Possibilistic Fuzzy Neural Gas

Release 1.0.0

Amit Kumar

Jul 13, 2021

CONTENTS:

CHAPTER

ONE

INTRODUCTION

Neural Gas is inspired by the Self-Organizing Map for finding optimal data representations based on feature vectors. The algorithm was named **Neural Gas** because of the dynamics of the feature vectors during the adaptation process, which distribute themselves like a gas within the data space. Among the variety of methods that has been developed for clustering, Neural gas uses *prototype-based* approach. Prototype-based technique tries to define clusters using small set of prototypes. Where each prototype tries to represent the group of data points based on similarity measure to the prototype which can be influenced by size and shape of the parameter. Example of prototype based clustering algorithms are: c-Means or k-means (where, c or k is the number of prototypes), Self-Organizing Map, Neural Gas etc.

The principle version of all these clustering methods is that each data point is represented uniquely by exactly one prototype. This is also known as crisp clustering. In real world most of the time data is overlapping, so a clustering method which uses soft assignment of data point to the prototypes can be very helpful in understanding the structure of the data. This is also known as clustering. Example of fuzzy clustering algorithms are: Fuzzy c-means, Fuzzy Self Organizing Map, Fuzzy Neural Gas etc.

Possibilistic Fuzzy Neural Gas [1] helps in clustering when the data is overlapping. There are two kind of a assignments that are used:

`Probabilistic` – which ranges [0, 1] and for each data point to all the prototypes the values should sums up to 1.

`Possibilistic` – the value of the assignment decreases with the increase in the distance to the prototype.

PFNG can be used to cluster different kind of data:

`Vectorial data:` – Contains N x M-dimensional real valued vectors, and the distance measures used here is squared Euclidean distance.

`Relational data:` – It is a non vectorial data, the dissimilarity matrix **D** is provided which is of N x N dimension and the matrix is symmetric and complete. e.g. music, text, gene sequence etc.

`Median data:` –: Contains N x N dimensional data, the data has same properties as Relational data.

CHAPTER

TWO

ALGORITHM DESCRIPTION

2.1 Dataset

Given a data set $X = \{x_1, x_2, ...x_N\}$ of length N. To cluster group of similar objects together there is a need of a measure called Dissimilarity $D_{ij} = D(x_i, x_j)$ and depending upon the type of data D_{ij} can be distinguished as:

1. Vectorial: The data is provided as d-dimensional real valued vectors. The distance measure used here is squared Euclidean Distance formulated as :

$$D_{ij} = d^2(x_i, x_j) = \sum_{n=1}^{d}(x_{in}, x_{jn}) = \langle x_i, x_i \rangle_E - 2\langle x_i, x_j \rangle_E + \langle x_j, x_j \rangle_E \tag{2.1}$$

   For Relational and Median, the data which is being dealt is non-Vectorial data e.g. Music, Video etc. so, the D_{ij} is given before for clustering. Now, the $D_{ij} \in \mathbb{R}_+^{NxN}$ is assumed to have all pairwise dissimilarities where, the elements at $D_{ij} = D_{ji}$ and $D_{ii} = 0$.

2. Relational: Here assumption is made that there exists a non-linear mapping which projects the data into Euclidean space. Now, the D_{ij} is considered to have squared Euclidean distance with

$$D_{ij} = d_E^2(g(x_i), g(x_j)) = d_E^2(x_i, x_j) \tag{2.2}$$

   .

3. Median: The clustering for these kind of data relies on the dissimilarities which are given in **D**.

2.2 PFNG

The PFNG includes local neighborhood relations by taking in account the distance between the data samples and prototypes by calculating the ranks of the prototypes.

