**CLUSTERING**

**A PROJECT REPORT**

***Submitted by***

**PIYUSH ANAND**

***Under the guidance of***

**Dr. JHARESWAR MAITI**

**Industrial and Systems Engineering Department**

**Indian Institute of Technology,**

**Kharagpur-721302**

**[2020]**

**Acknowledgement**

I wholeheartedly thank our project guide, Prof. Jhareswar Maiti for his guidance

and constant supervision during the tenure of fulfillment of our project. I

thank him for making me realize the importance of learning while doing project and

encouraging me to drive ahead.

I would like to thank the department for providing me this opportunity.

I also acknowledge with a deep sense of reverence my gratitude towards my family and parents who have always supported me morally and economically.

At last I would like to express my gratitude to my friends who have helped me in this project.

**CONTENTS**

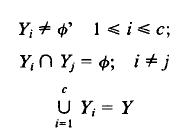
* **INTRODUCTION.**
* **Agglomerative Hierarchical Clustering.**
* **Non-hierarchical joining algorithms ( K-means clustering).**
* **FCM: The FUZZY c-MEANS CLUSTERING ALGORITHM.**
* **A FUZZY K MODES ALGORITHM FOR CLUSTERING CATEGORICAL DATA.**
* **FUZZY CLUSTERING OF CATEGORICAL DATA USING FUZZY CENTROIDS.**
* **INTEGRATED CLUSTERING AND SUPERVISED LEARNING FOR CATEGORICAL DATA.**
* **References.**

INTRODUCTION

Cluster analysis is a class of techniques that are used to classify objects or cases into relative groups called Clusters. The grouping is done with respect to the observations.

To be more specific, Cluster is a group homogenous things. It is also known as segmentation analysis, taxonomy analysis, or clustering.

Let Y be a sample of N observations . If c is an integer,1<c<N, a conventional(or “hard”) c partition of Y is a c-tuple of subsets of Y that satisfies three conditions:



**The principle (Kaufman and Rousseeuw, 2005):**

* Objects in the same group are similar
* Objects in the different groups are as dissimilar as possible

**The Advantages of Cluster Analysis**

* Cluster analysis allows us to identify and define patterns amongst data elements.
* Finding this pattern may help us to distinguish and outline structures which might not be apparent before, but which gave significant meaning.
* After discovering such structures from the dataset, informed decision making becomes more effective and easier.

**Criteria for clustering**

* **Variables to be considered**
* Important variables are to be considered and trivial variables are to be discarded.
* Variables may be of different types based on measurement like nominal, ordinal, interval and ratio.
* **Similarity and dissimilarity measures**
* It is usually a measure of distance between the objects to be clustered

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Objects | A | B | C | D | E |
| A | 0 |  |  |  |  |
| B | dab | 0 |  |  |  |
| C | dac | dbc | 0 |  |  |
| D | dad | dbd | dcd | 0 |  |
| E | dae | dbe | dce | dde | 0 |

dab is the distance between

object A and B

**Distance measures**

* **Euclidean distance**

d( i,j ) ={ (xi1  – xj1 )2 + (xi2 – xj2)2 + ……. + (xip – xjp )2  }1/2

* **Squared Euclidean Distance**

d( i,j ) ={ (xi1  – xj1 )2 + (xi2 – xj2)2 + ……. + (xip – xjp )2  }

* **Manhattan Distance**

d( i,j ) = | (xi1 – xj1 ) | + | (xi2 – xj2) | + …… + | (xip – xjp ) |

* **Minkowski distance**

d( i,j ) = { | (xi1 – xj1 ) |m + | (xi2 – xj2) |m + …… + | (xip – xjp ) |m }1/m

**Clustering Algorithms**

* **Hierarchical joining algorithms**
* This is the most common method of clustering. It creates a series of models with cluster solutions from 1 to n. This approach also works with variables instead of cases.
* This can further be classified into
* **Single**  (nearest neighbour)

Distance between two clusters = distance between two closest neighbour.

* **Complete** ( furthest neighbour)

Distance between two clusters = distance between two distant furthest members.

* **Centroid**

Distance between two clusters = distance between multivariate means of each cluster.

* **Average**

Distance between two clusters = average distance between al the members of the two clusters

* **Median**

Distance between two clusters = median distance between all the members of the two clusters

* **Ward**

Distance between two clusters = average distance between all members of the two clusters with adjustment for covariances.

**Dendrogram**

Dendrogram is a diagram that shows the hierarchical relationship between objects. It is most commonly created as an output from hierarchical clustering.

**Cluster Validation**

Clustering of data objects is a well-established data mining task that aims at grouping these objects. The grouping is made such that *similar* objects are aggregated together in the same group (or cluster) while *dissimilar* ones are grouped in different clusters. In this context, the definition of similarity, and thus the final clustering is highly dependent on the applied distance function between the data objects. Different to classification, clustering does not use a subset of the data objects with known class labels to learn a classification model. As a completely unsupervised task, clustering calculates the similarity between objects without having any information about their correct distribution (also known as the ground truth). The latter fact motivated the research in the field of clustering validation notably more than the field of classification evaluation. It has been even stated that clustering validation is regarded as important as the clustering itself.

There are two types of clustering validation. The *external* validation, which compares the clustering result to a reference result which is considered as the ground truth. If the result is somehow similar to the reference, we regard this final output as a “good” clustering. This validation is straightforward when the similarity between two clustering has been well-defined, however, it has fundamental caveat that the reference result is not provided in most real applications. Therefore, external evaluation is largely used for synthetic data and mostly for tuning clustering algorithms.

*Internal* validation is the other type clustering evaluation, where the evaluation of the clustering is compared only with the result itself, i.e., the structure of found clusters and their relations to each other. This is much more realistic and efficient in many real-world scenarios as it does not refer to any assumed references from outside which is not always feasible to obtain. Particularly, with the huge increase of the data size and dimensionality as in recent applications with streaming data outputs, one can hardly claim that a complete knowledge of the ground truth is available or always valid.

Internal clustering validation is based only on the intrinsic information of the data. Since we can only refer to the input dataset itself, internal validation needs assumptions about a “good” structure of found clusters which are normally given by reference result in external validation. Two main concepts, the compactness and the separation, are the most popular ones. Most other concepts are actually just combinations of variations of these two.

The *Compactness* measures how closely data points are grouped in a cluster. Grouped points in the cluster are supposed to be related to each other, by sharing a common feature which reflects a meaningful pattern in practice. Compactness is normally based on distances between in-cluster points. The very popular way of calculating the compactness is through variance, i.e., average distance to the mean, to estimate how objects are bonded together with its mean as its center. A small variance indicates a high compactness.

The *Separation* measures how different the found clusters are from each other. Users of clustering algorithms are not interested in similar or vague patterns when clusters are not well-separated. A distinct cluster that is far from the others corresponds to a unique pattern. Similar to the compactness, the distances between objects are widely used to measure separation, e.g., pair wise distances between cluster centers, or pair wise minimum distances between objects in different clusters. Separation is an inter-cluster criterion in the sense of relation between clusters.

Obviously, clustering evaluation is a stand-alone process that is not included within clustering task.

Some of the clustering validation indices used are discussed below:

* Elbow method:- A fundamental step for any unsupervised algorithm is to determine the optimal number of clusters into which the data may be clustered. The **Elbow Method** is one of the most popular methods to determine this optimal value of number of clusters (k).

We now define the following:-

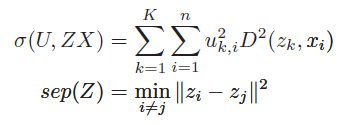
**Distortion:** It is calculated as the average of the squared distances from the cluster centers of the respective clusters. Typically, the Euclidean distance metric is used.

**Inertia:** It is the sum of squared distances of samples to their closest cluster center.

We iterate the values of k and calculate the distortion and inertia for each value of k in the given range.

To determine the optimal number of clusters, we have to select the value of k at the “elbow” i.e. the point after which the distortion/inertia start decreasing in a linear fashion.

* Xie-Beni(XB) Index:- The XB index [24] is defined as a function of the ratio of the total variation *σ* to the minimum separation *sep* of the clusters. Here, *σ* and *sep* can be written as



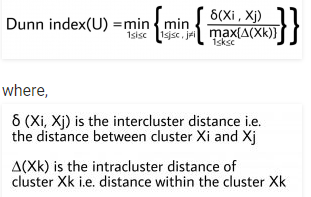
Where ||.|| is the Euclidean norm, and D(zk,xi), as mentioned earlier, is the distance between the pattern xi and the cluster medoid zk. The XB index is then defined as

Image

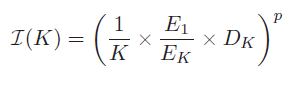
The objective is therefore to minimize the XB index for achieving proper clustering.

* Dunn Index:- The Dunn index (DI) is a metric for evaluating clustering algorithms, is an internal evaluation scheme, where the result is based on the clustered data itself. Like all other such indices, the aim of the Dunn index is to identify sets of clusters that are compact, with a small variance between members of the cluster, and well separated, where the means of different clusters are sufficiently far apart, as compared to the within cluster variance.  
  Higher the Dunn index value, better is the clustering. The number of clusters that maximizes Dunn index is taken as the optimal number of clusters k. It also has some drawbacks. As the number of clusters and dimensionality of the data increase, the computational cost also increases.

