Output[i] in Membership[i]:
            plt.scatter(Data[i][0],Data[i][1],color = color[Output[i]])
        else:
            plt.scatter(Data[i][0],Data[i][1],color ='black')
    plt.title(file_name)
```

Line 11, Column 34 Spaces: 4 Python

File Edit Selection Find View Goto Tools Project Preferences Help

Fuzzy_DBSCAN_1.py | new_algo.py | Corresponding_values_of_dataset.txt | graph_plot.py

```python
    plt.title(file_name)
    plt.show()


file_name = "compound"
X  =  numpy.loadtxt(file_name+".txt",dtype  =  'float')
Data  =  X[:,  :-1]
Output  =  X[: ,-1]
Target = sequence_cluster(Output)
Epsilon = 1.6
Points = 3
t0 = time.time()
Membership_value,Type,cluster_len=DBSCAN(Data,Epsilon,Points)
Membership,c_len1 = Combine_clusters(Membership_value,cluster_len,Points)

Membership,c_len2 = Combine_clusters(Membership,cluster_len,Points)
t1 = time.time()

set_value = membership_set(Membership,cluster_len)
Membership_final,Result = mapping_cluster(Membership,set_value,Output,list(set(Output)))

# for i in range(len(Membership_final)):
#     print(Membership_final[i],Output[i],"\n")

PC = calculate_PC(len(Data),cluster_len,Membership_final)
FPI = calculate_FPI(len(Data),cluster_len,Membership_final)

F_measure  =  0
l  =  list(set(Output))
for i in l:
    try:
        f_Precision,Cj = F_Precision(Membership_final,i)
        f_recall,Dc = F_recall(Output,Membership_final,i)
        f_measure = 2*(f_Precision*f_recall)/(f_recall+f_Precision)
        F_measure += (Cj/len(Data))*f_measure
    except:
        F_measure +=0
Accuracy =  accuracy(Membership,Output)
```

Line 11, Column 34 Spaces: 4 Python

File Edit Selection Find View Goto Tools Project Preferences Help

Fuzzy_DBSCAN_1.py | new_algo.py | Corresponding_values_of_dataset.txt | graph_plot.py

```python
PC = calculate_PC(Len(Data),cluster_len,Membership_final)
FPI = calculate_FPI(Len(Data),cluster_len,Membership_final)

F_measure  =  0
l  =  list(set(Output))
for i in l:
    try:
        f_Precision,Cj = F_Precision(Membership_final,i)
        f_recall,Dc = F_recall(Output,Membership_final,i)
        f_measure = 2*(f_Precision*f_recall)/(f_recall+f_Precision)
        F_measure += (Cj/Len(Data))*f_measure
    except:
        F_measure +=0
Accuracy =  accuracy(Membership,Output)
k = list(set(Output))
print("File_name:  ",file_name,"\n","Accuracy: " ,Accuracy,"\n","PC:  ",PC,"\n","FPI:  ",FPI,"\n","F_measure:  ",F_measure,"\n","Time_complexi
Plot_cluster(Membership,Data,Output,file_name)
# if F_measure>max_F_measure:
#     max_F_measure = F_measure
#     q1 = Epsilon
#     q2 = Points
#     if PC>max_Pc:
#         max_Pc = PC
#         if FPI>max_FPI:
#             max_FPI = FPI

# count+=1
# # print(count)
# Points += 1
# Epsilon += .1



# max_F_measure,max_Pc,max_FPI = 0,0,0
# count = 0
# Epsilon = .7
# for z in range(3):
#     Points = 6
```

Line 11, Column 34 Spaces: 4 Python

File Edit Selection Find View Goto Tools Project Preferences Help

Fuzzy_DBSCAN_1.py | new_algo.py | Corresponding_values_of_dataset.txt | graph_plot.py

```python
# for z in range(3):
#     Points = 6
#     for x in range(6):
#         Membership_value,Type,cluster_len=DBSCAN(Data,Epsilon,Points)
#         Membership,c_len1 = Combine_clusters(Membership_value,cluster_len,Points)

#         Membership,c_len2 = Combine_clusters(Membership,cluster_len,Points)

#         set_value = membership_set(Membership,cluster_len)
#         Membership_final,Result = mapping_cluster(Membership,set_value,Output,list(set(Output)))