2.2.1 Local Cost

It is calculated as

$$lc_\sigma = \sum_{l=1}^{N_p} h_\sigma(w_j, w_l) \cdot D_{il}, \tag{2.3}$$

where, N_p is the number of prototypes and $h_\sigma(w_j, w_l)$ is neighborhood function defined as,

$$h_\sigma(w_j, w_l) = c_\sigma \cdot exp\left(\frac{-rk_j(w_j, W)}{2\sigma^2}\right) \tag{2.4}$$

where, c_σ ensures the $\sum_{l=1}^{N_p} h_\sigma(w_j, w_l) = 1$, and $\sigma > 0$ is the range of the neighborhood. Now, $rk_j(w_j, W))$ updates the closest prototype the most and vice versa. It is calculated as,

$$rk_j(w_j, W)) = \sum_{k=1}^{N_p} \mathcal{H}(d(w_l, w_j) - d(w_l, w_k)) \tag{2.5}$$

where, $\mathcal{H}(x)$ is a Heaviside function, and it gives rank or prototype w_l with respect to w_j

2.2.2 Membership and Typicality

The degree of membership and typicality of a data point v_i with respect of prototype w_j is calculates as

$$u_{ij} = \left(\sum_{l=1}^{N_p} \left(\frac{lc(v_i, w_j)}{lc(v_i, w_l)}\right)^{m-1}\right)^{-1} \tag{2.6}$$

and,

$$t_{ij} = \left(1 + \left(\frac{b}{\gamma_i} lc(v_i, w_j)\right)^{\eta-1}\right)^{-1} \tag{2.7}$$

2.3 Vectorial Data

2.3.1 Prototype update

Given $V = \{v_1, v_2, ...v_{N_v}\}$, a d-dimensional real vectors, and set of prototypes $W = \{w_1, w_2, ...w_{N_p}\}$ where, v_i and $w_j \in \mathbb{R}^d$. If sED is the distance measure then prototype can be updates explicitly using the equation,

$$\mathbf{w}_j = \frac{\sum_{i=1}^{N_v} \sum_{l=1}^{N_p} \beta_{il} h_\sigma(w_j, w_l) v_i}{\sum_{i=1}^{N_v} \sum_{l=1} \beta_{il} h_\sigma(w_j, w_l)} \tag{2.8}$$

where, $\beta_{il} = (au_{il}^m + bt_{il}^\eta)$. if sED is not the distance measure then Stochastic Gradient Descent Learning is performed according to,

$$\Delta \mathbf{w}_j \propto \sum_{i=1}^{N_v} \sum_{l=1}^{N_p} \beta_{il} h_\sigma(w_j, w_l) \cdot \frac{\partial(v_i, w_j)}{\partial w_j} \tag{2.9}$$

2.4 Relational Data

2.4.1 Prototype

Here, prototypes are defined as the convex linear combination of data points and is defined as,

$$w_j = \sum_{i=1}^{N_v} \alpha_{ji} v_i \tag{2.10}$$

where, $\alpha_{ji} > 0$ and $\sum_{i=1}^{N_v} \alpha_{ji} = 1$

2.4.2 Distance between data point and prototypes

If dissimilarity matrix D is Euclidean then, the distance used in the (2.3) is calculated as,

$$d_V^2(v_i, w_j) = \sum_{l=1}^{N_v} \alpha_{jl} \cdot D_{il} - \frac{1}{2} \alpha_j^T \cdot D \cdot \alpha_j \tag{2.11}$$

2.4.3 Distance between prototypes

The distance between prototypes used in calculating the rank (2.5) can be defines as,

$$d_V^2(w_j, w_k) = \sum_{r=1}^{N_v} \sum_{s=1}^{N_v} \langle v_r, v_s \rangle (\alpha_{jr}\alpha_{js} - 2\alpha_{jr}\alpha_{ks} + \alpha_{kr}\alpha_{ks}) \tag{2.12}$$

where, $\langle v_r, v_s \rangle$ is double centering [2] of the data and is defined as,

$$\langle v_r, v_s \rangle = -\frac{1}{2} \left(D_{rs} - \frac{1}{N_v} \sum_{r=1}^{N_v} D_{rs} - \frac{1}{N_v} \sum_{s=1}^{N_v} D_{rs} + \frac{1}{N_v^2} \sum_{r=1}^{N_v} \sum_{s=1}^{N_v} D_{rs} \right) \tag{2.13}$$

2.4.4 Prototype update

The update of the real prototype is done indirectly by updating the coefficients using the equation[3],

$$\Delta\alpha_{pq} \propto -\sum_{i=1}^{N_v} \beta_{ip} \sum_{l=1}^{N_p} h_\sigma(w_p, w_l) \left(\mathbf{D}_i - \mathbf{D}_q \sum_{k=1}^{N_v} \alpha_{pk} \right) \tag{2.14}$$

Every time coefficient is updated there is a need to normalise α_{pk} so that it sums to 1.