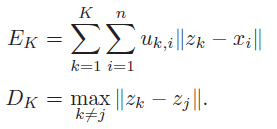
The Dunn index for c number of clusters is defined as :



* *I* index:- A cluster validity index *I* is defined as follows:-



Where K is the number of clusters. Here



The index *I* is a composition of three factors, namely, 1*/K*, *E*1*/EK*, and *DK*. The first factor will try to reduce index *I* as *K* is increased. The second factor consists of the ratio of *E*1, which is constant for a given data set, to *EK*, which decreases with increase in *K*. To compute *E*1, the value of *K* is taken as one, i.e., all the points are considered to be in the same cluster. Note that index *I* increases as *EK* decreases. This, in turn, indicates that formation of more number of clusters, which are compact in nature, would be encouraged. Finally, the third factor *DK* (which measures the maximum separation between two clusters over all possible pairs of clusters) will increase with the value of *K*. However, note that this value is upper bounded by the maximum separation between two points in the data set. Thus, the three factors are found to compete with and balance each other critically. The power *p* is used to control the contrast between the different cluster configurations. In this paper, we have taken *p* = 1.

The objective is therefore to minimize the *I* index for achieving proper clustering.

**Agglomerative Hierarchical Clustering algorithms**

* Step 1: Identify the variable (p) and objects (n)
* Step 2: Collect data ( Xn\*p )
* Step 3: Select similarity or dissimilarity measures
* Step 4: Obtain distance matrix ( Dn\*n )
* Step 5: Start with *n* clusters where each cluster contains a single entity
* Step 6: Find out the nearest pairs of clusters from ( Dn\*n ). Let the distance between most similar clusters *P* and *Q* be *dpq*
* Step 7: Merge clusters P and Q and label the newly formed cluster as (PQ). Update the entries of the matrix D by
* Deleting the rows and columns corresponding to the cluster P and Q
* Adding a row and column giving the distances between cluster (PQ)
* Step 8: Repeat the step 6 and 7 for total ( n-1) times. When all the objects will be in a single cluster the algorithm terminates.
* Step 9: Record the identity of the clusters that are merged and the levels at which the mergers takes place.

**Code for the Hierarchical Clustering**

# Hierarchical Clustering

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Cl1.csv')

X = dataset.iloc[:, :].values

print("Press 1 for Single Linkage")

print("Press 2 for Complete Linkage")

print("Press 3 for Average Linkage")

print("Press 4 for Ward Linkage")

icon=int(input())

if icon==1:

v='single'

elif icon==2:

v='complete'

elif icon==3:

v='average'

else:

v='ward'

# Using the dendrogram to find the optimal number of clusters

import scipy.cluster.hierarchy as sch

dendrogram = sch.dendrogram(sch.linkage(X, method = v))

plt.title('Dendrogram')

plt.xlabel('Customers')

plt.ylabel('Euclidean distances')

plt.show()

if icon==1:

print("Press 1 for Euclidean distance")

print("Press 2 for Manhattan distance")

print("Press 3 for Minkowski distance")

p=int(input())

if p==1:

a='euclidean'

elif p==2:

a='manhattan'

else:

a='minkowski'

elif icon==2:

print("Press 1 for Euclidean distance")

print("Press 2 for Manhattan distance")

print("Press 3 for Minkowski distance")

p=int(input())

if p==1:

a='euclidean'

elif p==2:

a='manhattan'

else:

a='minkowski'

elif icon==3:

print("Press 1 for Euclidean distance")

print("Press 2 for Manhattan distance")

print("Press 3 for Minkowski distance")

p=int(input())

if p==1:

a='euclidean'

elif p==2:

a='manhattan'

else:

a='minkowski'

else:

a='euclidean'

n=int(input("Enter the number of clusters"))

# Fitting Hierarchical Clustering to the dataset

from sklearn.cluster import AgglomerativeClustering

hc = AgglomerativeClustering(n\_clusters = n, affinity = a, linkage = v)

y\_hc = hc.fit\_predict(X)

# Visualising the clusters

jet=plt.get\_cmap('jet')

colors=iter(jet(np.linspace(0,1,n)))

for i in range(n):

plt.scatter(X[y\_hc == i, 0], X[y\_hc == i, 1], s = 100, c = next(colors))

**Non-hierarchical joining algorithms ( K-means clustering)**

* Step 1: Identify the variable (p) and objects (n)
* Step 2: Collect data ( Xn\*p )
* Step 3: Select similarity or dissimilarity measures
* Step 4: Obtain distance matrix ( Dn\*n )
* Step 5: Partition the objects into *K* initial clusters
* Step 6: Proceed through the list of objects, assigning an object to the cluster whose centroid (mean) is the nearest
* Step 7: Recalculate the centroid for the cluster receiving the new object and for the clusters losing the object
* Step 8: Repeat step 6 and 7 until no more reassignments take place

|  |  |  |
| --- | --- | --- |
| Object | X1 | X2 |
| A | 5 | **3** |
| B | -1 | 1 |
| C | 1 | -2 |
| D | -3 | -2 |

For example

* Let the dataset

be

* Arbitarily assign objects to clustes

|  |  |  |
| --- | --- | --- |
| Cluster | X1 | X2 |
| AB | 2 | 2 |
| CD | -1 | -2 |

* Find out the distance between Clusters and the objects and reassign the cluster to the object on the basis of the distance between them. For example, object B is less away from CD compared to AB. B will come under cluster CD.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cluster | A | B | C | D |
| AB | 10 | 10 | 17 | 45 |
| CD | 61 | 9 | 4 | 4 |
| Status | AB | CD | CD | CD |

|  |  |  |
| --- | --- | --- |
| Cluster | X1 | X2 |
| A | 5 | 3 |
| BCD | -1 | -1 |

* Update the Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Object | A | B | C | D |
| A | 0 | 40 | 41 | 89 |
| BCD | 52 | 4 | 5 | 5 |
| Status | A | BCD | BCD | BCD |

* Again compare the distance between the Objects and the new set of Clusters. If the objects are already assigned to the nearest cluster then the number of clusters formed is final.

**Code for the K means**

# K-Means Clustering

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Cl1.csv')

X = dataset.iloc[:, :].values

n=np.shape(X)[0]

a=n//2

# Using the elbow method to find the optimal number of clusters

from sklearn.cluster import KMeans

wcss = []

for i in range(1, a):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 42)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

plt.plot(range(1, a), wcss)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()

n=int(input("Enter the number of clusters"))

# Fitting K-Means to the dataset

kmeans = KMeans(n\_clusters = n, init = 'k-means++', random\_state = 42)

y\_kmeans = kmeans.fit\_predict(X)

print(kmeans.cluster\_centers\_)

# Visualising the clusters

jet=plt.get\_cmap('jet')

colors=iter(jet(np.linspace(0,1,n)))

for i in range(n):

plt.scatter(X[y\_kmeans == i, 0], X[y\_kmeans == i, 1], s = 100, c = next(colors))

plt.scatter(kmeans.cluster\_centers\_[i:, 0], kmeans.cluster\_centers\_[i:, 1], s = 300, c = 'yellow')

**FCM: The Fuzzy c-Means Clustering Algorithm**

**Limitations of k means:**

A significant fact about k means algorithm is the defect in the belief that each point in the dataset is unequivocally grouped with other members of “its” cluster, and thus bears no similarity or resemblance to other members of dataset.

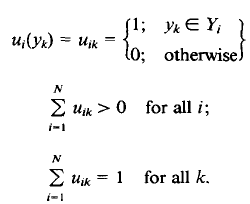
Problems in such cluster algorithms are caused by outliers. Aberrant points may be real outliers or noise due to measurement errors.

The existence of this dilemma led the analysts to consider fuzzy clustering methods as an adjunct procedure which might circumvent the problems caused by data of this type.

**Fuzzy Clustering:**

The FCM algorithms are best described by recasting conditions(equation 1) in matrix-theoretic terms.

Let U be a real c ᵡ N matrix, U=[uik]. U is the matrix representation of the partition of dataset in equation(1) in the situation



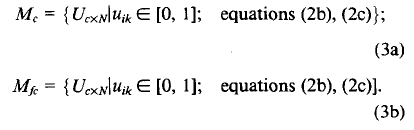
In equation(2) U is a function of dataset(Y) . U is also termed as hard c-partition of Y.

Generalizing this idea we have U as a fuzzy c-partition of Y when the elements of U are numbers in the unit interval[0,1] that continues to satisfy equations(2b) and (2c).

From the present discussion it should be noted that hard partitions of Y are a special type of fuzzy ones wherin each data point is grouped with its intracluster neighbors.

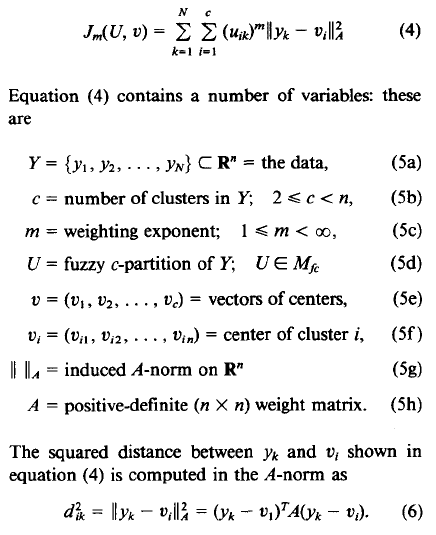
This is a hard requirement for datasets which contain mixtures, hybrids ,outliers etc.