#         # for i in range(len(Membership_final)):
#         #     print(Membership_final[i],Output[i],"\n")

#         PC = calculate_PC(len(Data),cluster_len,Membership_final)
#         FPI = calculate_FPI(len(Data),cluster_len,Membership_final)

#         F_measure  =  0
#         l  =  list(set(Output))
#         for i in l:
#             try:
#                 f_Precision,Cj = F_Precision(Membership_final,i)
#                 f_recall,Dc = F_recall(Output,Membership_final,i)
#                 f_measure = 2*(f_Precision*f_recall)/(f_recall+f_Precision)
#                 F_measure += (Cj/len(Data))*f_measure
#             except:
#                 F_measure +=0
#         k = list(set(Output))
#         c_len = c_len1-(cluster_len-c_len2)
#         # print(c_len,PC,FPI,F_measure)
#         if F_measure>max_F_measure:
#             max_F_measure = F_measure
#             q1 = Epsilon
#             q2 = Points
#             if PC>max_Pc:
#                 max_Pc = PC
#                 if FPI>max_FPI:
#                     max_FPI = FPI

```

Line 11, Column 34 Spaces: 4 Python

File Edit Selection Find View Goto Tools Project Preferences Help

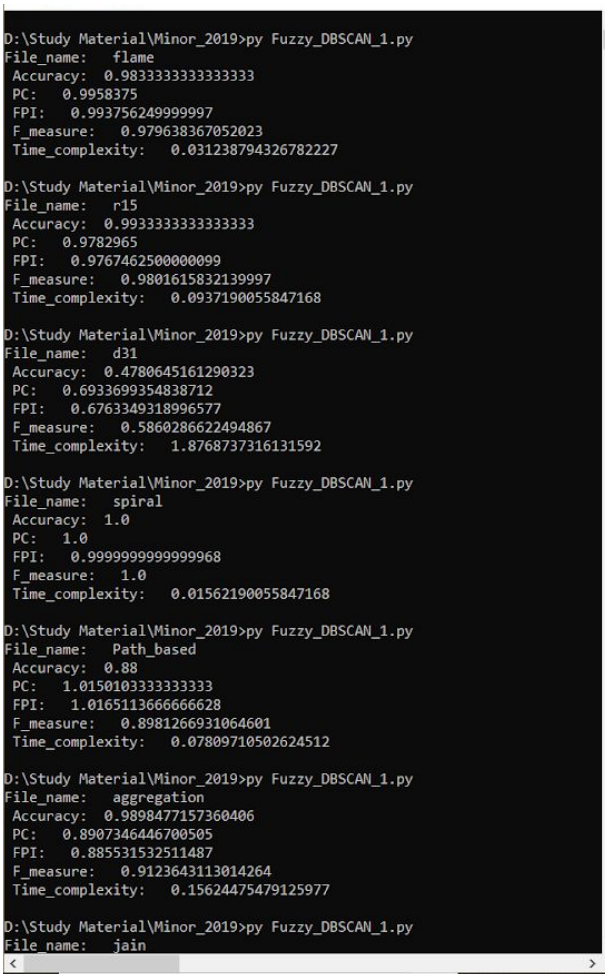
Fuzzy_DBSCAN_1.py | new_algo.py | Corresponding_values_of_dataset.txt | graph_plot.py

```python
#                 f_measure = 2*(f_Precision*f_recall)/(f_recall+f_Precision)
#                 F_measure += (Cj/len(Data))*f_measure
#             except:
#                 F_measure +=0
#         k = list(set(Output))
#         c_len = c_len1-(cluster_len-c_len2)
#         # print(c_len,PC,FPI,F_measure)
#         if F_measure>max_F_measure:
#             max_F_measure = F_measure
#             q1 = Epsilon
#             q2 = Points
#             if PC>max_Pc:
#                 max_Pc = PC
#                 if FPI>max_FPI:
#                     max_FPI = FPI
```


```python

#         count+=1
#         # print(count)
#         Points += 1
#     Epsilon += .1

# print("\n\n\n\n")
# print(max_Pc,max_FPI,max_F_measure,q1,q2)

```

Line 11, Column 34 Spaces: 4 Python

Snapshots of output