2.5 Median Data

Here prototypes are chosen from the data samples.

$$w_j = v_l \tag{2.15}$$

The prototype which gives least error is selected as the new prototype which is given as,

$$l = \arg\min_{l'} \sum_{i=1}^{N_v} l c_{il'} + \gamma_j \sum_{i=1}^{N_v} (1 - t_{ij})^\eta \tag{2.16}$$

2.6 Pseudo Code

1. Set the number of prototypes N_p.
2. Initialise prototypes:
   - Vectorial: generate random vectors in data space.
   - Relational: initialise random coefficients α_j.
   - Median: select random data samples from the data itself.

   repeat:

3 : calculate fuzzy assignment u_{ij} using (2.6).

4 : calculate typicality t_{ij} using (2.7) .

5 : update probability using u_{ij} and t_{ij} from step 3 and 4
   - Vectorial: calculate w_j using (2.8) .
   - Relational: update coefficients α_j using (2.14).
   - Median: select random data samples from the data itself using (2.16).

   until: Convergence or number of epochs or manual stop.

CHAPTER

THREE

INSTALLATION REQUIREMENTS

3.1 Libraries

The basic requirements to run the code are:

- Minimum python 3.8 version.
- numpy 1.21.0
- matplotlib 3.4.2
- math
- imagio
- os
- shutil
- scipy.spatial.distance
- sklearn

3.2 Execution

- Copy the code.
- Go at the end of the code.
- Change the location of the data and change the number of prototypes.
- Run the code.

CHAPTER

FOUR

VECTORIAL DATA

class Vectorial.**VectorialNeuralGas**(*data*, *epochs*, *generate_plot=False*, *col1=None*, *col2=None*)

Bases: object

Vectorial Neural Gas Clustering uses prototypes to represent the data by small number of vectors which are the points in data space. Each prototype tries to express the distribution of similar data points using the similarity measures. This algorithm uses membership and typicality values to find group with similar features when the data is overlapping. The prototype updates in this program is done explicitly.

updateMemTyp()

This function updates the membership values and typicality values after the every iteration of the prototype update.

class Vectorial.**Vectorial_Data**(*file*, *number_of_prototype*, *use_Columns*)

Bases: object

This class calculates various metric to perform task of Vectorial data clustering.

b

self.gamma is a constant.

beta_ij()

This function calculates the degree of belongingness of a data point to the prototypes.

var second_term: float value which contains value of typicality[i][l] to the power of typicality constant and then multiplied with scalar b.

var first term: float value which contains value of membership[i][l] to the power of membership constant and then multiplied with scalar a.

return: a N x K matrix.

create_animation(*name*)

This function creates an .gif file using the image saved at every epoch.

param name: name to be used to save the .gif file.

data

self.number_of_prototype int value which contains the number of prototype(say K) to represent the data.

distance_prototype(*j*, *l*)

This function calculates the rank of the lth prototype according to j^{th} prototypes.

param j: contains the index of j^{th} prototype. param l: contains the index of lth prototype.

var l: contains the 1 x M vector of lth prototype. var j: contains the 1 x M vector of j^{th} prototype. var rank_j: every time adds 1, when the d(w_l,w_j) - d(w_l, w_k) > 0 else 0.

returns: the rank of lth prototype according to j^{th} prototype. rtype: int

gamma
self.typicality_matrix t_ij is N x K dimension matrix whose values are in range [0,1].

generate_plot(*col1*, *col2*, *filename*)
This function generates plot of the data as an image file.

param col1: first column to be used to plot the data. param col2: second column to be used to plot the data. param filename: file name to be used for the generated image.

var count: iterates through the length of the data. var n: contains the index of the nearest prototype.

generate_vectors()
This function iterates through number of prototypes K, to generate and appends the vector as an array in self.prototypes.