Accordingly, the fuzzy set and in turn fuzzy partition were introduced as a means for clustering and classification models with the aim of accommodating this need. By this device, a point may belong entirely to a single cluster but in general is able to enjoy partial membership in several fuzzy clusters. We denote the sets of all hard and fuzzy c-partition of Y by:



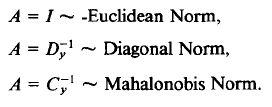
Several clustering criteria have been proposed to identify optimal fuzzy c partitions in Y. The two methods used here are:

* Elbow method
* Generalized least-squared errors function.



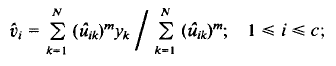
Two of the parameters of Jm require further discussion, namely m and A. Weighting exponent m controls the relative weights placed on each of the squared errors. For most data, 1.5≤m≤3.0 gives good results.

The other parameter is weight matrix A. This matrix controls the shape that optimal clusters assume in Rn. The norms of greatest interest for use in equation(4) are:



Fuzzy c-means (FCM) clustering Algorithms

* Fix c, m, A, ||k||A. Choose an initial matrix U(o)€ Mfc.
* Compute means v̂̂(k), i=1,2,…c, with given equation:



* Compute an updated distance matrix with the given equation:

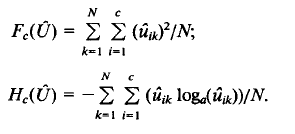
Image

* Compute an updated membership matrix Ũk+1=[ũk+1] with the given equation:

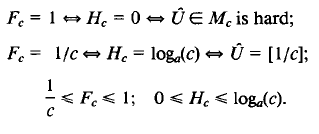
Image

* Compare Ũk+1 to Ũk in any convenient matrix norm. If ||Ũk+1-Ũk||<€, stop. Otherwise, set Ũk = Ũk+1 and return to step 2.

Along with Jm several other cluster validity functional have been calculated on Ũ which are partition coefficient and entropy of Ũ€ Mfc:



Logarithmic bases a€(1,ꝏ). Properties of Fc and Hc utilized for validity checks are:



The FCM algorithm listed below calculates F, H, and (1-F), the latter quantity owing to the inequality (1-F)<H for Ũ≠ Mc.

Proposed Improvement:

* Introduce the concept of elbow method to decide the optimal number of clusters for given dataset.
* To improve the result further, the clustering method is integrated with SVM. A fraction of data points are selected from different cluster based on their proximity to the respective centers and are used to train the SVM. The remaining points are assigned using trained classifier.

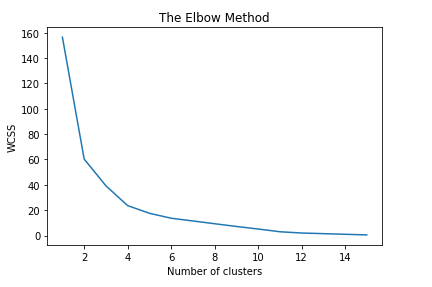
Example :-

Using Euclidean Norm

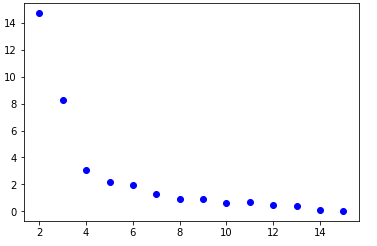
DATASET

|  |  |  |
| --- | --- | --- |
| SI NO | yk1 | yk2 |
| 1 | 0 | 4 |
| 2 | 0 | 3 |
| 3 | 1 | 5 |
| 4 | 2 | 4 |
| 5 | 3 | 3 |
| 6 | 2 | 2 |
| 7 | 2 | 1 |
| 8 | 1 | 0 |
| 9 | 5 | 5 |
| 10 | 6 | 5 |
| 11 | 7 | 6 |
| 12 | 5 | 3 |
| 13 | 7 | 3 |
| 14 | 6 | 2 |
| 15 | 6 | 1 |
| 16 | 8 | 1 |

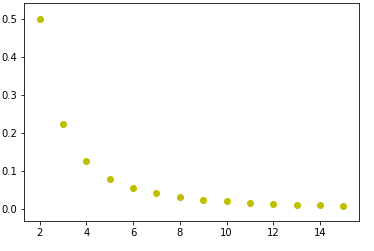
For the above data if we apply elbow method we get the following results



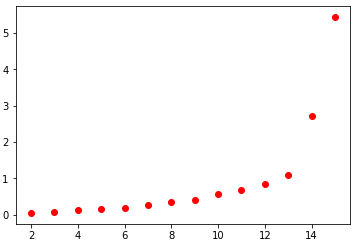
XB index:



*I* index:



Dunn index:



If we select number of clusters as 4, then we get the following result.

Membership matrix:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 0 | 0.97122 | 0.848 | 0.963 | 0.9 | 0.356 | 0.049 | 0.001 | 0.05 | 0.0135 | 6.33e-05 | 0.01079 | 0.057219 | 0.00889 | 0.001344 | .00571499 | 0.01211 |
| 1 | 0.022 | 0.1335 | 0.0211 | 0.0656 | 0.44844 | 0.938 | 0.99827 | 0.922 | 0.00864526 | 5.01e-05 | 0.00825 | 0.086 | 0.0124 | 0.00247 | 0.01516 | 0.022 |
| 2 | 0.00275 | 0.00856 | 0.0051 | 0.0124 | 0.0955 | 0.007 | 0.0003964 | 0.017336 | 0.0232 | 0.000254 | 0.0368 | 0.553617 | 0.846643 | 0.989436 | 0.09634 | 0.9136 |
| 3 | 0.00394 | 0.009764 | 0.01 | 0.0218 | 0.0995 | 0.0051 | 0.0002 | 0.0096 | 0.954513 | 0.999632 | 0.944132 | 0.303 | 0.131 | 0.00647 | 0.0156 | 0.051 |

Jm=20.3023

Fc=0.824656

Hc=0.360385

Cluster centres are:

|  |  |
| --- | --- |
| 0.881574 | 3.99039 |
| 1.79235 | 1.21169 |
| 6.51996 | 1.83679 |
| 5.94196 | 5.16723 |

Results using SVM classifier is:

|  |  |
| --- | --- |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 1 |
| 7 | 1 |
| 8 | 1 |
| 9 | 3 |
| 10 | 3 |
| 11 | 3 |
| 12 | 2 |
| 13 | 2 |
| 14 | 2 |
| 15 | 2 |
| 16 | 2 |

**Code for fuzzy c-means clustering algorithm**

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Data.csv')

X = dataset.iloc[:,0:2].values

a=np.shape(X)

ns=a[0] #number of records

nd=a[1] #number of features

cluster=[] # A list to include all clusters

Update=[] # A list to include all partion martix

IKK=[]

XBB=[]

Dunnn=[]

A=np.zeros([nd,nd],dtype=float)

B=np.zeros([nd,nd],dtype=float)

# Calculatin A norm

icon=int(input("Press 1 for Euclidean Norm, 2 for Diagonal Norm, 3 for Mahalonobis Norm\n"))

if (icon==1):

for i in range(nd):

for j in range(nd):

if(i==j):

A[i][j]=1

else:

A[i][j]=0

elif (icon==2):

cy=np.sum(X,axis=0,keepdims=True)

cy=cy/ns

Cy=np.zeros([nd,nd],dtype=float)

for i2 in range(ns):

b=np.dot(np.transpose(X[i2,:]-cy),X[i2,:]-cy)

Cy=np.add(Cy,b)

W=np.linalg.eigvals(Cy)

for i in range(nd):

for j in range(nd):

if(i==j):

B[i][j]=W[i]

else:

B[i][j]=0

A=np.linalg.inv(B)

else:

cy=np.sum(X,axis=0,keepdims=True)

cy=cy/ns

Cy=np.zeros([nd,nd],dtype=float)

for i2 in range(ns):

b=np.dot(np.transpose(X[i2,:]-cy),X[i2,:]-cy)

Cy=np.add(Cy,b)

A=np.linalg.inv(Cy)

n=np.shape(X)[0]

# Using the elbow method to find the optimal number of clusters

from sklearn.cluster import KMeans

wcss = []

for i in range(1, n):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 42)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

plt.plot(range(1, n), wcss)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()

m=float(input("Enter The value of weight exponent m between 1.5 and 3"))

kbegin=2

kcease=int(input("Enter the total number of cluster"))

eps=0.01 #Ending Criteria

lmax=50 #Ending Criteria

jm=np.zeros([kcease-1],dtype=float) #Cost Function

fc=np.zeros([kcease-1],dtype=float) #Partition Coefficient

hc=np.zeros([kcease-1],dtype=float) #Entropy

dif=np.zeros([kcease-1],dtype=float) #1-fc

for zz in range(2,kcease+1):

c=zz #Number of cluster

print(c)

U=np.random.rand(c,ns) # Fuzzy matrix

b=np.sum(U,axis=0,keepdims=True)

U=U/b #adjusting matrices to satisfy equation 2

v=np.zeros([c,nd],dtype=float) #Initial centres

for i in range(c): #Calculating Initial clusters

aa=0

for j in range(ns):

aa+=(U[i][j]\*\*m)

v[i]=v[i]+((U[i][j]\*\*m)\*X[j])

v[i]=v[i]/aa

d=np.zeros([c,ns],dtype=float) #calculating initial distance matrix

for i in range(c):

p=(X-v[i])

q=np.dot(p,A)

ll=np.dot(q,np.transpose(p))

for j in range(ns):

d[i][j]=ll[j][j]

for mm in range(lmax): #calculating clusters

U2=np.zeros([c,ns],dtype=float) #Calculating Updated U matrix

for i in range(c):

for k in range(ns):

aa=0

for j in range(c):

aa+=((d[i][k]/d[j][k])\*\*(1/(m-1)))