```
D:\Study Material\Minor_2019>py Fuzzy_DBSCAN_1.py
File_name:   flame
 Accuracy:  0.9833333333333333
 PC:   0.9958375
 FPI:   0.993756249999997
 F_measure:   0.979638367052023
 Time_complexity:   0.031238794326782227

D:\Study Material\Minor_2019>py Fuzzy_DBSCAN_1.py
File_name:   r15
 Accuracy:  0.9933333333333333
 PC:   0.9782965
 FPI:   0.9767462500000099
 F_measure:   0.9801615832139997
 Time_complexity:   0.0937190055847168

D:\Study Material\Minor_2019>py Fuzzy_DBSCAN_1.py
File_name:   d31
 Accuracy:  0.4780645161290323
 PC:   0.6933699354838712
 FPI:   0.6763349318996577
 F_measure:   0.5860286622494867
 Time_complexity:   1.8768737316131592

D:\Study Material\Minor_2019>py Fuzzy_DBSCAN_1.py
File_name:   spiral
 Accuracy:  1.0
 PC:   1.0
 FPI:   0.9999999999999968
 F_measure:   1.0
 Time_complexity:   0.01562190055847168

D:\Study Material\Minor_2019>py Fuzzy_DBSCAN_1.py
File_name:   Path_based
 Accuracy:  0.88
 PC:   1.0150103333333333
 FPI:   1.0165113666666628
 F_measure:   0.8981266931064601
 Time_complexity:   0.07809710502624512

D:\Study Material\Minor_2019>py Fuzzy_DBSCAN_1.py
File_name:   aggregation
 Accuracy:  0.9898477157360406
 PC:   0.8907346446700505
 FPI:   0.885531532511487
 F_measure:   0.9123643113014264
 Time_complexity:   0.15624475479125977

D:\Study Material\Minor_2019>py Fuzzy_DBSCAN_1.py
File_name:   jain
```

```
D:\Study Material\Minor_2019>py Fuzzy_DBSCAN_1.py
File_name:   spiral
 Accuracy:  1.0
 PC:   1.0
 FPI:   0.9999999999999968
 F_measure:   1.0
 Time_complexity:   0.01562190055847168

D:\Study Material\Minor_2019>py Fuzzy_DBSCAN_1.py
File_name:   Path_based
 Accuracy:  0.88
 PC:   1.0150103333333333
 FPI:   1.0165113666666628
 F_measure:   0.8981266931064601
 Time_complexity:   0.07809710502624512

D:\Study Material\Minor_2019>py Fuzzy_DBSCAN_1.py
File_name:   aggregation
 Accuracy:  0.9898477157360406
 PC:   0.8907346446700505
 FPI:   0.885531532511487
 F_measure:   0.9123643113014264
 Time_complexity:   0.15624475479125977

D:\Study Material\Minor_2019>py Fuzzy_DBSCAN_1.py
File_name:   jain
 Accuracy:  1.0
 PC:   1.0001297587131368
 FPI:   1.0001730116175163
 F_measure:   1.0013336083041178
 Time_complexity:   0.10934877395629883

D:\Study Material\Minor_2019>py Fuzzy_DBSCAN_1.py
File_name:   compound
 Accuracy:  0.8320802005012531
 PC:   0.8913428571428572
 FPI:   0.869611428571418
 F_measure:   0.8670047247662973
 Time_complexity:   0.08754515647888184

D:\Study Material\Minor_2019>
```

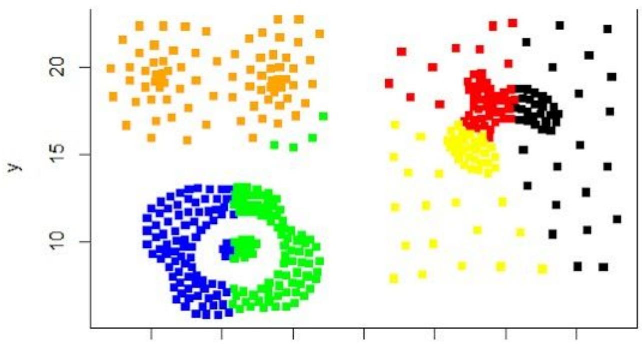
```
D:\Study Material\Minor_2019>py new_algo.py
File_name:   flame
 Accuracy:  0.9958333333333333
 PC:   1.0697070916666662
 FPI:   1.0813249402777776
 F_measure:   1.053621280332056
 Time_complexity:   0.0