Returns a K x M dimension matrix

get_nearest_n(*point*)
This function iterates through data length and fetches the maximum value of a degree of affinity of a data to different clusters. param point: contains the index of the data.

var nearest: initialises nearest to the first value of Beta_ij as nearest. var nearest_prototype: contains the index of the prototype which is nearest.

return: prototype which is nearest to the data.

initializeMembershipMatrix()
Initializes the random values in range (0,1), with the condition that the sum of elements of any row is 1.

var random_num_list: contains the list of size K with random numbers in range [0.0 , 1.0). var summation: contains the sum of list. var temp_list: contains the list which sums up to 1.

return: a N x K membership matrix.

initializeTypicalityMatrix()
Initializes the random values in range [0,1), but the row wise element sum need not be equal to 1.

var random_number_list: contains the list of size K with random numbers in range [0 , 1).

return: a N x K typicality matrix.

length
self.mem is the hyper-parameter that controls how fuzzy the cluster will be, ranges usually between 1.2 to 2.0.

local_loss(*i*, *j*)
The function calculates the error sum, between data points at i^{th} index and prototypes at j^{th} index.

param i: contains the index of the i^{th} data. param j: contains the index of the j^{th} prototype.

var local_error: a double value which contains the sum of loss between data at i^{th} index and all the prototypes.

return: local loss between the data at index i and prototype at index j

mem
self.typ is the hyper-parameter that controls the degree of typicality values.

membership_matrix
self.generate_vectors generates random vectors.

neighborhood_function(*j*, *l*)
The function calculates range of prototype at j^{th} index in comparison with prototype at lth index using rank.

param j: contains the index of j^{th} prototype. param l: contains the index of lth prototype.

var rank_lj: int value returned by the ranking function. var sigma: a double value. var denominator: a double value. var numerator: int value containing (rank_lj)^2. var rank_neighborhood: double value containing the neighborhood value.

returns: the winner prototype which is needed to be updated. rtype: float

number_of_prototypes
self.length contains the length or the int value of the number rows(N) of the data.

print_dataset()

random_vectors()
This function generates random vector which has same number of column as of the data.

:returns a point of 1 x N dimension

scale()
This function preprocesses the data on one scale.

sig
self.a and self.b are positive independent scalars not necessary to sum up to 1.

sqEuclideanDistance(*i*, *l*)
The function calculates the squared Euclidean Distance between data at index i and prototype at index l.

param i: contains the index of the i^{th} data. param l: contains the index of the lth prototype.

var distance_il: calculates the squared euclidean distance between data at i^{th} and prototype at lth index.

return: distance between the i^{th} data and the lth prototype. rtype: float

typ
self.sig is a constant value used in calculation of neighborhood.

typicality_matrix
self.membership_matrix is N x K dimension matrix whose values are in range [0, 1] and for any i u_ij the values should sum up to 1.

updateMembershipMatrix(*i*, *j*)
This function updates the membership value of the data at index i according to the prototype at index j.

param i: index of the i^{th} data. param j: index of the j^{th} prototype.

var power_m: a float value. var i: contains data at i^{th} index. var j: contains prototype at j^{th} index. var numerator: calculates the numerator part of updating the membership equation. var denominator: calculates the denominator part of updating the membership equation. var div: contains the result after division of the numerator with denominator.

updatePrototype(*j*)
This function updates j^{th} the prototype explicitly as the distance measure chosen is squared Euclidean Distance.

param j: index of the prototype which has to be updated.