U2[i][k]=1/aa

if(np.max(U2-U)<=eps): #checking condition

break

else:

U=U2

for i in range(c):

aa=0

v[i]=0 #Updating v matrix

for j in range(ns):

aa+=(U[i][j]\*\*m)

v[i]=v[i]+((U[i][j]\*\*m)\*X[j])

v[i]=v[i]/aa

for i in range(c): #Updating distance matrix

p=(X-v[i])

q=np.dot(p,A)

ll=np.dot(q,np.transpose(p))

for j in range(ns):

d[i][j]=ll[j][j]

Update.append(U2)

cluster.append(v) # Storing Clusters

for k in range(c):

for i in range(ns):

jm[zz-2]=jm[zz-2]+(U[k][i]\*\*m)\*d[k][i]

fc[zz-2]=fc[zz-2]+((U[k][i]\*\*2)/ns)

hc[zz-2]=hc[zz-2]+(((U[k][i]\*np.log(U[k][i]))/ns)\*(-1))

dif[zz-2]=1-fc[zz-2]

U=U2

cd=np.zeros([c,c],dtype=float) #getting cluster distance matrix

for i in range(c):

for j in range(c):

for k in range(nd):

if v[i][k]!=v[j][k]:

cd[i][j]+=1

#Calculating XB Index

d2=np.multiply(d,d)

U2=np.multiply(U,U)

var1=np.sum(np.multiply(U2,d2))

minsep=nd

for i in range(c):

for j in range(c):

if cd[i][j]>0 and cd[i][j]<minsep:

minsep=cd[i][j]

XB=var1/ns

XB=XB/minsep

print("%%%%%%%%%XB {0}".format(XB))

# Calculating Dunn Index

delta=np.multiply(U,d)

delta2=np.sum(delta,axis=1)

Xk=np.max(delta2)

dunn=np.inf

maxdunn=0

maxc=0

for i in range(c-1):

for j in range(i+1,c):

gg=cd[i][j]/Xk

if gg<dunn:

dunn=gg

if dunn>maxdunn:

maxdunn=dunn

maxc=c

print("\*\*\*\*\*\*\*\*\*Dunn {0}".format(dunn))

#calculating I Index

Dk=np.max(cd)

Ek=np.sum(np.multiply(U,d))

E1=np.sum(np.multiply(U,d),axis=1)[0]

Ik=(Dk\*E1)/(Ek\*c)

print("##########IK {0}".format(Ik))

IKK.append(Ik)

XBB.append(XB)

Dunnn.append(dunn)

XBB=np.array(XBB)

Dunnn=np.array(Dunnn)

IKK=np.array(IKK)

#Applying SVM to enhance clustering

oc=int(input("Enter the optimal number of cluster"))

U3=Update[oc-2]

U3=np.array(U3)

e=[]

mar=float(input("Enter the marginal value"))

for i in range(oc):

li=[]

for j in range(ns):

if(U3[i][j]>=mar):

#print(j)

li.append(j)

e.append(li)

Xtrain=[]

ytrain=[]

nshape=[]

count=0

for i in e:

for k in i:

nshape.append(k)

Xtrain.append(X[k])

ytrain.append(count)

count+=1

Xtrain=np.array(Xtrain)

ytrain=np.array(ytrain)

nshape=np.array(nshape)

Xtest=[]

for j in range(ns):

if j not in nshape:

Xtest.append(X[j])

Xtest=np.array(Xtest)

# Fitting Kernel SVM to the Training set

from sklearn.svm import SVC

classifier = SVC(kernel = 'rbf', random\_state = 0)

classifier.fit(Xtrain, ytrain)

# Predicting the Test set results

y\_pred = classifier.predict(Xtest)

ypred = classifier.predict(X)

**Fuzzy K-modes algorithm for clustering categorical data**

**Introduction**

The k-means algorithm is a well known for its efficiency in clustering large data sets. Fuzzy version of the k-means algorithm have been studied where each point in dataset is allowed to have membership functions to all clusters rather than having a distinct membership to exactly one cluster. However working only on numeric data limits the use of these k-means type algorithm in areas such as data mining where large categorical data sets are frequently encountered.

To tackle the problem of clustering large categorical datasets in data mining, the k-modes algorithm has recently been proposed in. The k-modes algorithm extends the

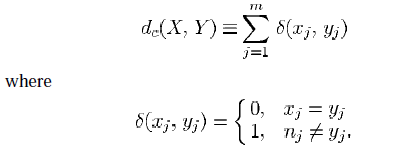
k-means algorithm by using a simple matching dissimilarity measure for categorical objects, modes instead of means for clusters, and a frequency-based method to update modes in the clustering process to minimize the clustering cost function. These extensions have removed the numeric-only limitation of k-means algorithm and enable it to be used to efficiently cluster large categorical datasets from real-world databases.

**Hard and fuzzy K-Modes Algorithm**

The hard k-modes algorithm, first introduced in, has made the following modifications to the -means algorithm: 1) using a simple matching dissimilarity measure for categorical

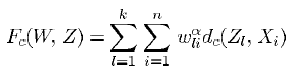
objects ; 2) replacing the means of clusters with the modes; and 3) using a frequency-based method to find the modes to update cluster centres. These modifications have removed the numeric-only limitation of the k-means algorithm but maintain its efficiency in clustering large categorical datasets .

Let X and Y be two categorical objects represented by [x1,x2…..xm] and [y1,y2…..ym] respectively. The simple matching dissimilarity measure between X and Y is defined as follows:



The k-modes algorithm uses the k-means paradigm to cluster categorical data. The objective of clustering a set of n categorical objects into k clusters is to find W and Z that

minimize:



with other conditions same as in fuzzy c-means algorithm. Here, Z represents a set

of modes for clusters. We can still use Fuzzy c-means algorithm to minimize Fc(W,Z) . However, the way to update Z at each iteration is different from the method given in Fuzzy c-means.

**The Hard k-modes Update Method**: Let X be a set of categorical objects described by categorical attributes A1,A2…Am and DOM(Aj)={aj1, aj2,… ajn} where n is the number of categories of attribute Aj for 1≤j≤m. Let the cluster centres Zl be represented by [zl,1,zl,2,…zl,m] for 1≤l≤k. Then the quantity Image is minimized iff zl,j=aj(r)€ DOM(Aj) where

Image

for 1≤j≤m. Here |X| denotes the number of elements in the set X.

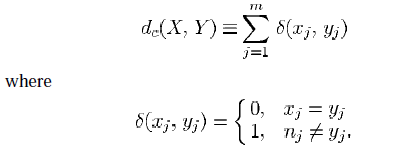
**The Fuzzy k-modes Update Method**: Let X be a set of categorical objects described by categorical attributes A1,A2…Am and DOM(Aj)={aj1, aj2,… ajn} where n is the number of categories of attribute Aj for 1≤j≤m. Let the cluster centres Zl be represented by [zl,1,zl,2,…zl,m] for 1≤l≤k. Then the quantity Image is minimized if zl,j=aj(r)€ DOM(Aj) where

Image

for 1≤j≤m.

Fuzzy c-means (FCM) clustering Algorithms

* Fix c, m. Choose an initial number of ‘c’ clusters from the dataset in v matrix.
* Compute distance matrix with the given equation:



* Compute membership matrix Ũk with the given equation:

Image

* Update the v matrix using Hard k-modes Update Method or Fuzzy k-modes Update Method.
* Iterate for a fixed number of iterations till Jmk+1-Jmk <€ and then stop.

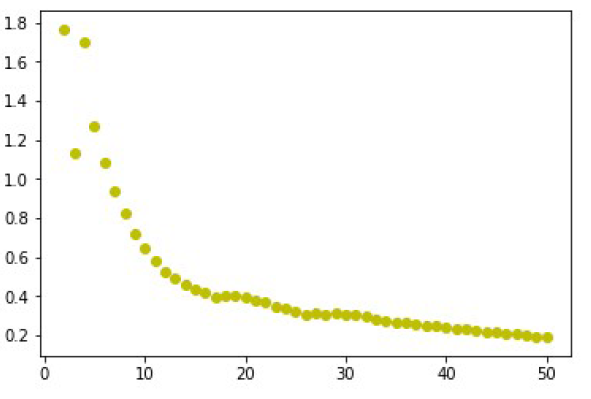
EXAMPLE:-

The dataset consists of 704 rows and 5 columns. The dataset is a categorical value dataset.

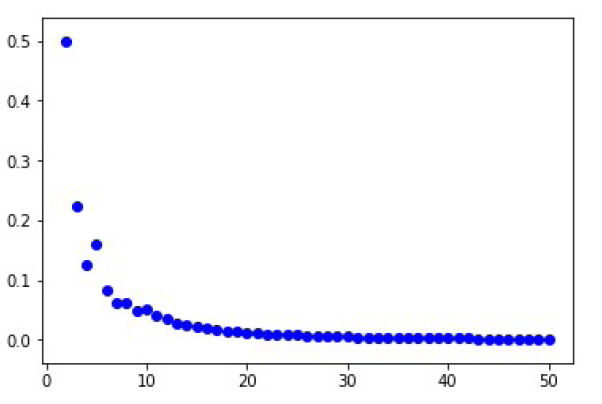
For the above data if we apply Xie-Beni(XB) index, Dunn index, *I* index we get the following results.