D:\Study Material\Minor_2019>py new_algo.py
File_name:   r15
 Accuracy:  0.9933333333333333
 PC:   0.7894522533333326
 FPI:   0.7848751284057983
 F_measure:   0.8784267334308512
 Time_complexity:   0.11121797561645508

D:\Study Material\Minor_2019>py new_algo.py
File_name:   d31
 Accuracy:  0.8070967741935484
 PC:   0.7232166567741921
 FPI:   0.7216350376700513
 F_measure:   0.7221397888327192
 Time_complexity:   6.931645631790161

D:\Study Material\Minor_2019>py new_algo.py
File_name:   spiral
 Accuracy:  1.0
 PC:   0.9485248878205125
 FPI:   0.9474953855769223
 F_measure:   0.973028846153846
 Time_complexity:   0.062483787536621094

D:\Study Material\Minor_2019>py new_algo.py
File_name:   Path_based
 Accuracy:  0.98
 PC:   0.8879432566666666
 FPI:   0.8839412301190477
 F_measure:   0.9276866519936596
 Time_complexity:   0.03124260902404785

D:\Study Material\Minor_2019>py new_algo.py
File_name:   aggregation
 Accuracy:  1.0
 PC:   0.9643724302030453
 FPI:   0.9632590686468936
 F_measure:   0.9868304175963555
 Time_complexity:   0.09372854232788086

D:\Study Material\Minor_2019>py new_algo.py
File_name:   jain
```

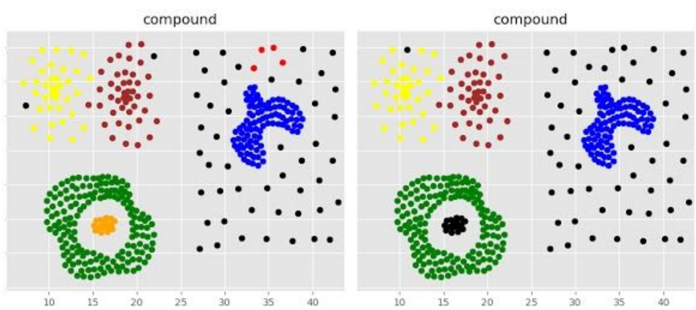
```
D:\Study Material\Minor_2019>py new_algo.py
File_name:   Path_based
 Accuracy:  0.98
 PC:   0.8879432566666666
 FPI:   0.8839412301190477
 F_measure:   0.9276866519936596
 Time_complexity:   0.03124260902404785

D:\Study Material\Minor_2019>py new_algo.py
File_name:   aggregation
 Accuracy:  1.0
 PC:   0.9643724302030453
 FPI:   0.9632590686468936
 F_measure:   0.9868304175963555
 Time_complexity:   0.09372854232788086

D:\Study Material\Minor_2019>py new_algo.py
File_name:   jain
 Accuracy:  0.9919571045576407
 PC:   0.8763220965147454
 FPI:   0.8732301489276139
 F_measure:   0.9263449180972172
 Time_complexity:   0.04687047004699707

D:\Study Material\Minor_2019>py new_algo.py
File_name:   compound
 Accuracy:  0.8345864661654135
 PC:   0.7328929323308272
 FPI:   0.7268223171565271
 F_measure:   0.7813921011125773
 Time_complexity:   0.07811570167541504

D:\Study Material\Minor_2019>py new_algo.py
File_name:   compound
 Accuracy:  0.8796992481203008
 PC:   0.6540726466165414
 FPI:   0.6460278244448323
 F_measure:   0.7374061764177082
 Time_complexity:   0.0624849796295166

D:\Study Material\Minor_2019>
```

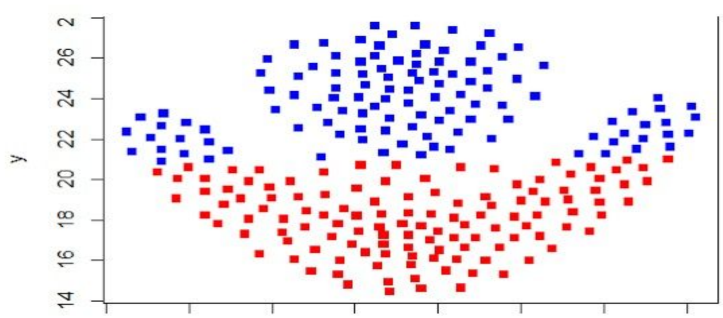
Testing

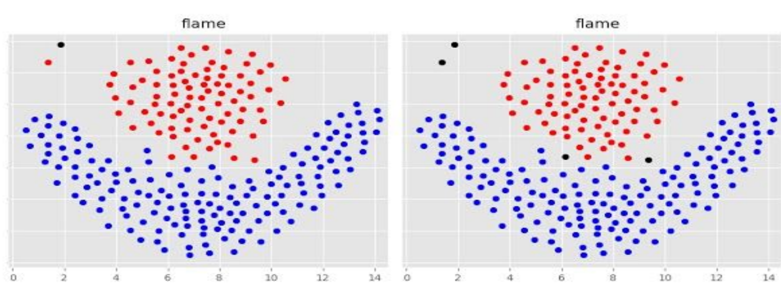
We have tested our proposed algorithm on 8 different data set and results have been mentioned above. Very clearly our proposed algorithm has performed better in terms of time complexity and with respect to accuracy when compared to FCM or soft DBScan.

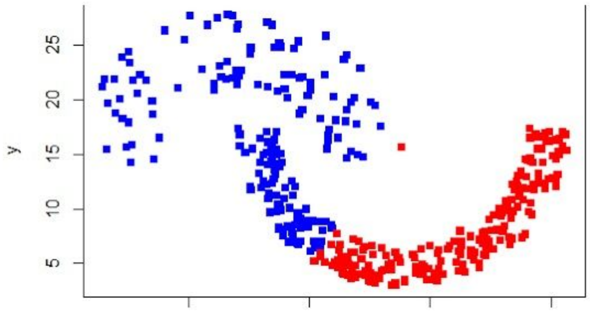
The comparison of our modified algorithm with Fuzzy C mean and Fuzzy DBScan(algo 4.3)

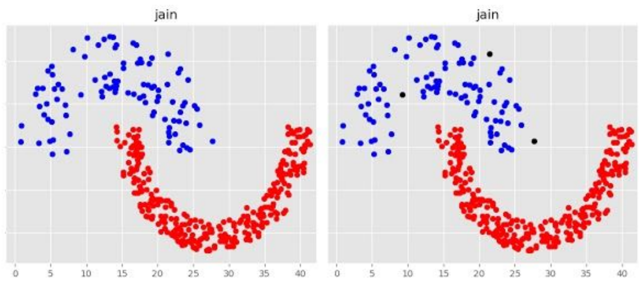