A) Fuzzy K-Modes:-

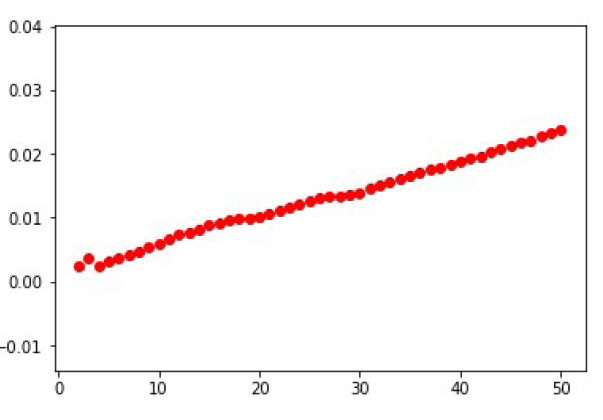
XB index



*I* index



Dunn Index



Taking **12** as the optimal number of clusters we get the following **cluster centers**

[['Y4', 'A', 'I3', 'A3', 'PC1'], ['Y4', 'A', 'I1', 'A2', 'PC2'], ['Y4', 'C', 'I1', 'A3', 'PC2'],

['Y4', 'A', 'I1', 'A2', 'PC8'], ['Y3', 'A', 'I3', 'A2', 'PC7'], ['Y4', 'B', 'I3', 'A3', 'PC8'],

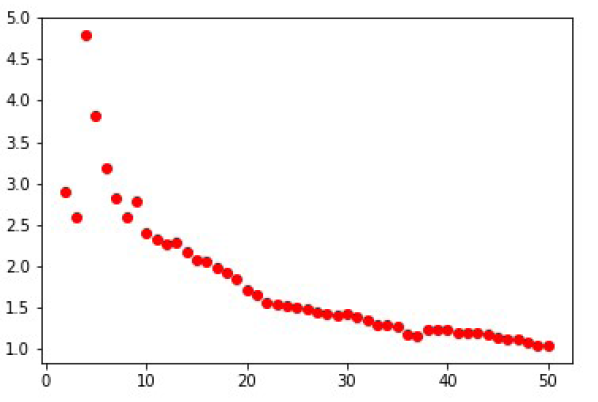
['Y4', 'A', 'I3', 'A2', 'PC4'], ['Y3', 'B', 'I3', 'A1', 'PC5'], ['Y4', 'B', 'I1', 'A2', 'PC8'],

['Y4', 'A', 'I1', 'A3', 'PC4'], ['Y4', 'A', 'I3', 'A2', 'PC2'], ['Y4', 'B', 'I1', 'A3', 'PC4']].

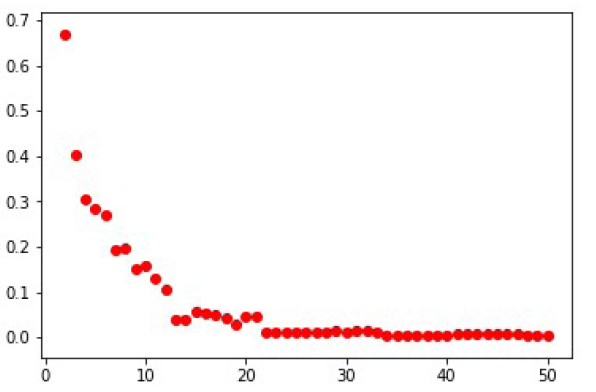
**The membership value is attached with the report.**

B) Hard K-Modes:-

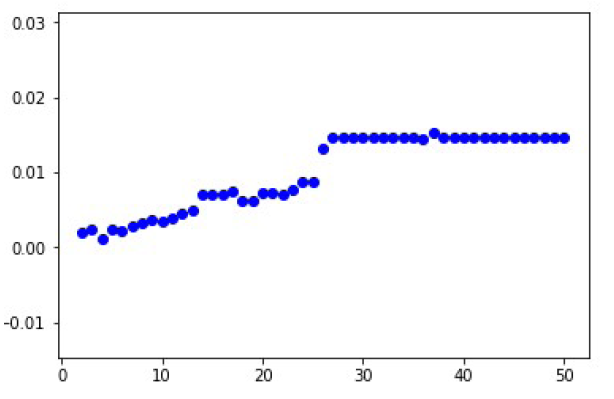
XB index



*I* index



Dunn index



Taking **22** as the optimal number of clusters we get the following **cluster centers**

[['Y4', 'A', 'I3', 'A1', 'PC1'], ['Y4', 'A', 'I1', 'A2', 'PC2'], ['Y4', 'C', 'I1', 'A2', 'PC1'],

['Y4', 'A', 'I1', 'A3', 'PC8'], ['Y4', 'A', 'I3', 'A2', 'PC7'], ['Y4', 'B', 'I3', 'A3', 'PC2'],

['Y4', 'C', 'I3', 'A2', 'PC4'], ['Y3', 'B', 'I3', 'A1', 'PC5'], ['Y4', 'B', 'I1', 'A2', 'PC8'],

['Y4', 'A', 'I1', 'A3', 'PC3'], ['Y4', 'A', 'I3', 'A2', 'PC2'], ['Y4', 'B', 'I1', 'A3', 'PC4'],

['Y4', 'A', 'I1', 'A3', 'PC4'], ['Y4', 'A', 'I1', 'A3', 'PC2'], ['Y3', 'A', 'I3', 'A1', 'PC6'],

['Y3', 'B', 'I1', 'A1', 'PC5'], ['Y4', 'A', 'I3', 'A3', 'PC4'], ['Y4', 'B', 'I1', 'A3', 'PC8'],

['Y4', 'A', 'I3', 'A2', 'PC4'], ['Y4', 'B', 'I4', 'A2', 'PC1'], ['Y4', 'A', 'I1', 'A2', 'PC4'],

['Y5', 'A', 'I3', 'A3', 'PC7']].

**The membership value is attached with the report.**

**Codes for fuzzy k-modes algorithm for clustering of categorical data**

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Data2.csv')

X = dataset.iloc[:,0:5].values

a=np.shape(X)

ns=a[0] #number of records

nd=a[1] #number of features

cluster=[] # A list to include all clusters

Update=[] # A list to include all partion martix

m=float(input("Enter The value of weight exponent m between 1.5 and 3 for fuzzy clustering and 1 for hard clustering"))

kbegin=2

kcease=int(input("Enter the total number of cluster"))

jm=[] #Cost Function and ending criterian

lmax=50 #Ending Criteria

X3=X

X3=pd.DataFrame(X3)

X3=X3.drop\_duplicates().values

XBB=[]

Dunnn=[]

IKK=[]

X3=np.array(X3)

for zz in range(2,kcease+1):

jl=[]

c=zz

print(c)

v=[] #getting initial clusters

for i in range(c):

v.append(X3[i])

d=np.zeros([c,ns],dtype=float) #getting initial distance matrix

for i in range(c):

for j in range(ns):

for k in range(nd):

if v[i][k]!=X[j][k]:

d[i][j]+=1

U=np.zeros([c,ns],dtype=float) #getting initial fuzzy matrix

if m==1:

cmin=np.amin(d,axis=0)

for i in range(ns):

for j in range(c):

if d[j][i]==cmin[i]:

U[j][i]=1

break

else:

U[j][i]=0

else:

v=np.array(v)

ci=[]

xi=[]

for j in range(ns):

for x in range(c):

cou=0

for y in range(nd):

if X[j][y]==v[x][y]:

cou+=1

if cou==nd:

ci.append(x)

xi.append(j)

ns2=[]

for k in range(ns):

if k not in xi:

ns2.append(k)

for x in range(len(ci)):

U[ci[x]][xi[x]]=1

for ii in ns2:

for jj in range(c):

su=0

for kk in range (c):

su=su+((d[jj][ii]/d[kk][ii])\*\*(1/(m-1)))

U[jj][ii]=(1/su)

# Update.append(U)

# cluster.append(v)

# sum1=np.sum(np.multiply(U,d))

v=np.array(v)

for mm in range(lmax):

if m==1:

#Update.append(U)

#cluster.append(v)

fco=[]

for i in range(c):

co=[]

for j in range(ns):

if U[i][j]==1:

co.append(j)

fco.append(co)

k=0

for cou in fco:

nX=[]

for i in cou:

nX.append(X[i])

nX=np.array(nX)

for i in range(nd):

ftable=pd.crosstab(index=nX[:,i],columns='count')

amax=ftable['count'].idxmax(axis=0)

v[k][i]=amax #Updating clusters

k+=1

d=np.zeros([c,ns],dtype=float) #Updating distance matrix

for i in range(c):

for j in range(ns):

for k in range(nd):

if v[i][k]!=X[j][k]:

d[i][j]+=1

# sum2=np.sum(np.multiply(U,d))

# if sum2 >= sum1:

# break

# else:

# sum1=sum2

U=np.zeros([c,ns],dtype=float) #setting initial fuzzy matrix

cmin=np.amin(d,axis=0) #Updating fuzzy matrix

for i in range(ns):

for j in range(c):

if d[j][i]==cmin[i]:

U[j][i]=1

break

else:

U[j][i]=0

# sum2=np.sum(np.multiply(U,d))

# if sum2 >= sum1:

# break

# else:

# sum1=sum2

#Update.append(U)

#cluster.append(v)

else:

#Update.append(U)

#cluster.append(v)

X2=pd.DataFrame(X)

for i in range(nd):

Uc=X2.iloc[:,i].unique().tolist()

for l in range(c):

sm=np.zeros(len(Uc),dtype=float)

for j in range(len(Uc)):

for k in range(ns):

if Uc[j]==X2[i][k]:

sm[j]=sm[j]+U[l][k]\*\*m

lock=np.where(sm==np.amax(sm))

lock=np.array(lock)

v[l][i]=Uc[lock[0][0]] #Updating V matrix

d=np.zeros([c,ns],dtype=float) #Updating distance matrix

for i in range(c):

for j in range(ns):

for k in range(nd):

if v[i][k]!=X[j][k]:

d[i][j]+=1

# sum2=np.sum(np.multiply(U,d))

# if sum2 >= sum1:

# break

# else:

# sum1=sum2

U=np.zeros([c,ns],dtype=float) #Updating fuzzy matrix

X=np.array(X)

v=np.array(v)

ci=[]

xi=[]

for j in range(ns):

for x in range(c):

cou=0

for y in range(nd):

if X[j][y]==v[x][y]:

cou+=1

if cou==nd:

ci.append(x)

xi.append(j)

ns2=[]

for k in range(ns):

if k not in xi:

ns2.append(k)

for x in range(len(ci)):

U[ci[x]][xi[x]]=1

for ii in ns2:

for jj in range(c):

su=0

for kk in range (c):

su=su+((d[jj][ii]/d[kk][ii])\*\*(1/(m-1)))

U[jj][ii]=(1/su)

# sum2=np.sum(np.multiply(U,d))

# if sum2 >= sum1:

# break

# else:

# sum1=sum2

#

Update.append(U)

cluster.append(v)

jm.append(np.sum(np.multiply(U,d)))

cd=np.zeros([c,c],dtype=float) #getting cluster distance matrix

for i in range(c):

for j in range(c):

for k in range(nd):

if v[i][k]!=v[j][k]:

cd[i][j]+=1

#Calculating XB Index

d2=np.multiply(d,d)

U2=np.multiply(U,U)

var1=np.sum(np.multiply(U2,d2))

minsep=nd

for i in range(c):

for j in range(c):

if cd[i][j]>0 and cd[i][j]<minsep:

minsep=cd[i][j]

# print(minsep)

XB=var1/ns

XB=XB/minsep

print("%%%%%%%%%XB {0}".format(XB))

# Calculating Dunn Index

delta=np.multiply(U,d)

delta2=np.sum(delta,axis=1)

Xk=np.max(delta2)

dunn=np.inf

maxdunn=0

maxc=0

for i in range(c-1):

for j in range(i+1,c):

gg=cd[i][j]/Xk

if gg<dunn:

dunn=gg

if dunn>maxdunn:

maxdunn=dunn

maxc=c

print("\*\*\*\*\*\*\*\*\*Dunn {0}".format(dunn))

#calculating I Index

Dk=np.max(cd)

Ek=np.sum(np.multiply(U,d))

E1=np.sum(np.multiply(U,d),axis=1)[0]

Ik=(Dk\*E1)/(Ek\*c)

print("##########IK {0}".format(Ik))

XBB.append(XB)

Dunnn.append(dunn)

IKK.append(Ik)

XBB=np.array(XBB)

Dunnn=np.array(Dunnn)

IKK=np.array(IKK)

**Fuzzy clustering of categorical data using fuzzy centroids**

**INTRODUCTION**

In most fuzzy versions of clustering algorithms,the assigned memberships of data to a cluster are fuzzy, but the centroid itself is not fuzzy. However,the use of hard centroids can give rise to artifacts.For example, although the fuzzy k-modes algorithm

efficiently handles categorical data sets, it uses a hard centroid representation for categorical data in a cluster. This use of hard rejection of data can lead to misclassification in the region of doubt.To address the problems caused by using hard

centroids, in the present study we developed a fuzzy clustering algorithm with fuzzy centroids for clustering categorical data. The use of fuzzy centroids

allows the user to fully exploit the power of fuzzy sets in representing uncertainty and imprecision.

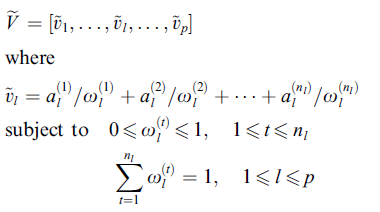
Let DOM(Al)={high, low} and let us consider three data X1, X2, and X3 whose degrees

of membership to the ith cluster are ui1=0.70, ui2=0.80,ui3=0.15 respectively.

X1 =[x1,l,…x1,l-1, “high”,..x1,p], X2 =[x2,l,…x2,l-1, “low”,..x2,p], X3 =[x3,l,…x3,l-1, “high”,..x3,p]

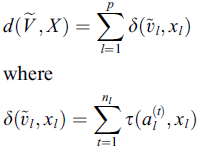
Consider the lth attribute, vi,l, of the cluster centroid V=[vi,1,…vi,l,...vi,p]. In fuzzy k – modes algorithmvi,l is assigned the value ‘‘high’’ or ‘‘low’’ depending on the calculations of ∑xj,l=high uijm=0.70m+0.15m and ∑x j,l=lowuijm=0.80m. For instance, vi,l is assigned ‘‘high’’ for m= 1.0, whereas vi.l is assigned ‘‘low’’ for m = 2.0. According to the decision,one of the two is rejected and, despite its potential, is not concerned with the computations of the membership degrees (uij) of data in the next

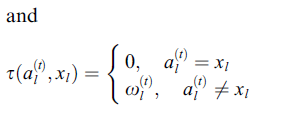
iteration. This can lead to the misclassifications of data, and therefore drive the algorithm to fall into a local minimum. To prevent this a soft decision is made when selecting the cluster centroids for categorical attributes, thereby preserving the uncertainty for long as possible before actual decisions are made. To achieve this objective, we introduce the concept of a fuzzy centroid. In a hard centroid, each attribute of the centroid has a single hard category value. In contrast, each attribute of a fuzzy centroid has a fuzzy category value to describe the information distributed in the cluster. For DOM(Al)={al(1),al(2),..al(n)}, the proposed fuzzy centroid, denoted by Ṽ , is defined as



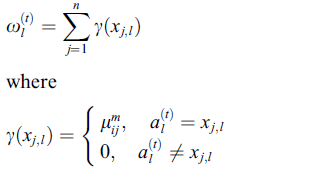
Distance measure and centroids’s update

Let Ṽ and X be a fuzzy centroid and a data point be represented as [ṽ1,ṽ2…ṽp] and [x1,x2,…xp]. The distance between Ṽ and X is defined as



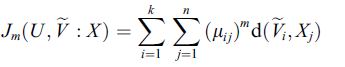


The fuzzy centroids Ṽi=[ṽi,1,…ṽi,l,..ṽi,p] is updated when the partition matrix U=[uij] is determined based on the distance measure. The attribute ṽl is then updated by determining wlt for 1≤t≤nl as follows:



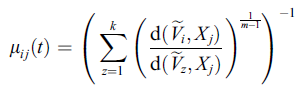
Proposed algorithm:-

To minimize the objective objective function with fuzzy centroids,

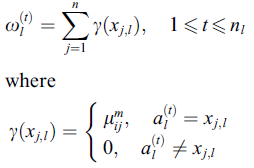


the proposed algorithm uses fuzzy c-means type procedure to cluster categorical data as follows:

* Given the number of clusters, k, and a chosen value of m, choose initial centroids Ṽ(0)(t=0). Each ṽi,l € Ṽi is assigned random membership values for wl(t).
* Compute the ith fuzzy cluster for i=1,2,…, k. For each xj:



* Update the fuzzy cluster centroid Ṽ(t+1)=[ṽi,1,…ṽi,l…ṽi,p] for i = 1, 2, . . . , k. For each ṽi,l={(al(t),wl(t))} for 1≤l≤p



* If there is no improvement in Jm, then stop; otherwise, set t←t+1 and go to

Step 2.

**Codes for fuzzy clustering of categorical data using fuzzy centroids**

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Data2.csv')

X = dataset.iloc[:,0:5].values

X2=pd.DataFrame(X)

a=np.shape(X)

ns=a[0]

nd=a[1]

cluster=[]

Update=[]

m=float(input("Enter The value of weight exponent m between 1.5 and 3"))

c=int(input("Enter the total number of cluster"))

jm=0

lmax=50 # Ending Criteria

v=[] #getting initial clusters

for i in range(c):

v.append(X[i,:])

v=np.array(v)

uni=[]

W=[] # calculating initial W matrix

for kk in range(c):

w=[]

for i in range(nd):

Uc=X2.iloc[:,i].unique().tolist()

uni.append(Uc)

#print(Uc)

l=len(Uc)

aa=np.random.rand(l)

su=np.sum(aa)

aa=aa/su

w.append(aa)

#print(w)

W.append(w)

uni=[]

l=[]

for i in range(nd):

Uc=X2.iloc[:,i].unique().tolist()

uni.append(Uc)

l.append(len(Uc))

d=np.zeros([c,ns],dtype=float) #getting initial distance matrix

for i in range(c):

for j in range(ns):

for k in range(nd):

for ll in range(l[k]):

if X[j][k]!=uni[k][ll]:

d[i][j]+=W[i][k][ll]

U=np.zeros([c,ns],dtype=float) #Calculating initial U matrix

for i in range(c):

for k in range(ns):

aa=0

for j in range(c):

aa+=((d[i][k]/d[j][k])\*\*(1/(m-1)))

U[i][k]=1/aa

jm=np.sum(np.multiply(U,d)) # initial cost matrix

W=np.array(W)

for mm in range(1000):

for i in range(c): # Updating W matrix

for k in range(nd):

for ll in range(l[k]):

W[i][k][ll]=0

for j in range(ns):

if X[j][k]==uni[k][ll]:

W[i][k][ll]=W[i][k][ll]+U[i][j]\*\*m

su=np.sum(W[i][k])

W[i][k]=W[i][k]/su

d=np.zeros([c,ns],dtype=float) #Updating distance matrix

for i in range(c):

for j in range(ns):

for k in range(nd):

for ll in range(l[k]):

if X[j][k]!=uni[k][ll]:

d[i][j]+=W[i][k][ll]

U=np.zeros([c,ns],dtype=float) #Updating U matrix

for i in range(c):

for k in range(ns):

aa=0

for j in range(c):

aa+=((d[i][k]/d[j][k])\*\*(1/(m-1)))

U[i][k]=1/aa

jm=np.sum(np.multiply(U,d)) # calculating cost matrix

print(jm)

**Integrated Clustering and Supervised Learning for Categorical Data Analysis**

**INTRODUCTION**

The problem of fuzzy clustering of categorical data, where no natural ordering among the elements of a categorical attribute domain can be found, is an important problem in exploratory data analysis. In this project, a modified differential evolution (DE)-based fuzzy c-medoids (FCMdd) clustering of categorical data has been presented. The

algorithm combines both local as well as global information with adaptive weighting. To improve the result further, the clustering method is integrated with a support vector machine (SVM), a well-known technique for supervised learning. A fraction

of the data points are selected from different clusters based on their proximity to the respective medoids is used for training the SVM.The clustering assignments of the remaining points are thereafter determined using the trained classifier.

This algorithm is based on finding *K* representative objects (also known as

medoids) from the data set in such a way that the sum of the within cluster dissimilarities is minimized.

**Differential evolution (DE)** is a stochastic search and optimization tool that uses a special kind of differential operator to create new offspring from parent vectors

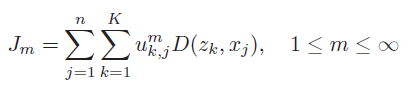
instead of the classical mutation operations. DE minimizes an objective function that can model the problem’s objectives while incorporating constraints .DE is able to reach the global optimal faster for several function optimization problem for different values of control parameters. The other important features of DE are, it is simple, fast, easy to use, and quite effective in nonlinear constraint optimization including penalty functions. It is useful for search and optimization in multimodal search spaces. The proposed modified variant of DE differs from the classical DE in the process of mutation. While doing the mutation, it uses three vectors: one representing the local best (LBest), the other the global best (GBest) which are adaptive in nature, and the third one is selected randomly.

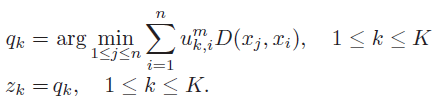
Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. They belong to a family of generalized linear classifiers. A special property of SVMs is that they simultaneously minimize the classification error and maximize the geometric margin. Hence, they are also known as maximum margin classifiers.SVM is a particularly a good tool to classify a set of points

which belong to two classes. SVMs are based on statistical learning theory , and they try to find the biggest margin to separate the two classes. SVMs embed data into a high dimensional feature space. This method then finds the hyperplane that separates the largest possible fraction of points such that points on the same side belong to the same class, while the distance of each class from the hyperplane is maximized.

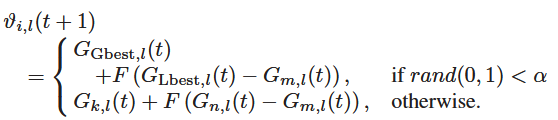
**Proposed Algorithm**

* **Vector Representation and Population Initialization***:* Each vector is a sequence of index values of *{*1*,* 2*, . . . , n}* from data space representing the *K* cluster medoids. If vector *I* encodes the medoids of *K* clusters, then its length *l* is *K*. The *K* cluster medoids encoded in each vector are initialized to *K* randomly chosen index points from the data set. This process is repeated for each of the *P* vectors in the population, where *P* is the size of the population.
* FitnessComputation*:* The fitness of each vector (*Jm*) is computed either using first equation given below. Subsequently, the medoids encoded in a vector are updated using second equation given below.

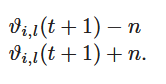




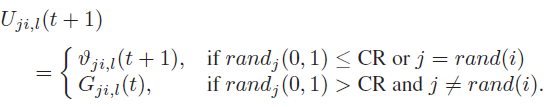
* **Mutation**: The ith individual vector of the population at time step (generation) t has k components, i.e., Gi,l(t) = [Gi,1(t),Gi,2(t), . . . , Gi,l(t)] . For each target vector Gi,l(t) that belongs to the current population, we use α as a criteria on favor of that MoDE randomly samples three other individuals, i.e., GLbest,l(t), GGbest,l(t), and Gm,l(t) (as describe earlier) and otherwise, Gk,l(t), Gn,l(t), and Gm,l(t) from the same generation (for distinct k, i, j, and m). Then, calculate the (component wise) difference, scales it by a scalar F (usually € [0, 1]), and create a trial offspring ϑi,l(t + 1). Thus, for the lth component of each vector, where l = 1, 2, . . .,K



If the index value of *ϑi,l*(*t* + 1) lie beyond the permissible range of *{*1*, . . . , n}*, then it scaled using one of the following two operations:



* **Crossover**: In order to increase the diversity of the perturbed parameter vectors, crossover is introduced. To this end, the trial vector Ui,l(t + 1) = [Ui,1(t + 1), Ui,2(t + 1), . . . , Ui,3(t + 1)] is formed, where



CR is the crossover constant *€* [0*,* 1] which has to be determined by the user.

*rand*(*i*) is a randomly chosen index *€* 1*,* 2*, . . . , l* which ensures that *Ui,l*(*t* + 1) gets at least one parameter from *ϑi,l*(*t* + 1).

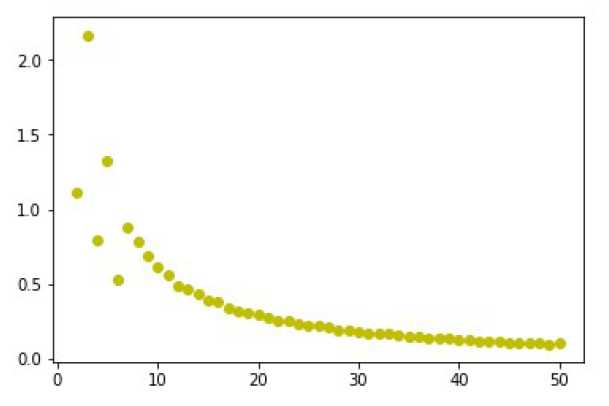
* **Selection**: To decide whether it should become a member of generation G + 1, the trial vector Ui,l(t + 1) is compared to the target vector Gi,l(t) using the greedy criterion. If vector Ui,l(t + 1) yields a smaller cost function value than Gi,l(t), then Ui,l(t + 1) is set to Gi,l(t); otherwise, the old value Gi,l(t) is retained.
* **Termination Criterion**: The processes of mutation, crossover and selection are executed for a fixed number of iterations.The best vector seen up to the last generation provides the solution to the clustering problem.

EXAMPLE:-

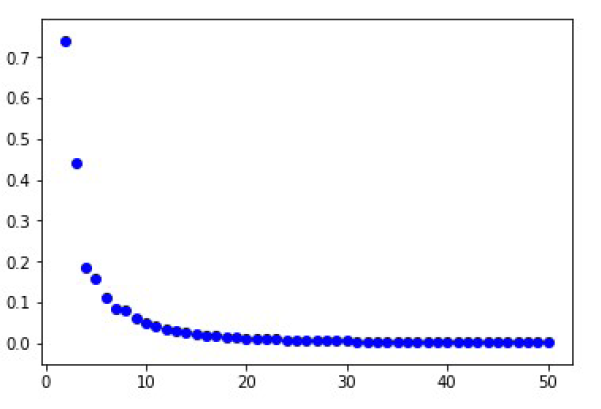
The dataset consists of 704 rows and 5 columns. The dataset is a categorical value dataset.

For the above data if we apply Xie-Beni(XB) index, *I* index we get the following results.

XB index



*I* index



Taking **12** as the optimal number of clusters we get the following **cluster centers**

[['Y5' 'B' 'I3' 'A3' 'PC2'], ['Y3' 'A' 'I1' 'A3' 'PC4'], ['Y4' 'B' 'I1' 'A3' 'PC4']

['Y4' 'B' 'I1' 'A2' 'PC2'], ['Y4' 'A' 'I1' 'A3' 'PC4'], ['Y5' 'A' 'I1' 'A3' 'PC4']

['Y4' 'B' 'I3' 'A2' 'PC2'], ['Y4' 'A' 'I1' 'A3' 'PC3'], ['Y4' 'A' 'I1' 'A3' 'PC8']

['Y4' 'B' 'I3' 'A3' 'PC2'], ['Y4' 'A' 'I3' 'A2' 'PC7'], ['Y4' 'A' 'I3' 'A3' 'PC2']]

The index of above clusters are [616, 52, 571, 368, 313, 349, 667, 635, 232, 331, 629 431].