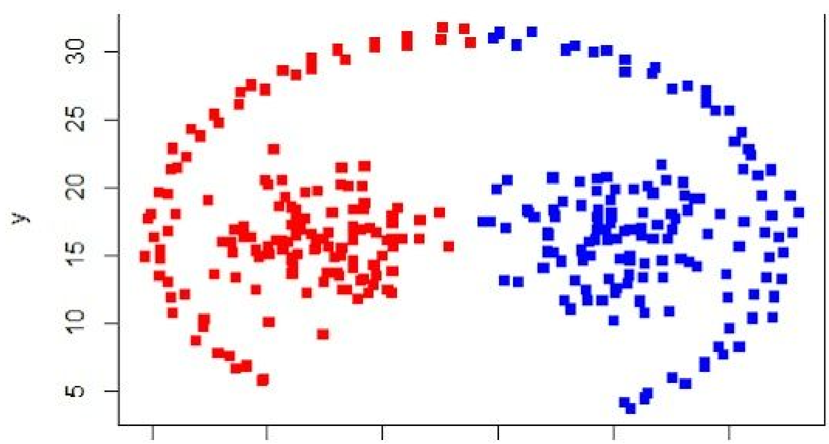


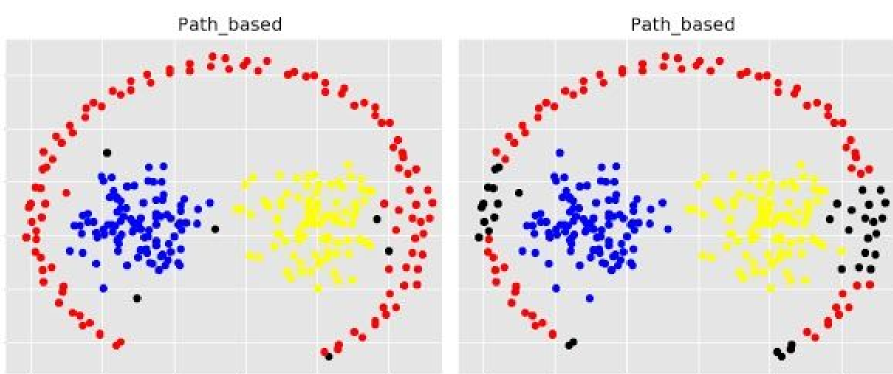


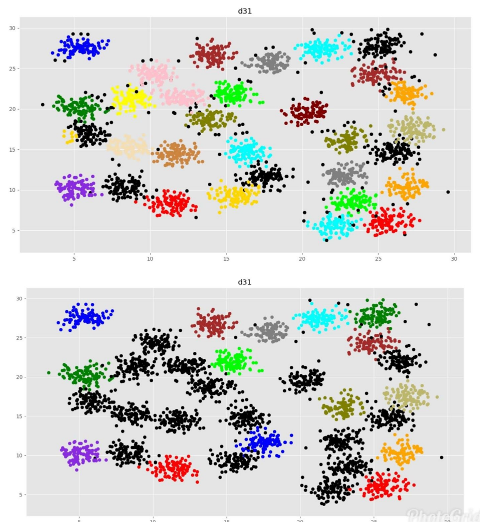


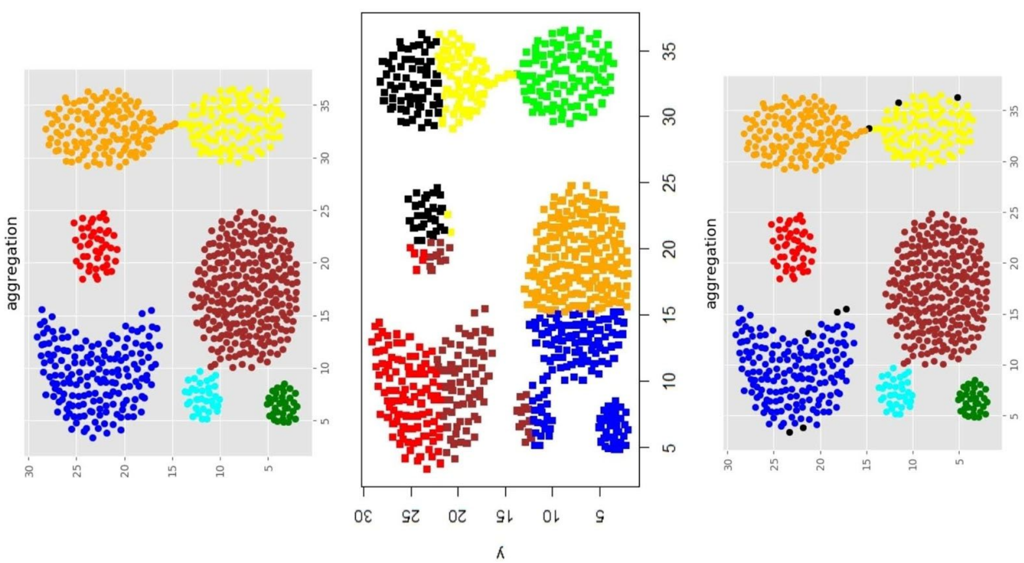

y
25
20
15
10
5

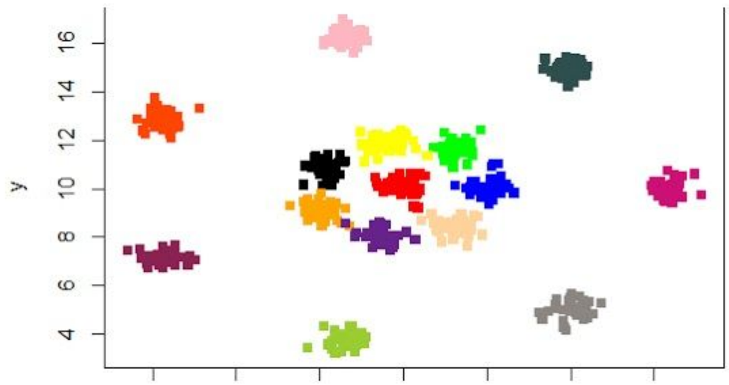

jain
jain
0 5 10 15 20 25 30 35 40
0 5 10 15 20 25 30 35 40

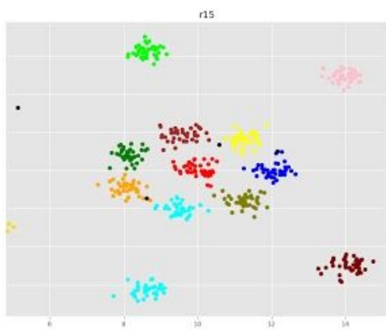

y
30
25
20
15
10
5

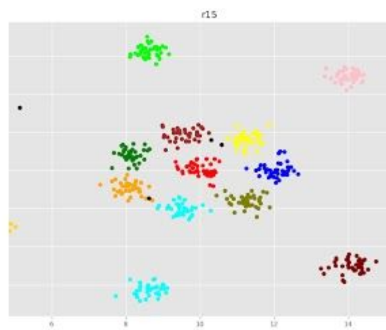

Path_based
Path_based
5 10 15 20 25 30
5 10 15 20 25 30

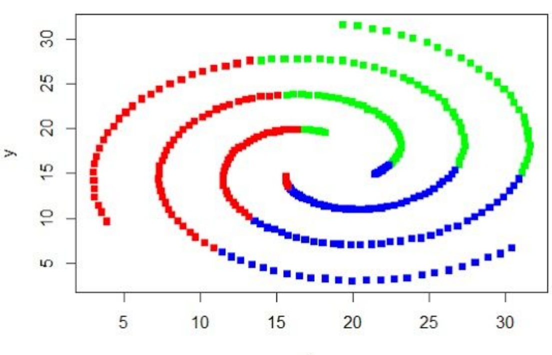

d31
d31

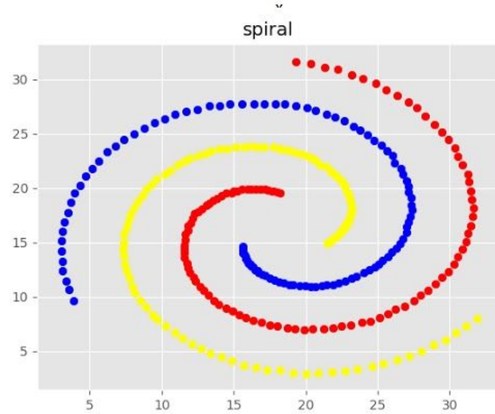

aggregation
y
aggregation

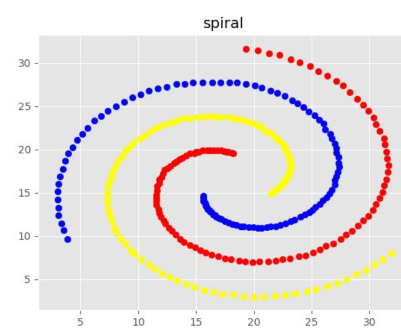

y
4
6
8
10
12
14
16


r15


r15


y
5
10
15
20
25
30
5
10
15
20
25
30


spiral
5
10
15
20
25
30
5
10
15
20
25
30


spiral
5
10
15
20
25
30
5
10
15
20
25
30

References

We have referred a research paper on Fuzzy DBScan algorithm and further proposed our own algorithm. Here is the link to the research paper.

https://link.springer.com/article/10.1007/s00500-016-2435-0

Division of Project

Algorithm 4.1: Naima Farooqi

Algorithm 4.2: Harsh Vishnoi

Algorithm 4.3: Aditya Gupta

New Modified Algorithm: All three