**The membership value is attached with the report.**

**Codes for fuzzy Integrating clustering and Supervised Learning for categorical data analysis**

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Data2.csv')

X = dataset.iloc[:,0:5].values

a=np.shape(X)

ns=a[0] #number of records

nd=a[1] #number of features

D=np.zeros([ns,ns],dtype=float) #calculating distance matrix

for i in range(ns):

for j in range(ns):

sum=0

for k in range(nd):

if X[i][k]!=X[j][k]:

sum+=1

D[i][j]=sum

p=int(input("Enter a population size"))

c=int(input("Enter the total number of clusters"))

m=float(input("Enter The value of weight exponent m between 1.5 and 3"))

cluster=[]

clus=[] #initializing population

for i in range(p):

cluster.append(np.random.randint(0,ns,size=c))

cluster=np.array(cluster)

for i in cluster:

v=i

d=[]

for j in i:

d.append(D[j])

d=np.array(d)

U=np.zeros([c,ns],dtype=float) #getting initial fuzzy matrix

v=np.array(v)

V=[]

for q in v:

V.append(X[q])

V=np.array(V)

ci=[]

xi=[]

for j in range(ns):

for x in range(c):

cou=0

for y in range(nd):

if X[j][y]==V[x][y]:

cou+=1

if cou==nd:

ci.append(x)

xi.append(j)

ns2=[]

for k in range(ns):

if k not in xi:

ns2.append(k)

for x in range(len(ci)):

U[ci[x]][xi[x]]=1

for ii in ns2:

for jj in range(c):

su=0

for kk in range (c):

su=su+((d[jj][ii]/d[kk][ii])\*\*(1/(m-1)))

U[jj][ii]=(1/su)

for pp in range(c): #Updating the cluster to get initial cluster.

su=[]

for qq in range(ns):

sum=0

for rr in range(ns):

sum=sum+((U[pp][rr])\*\*m)\*D[qq][rr]

su.append(sum)

#print(su)

clus.append(su.index(min(su)))

clus=np.array(clus)

clus=np.reshape(clus,(p,c))

jm=[]

for i in clus: #calculating fitness

v=i

d=[]

for j in v:

d.append(D[j])

d=np.array(d)

U=np.zeros([c,ns],dtype=float) #getting fuzzy matrix

v=np.array(v)

V=[]

for q in v:

V.append(X[q])

V=np.array(V)

ci=[]

xi=[]

for j in range(ns):

for x in range(c):

cou=0

for y in range(nd):

if X[j][y]==V[x][y]:

cou+=1

if cou==nd:

ci.append(x)

xi.append(j)

ns2=[]

for k in range(ns):

if k not in xi:

ns2.append(k)

for x in range(len(ci)):

U[ci[x]][xi[x]]=1

for ii in ns2:

for jj in range(c):

su=0

for kk in range (c):

su=su+((d[jj][ii]/d[kk][ii])\*\*(1/(m-1)))

U[jj][ii]=(1/su)

sum=0

for kk in range(c):

for jj in range(ns):

sum=sum+(U[kk][jj]\*\*m)\*d[kk][jj]

jm.append(sum)

jm=np.array(jm)

jm=np.reshape(jm,(p,1))

Gb=np.amin(jm)

lb=Gb

x=np.where(jm==np.amin(jm))

x=np.array(x)

Gbest=clus[x[0]]

Gbest=Gbest[0]

Glbest=Gbest

cr=float(input("Enter the crossover value between 0 and 1"))

lmax=int(input("Enter the total number of iterations"))

f=int(input("Enter the value of F"))

for zz in range(lmax):

al=1/(1+np.exp(-zz))

cl=[]

jl=[]

for iii in range(p):

i=clus[iii]

if(np.random.uniform()<al): #Mutation

vi=Gbest+(Glbest-i)\*f

else:

ab=np.random.randint(0,p,size=2)

vi=clus[ab[0]]+(clus[ab[1]]-i)\*f

for k in range(c): #adjustment

while(vi[k]>=ns or vi[k]<0):

if(vi[k]>=ns):

vi[k]=vi[k]-ns

if(vi[k]<0):

vi[k]=vi[k]+ns

UU=[]

pp=np.random.randint(0,c,size=1)

for k in range(c):

ll=np.random.uniform()

if(ll<=cr or k==pp):

UU.append(vi[k])

else:

UU.append(i[k])

UU=np.array(UU)

#print(UU)

v=UU

d=[]

for j in v: #making distance matrix

d.append(D[j])

d=np.array(d)

U=np.zeros([c,ns],dtype=float) #getting fuzzy matrix

v=np.array(v)

V=[]

for q in v: #getting values from dataset

V.append(X[q])

V=np.array(V)

ci=[]

xi=[]

for j in range(ns):

for x in range(c):

cou=0

for y in range(nd):

if X[j][y]==V[x][y]:

cou+=1

if cou==nd:

ci.append(x)

xi.append(j)

ns2=[]

for k in range(ns):

if k not in xi:

ns2.append(k)

for xx in range(len(ci)):

U[ci[xx]][xi[xx]]=1

for ii in ns2:

for jj in range(c):

su=0

for kk in range (c):

su=su+((d[jj][ii]/d[kk][ii])\*\*(1/(m-1)))

U[jj][ii]=(1/su)

sum=0 #calculating fitness

for kk in range(c):

for jj in range(ns):

sum=sum+(U[kk][jj]\*\*m)\*d[kk][jj]

#print(iii)

if(sum<=jm[iii]):

jl.append(sum)

cl.append(UU)

else:

jl.append(jm[iii])

cl.append(i)

jl=np.array(jl)

jl=np.reshape(jl,(p,1))

jm=jl

cl=np.array(cl)

x=np.where(jl==np.amin(jl))

x=np.array(x)

Glbest=cl[x[0]]

Glbest=Glbest[0]

lb=np.amin(jl)

if(lb<Gb):

Gb=lb

Gbest=Glbest

clus=cl

print(Gb)

print(Gbest)

v=Gbest

d=[]

for j in v: #making distance matrix

d.append(D[j])

d=np.array(d)

U=np.zeros([c,ns],dtype=float) #getting fuzzy matrix

v=np.array(v)

V=[]

for q in v: #getting values from dataset

V.append(X[q])

V=np.array(V)

ci=[]

xi=[]

for j in range(ns):

for x in range(c):

cou=0

for y in range(nd):

if X[j][y]==V[x][y]:

cou+=1

if cou==nd:

ci.append(x)

xi.append(j)

ns2=[]

for k in range(ns):

if k not in xi:

ns2.append(k)

for xx in range(len(ci)):

U[ci[xx]][xi[xx]]=1

for ii in ns2:

for jj in range(c):

su=0

for kk in range (c):

su=su+((d[jj][ii]/d[kk][ii])\*\*(1/(m-1)))

U[jj][ii]=(1/su)

sum=0 #calculating fitness

for kk in range(c):

for jj in range(ns):

sum=sum+(U[kk][jj]\*\*m)\*d[kk][jj]

for i in range(len(v)):

print(X[v[i]])

#Applying SVM to enhance clustering

#e=[]

#mar=float(input("Enter the marginal value"))

#for i in range(c):

# li=[]

# for j in range(ns):

# if(U[i][j]>=mar):

# #print(j)

# li.append(j)

# e.append(li)

#

#Xtrain=[]

#ytrain=[]

#nshape=[]

#count=0

#for i in e:

# for k in i:

# nshape.append(k)

# Xtrain.append(X[k])

# ytrain.append(count)

# count+=1

#

#Xtrain=np.array(Xtrain)

#ytrain=np.array(ytrain)

#

#nshape=np.array(nshape)

#

#Xtest=[]

#for j in range(ns):

# if j not in nshape:

# Xtest.append(X[j])

#

#

#Xtest=np.array(Xtest)

#

## Fitting Kernel SVM to the Training set

#from sklearn.svm import SVC

#classifier = SVC(kernel = 'rbf', random\_state = 0)

#classifier.fit(Xtrain, ytrain)

#

## Predicting the Test set results

#y\_pred = classifier.predict(Xtest)

#

#ypred = classifier.predict(X)

**References**

Bibliography

* Applied Multivariate Statistical Modeling by Dr. J Maiti, Department of industrial and systems Engineering, IIT Kharagpur.
* FCM: The Fuzzy c-means clustering Algorithm by James C. Bezdek, Mathematics Department, Utah State University, Logan, U.S.A., Robert Ehrlich, Geology Department, University of South Carolina, Columbia, U.S.A., William Full, Geology Department, Wichita State University, Wichita, U.S.A.
* A fuzzy k-modes algorithm for clustering categorical data. Huang,Z;Ng,MKP.
* Fuzzy clustering of categorical data using fuzzy centroids by Dae-Won Kim and Kwang H. Lee, Department of Electrical Engineering & Computer Science, KAIST, South Korea, Doheon Lee, Department of BioSystems and Advanced Information Technology Research Center, KAIST, South Korea.
* Integrated Clustering and Supervised Learning for Categorical Data Analysis by Ujjwal Maulik, Senior Member, IEEE, Sanghamitra Bandyopadhyay, Senior Member, IEEE, and Indrajit Saha.
* Using internal evaluation measures to validaye the quality of diverse stream clustering algorithms by Marwan Hassani, Architecture of Information Systems Group, Eindhoven University of Technology, Eindhoven, The Netherlands, Thomas Seidl, Database Systems Group, LMU Munich, Munich, Germany