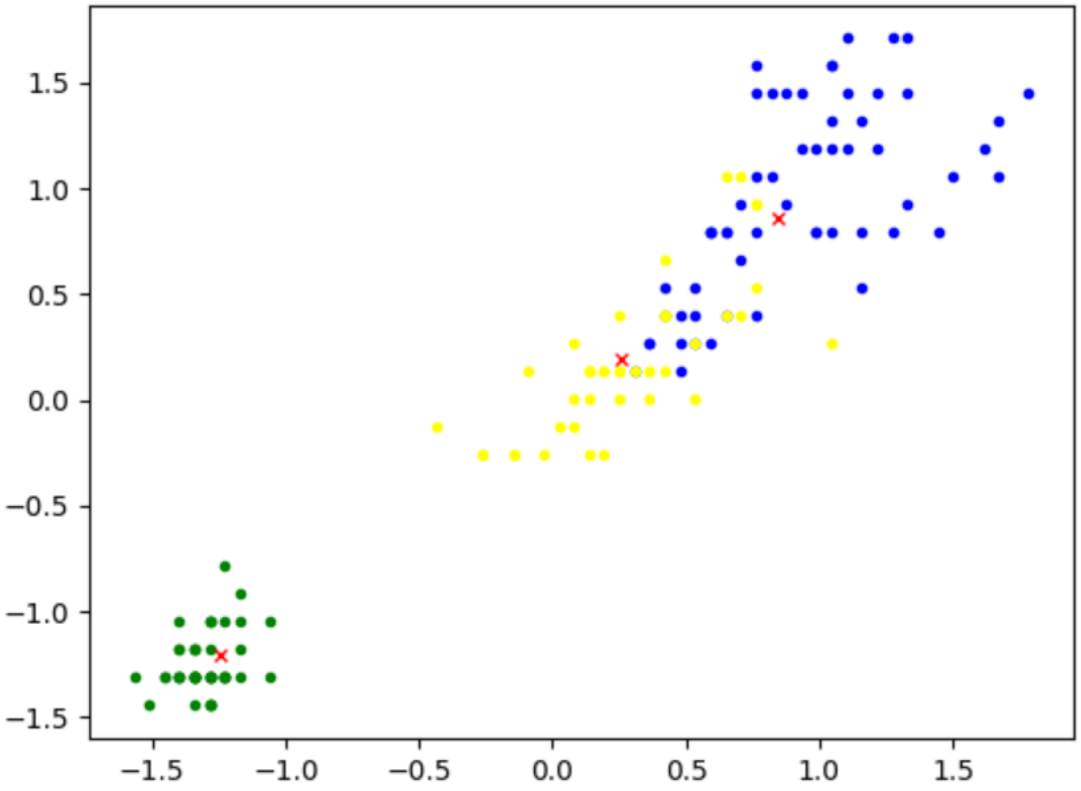
var num: calculates the numerator part of the prototype update equation. var den: calculates the denominator part of the prototype update equation.

updateTypicalityMatrix(*i*, *j*)
This function updates the typicality value of the data at index i according to the prototype at index j.

param i: index of the i^{th} data. param j: index of the j^{th} prototype.

var num: calculates the numerator part of updating the typicality equation. var den: calculates the denominator part of updating the typicality equation var div: contains the result after division of the numerator with denominator.

4.1 Example: Iris Data



CHAPTER

FIVE

RELATIONAL DATA

class Relational.**RelationalNeuralGas**(*data*, *epochs*)

Bases: object

updateMembershipTypicality()

updateMemTyp(): updates the membership values and typicality values after the every iteration of the prototype update.

class Relational.**Relational_Data**(*file*, *number_of_prototype*)

Bases: object

Relational Neural Gas

Here, assumptions are made that there exists a non-linear mapping which projects the data into a Euclidean Space thus, there is a Dissimilarity matrix (interpreted Euclidean Space X_E), which contains the pairwise dissimilarities between the data objects. The matrix D is symmetric, where D_ij = D_ji and D_ii = 0. Also, unlike Vectorial data there is no direct prototype involved but, they are described indirectly by the coefficient Alpha_j so, by updating Alpha virtual prototypes are generated.

CalculateCentering(*data_r*, *data_s*)

Takes the indexes of the data and returns the value at that particular location.

Parameters

- **data_r** – index of Dissimilarity data.
- **data_s** – index of the Dissimilarity data.

Returns Value after double centering the data at index data_r and data_s.

DoubleCentering()

Double centering the data means any row or any column should sum to 0.

beta_ij()

@:var second_term: float value which contains value of typicality[i][j] to the power of typicality constant
and then multiplied with scalar b.

@:var first term: float value which contains value of membership[i][j] to the power of membership constant
and then multiplied with scalar a.

Returns a M x k matrix.

distanceDataNeuron(*i*, *j*)

The distances between the D_i and Alpha_j for Relational data can be calculated using this function.

Parameters

- **i** – index of the Dissimilarity matrix.
- **j** – index of the coefficient Alpha.

Returns Distance between the D_i and Alpha_j.

distance_prototype(*j*, *k*)
Calculates the distance between the Alphas.

Parameters

- **j** – j^{th} index of the Alpha.
- **k** – kth index of the Alpha.

Returns Distance between the Alpha at j and Alpha at k.

generate_coefficient()
This function generates the coefficient Alpha which should range (0,1], with the condition that the sum of every single row should be equal to 1.

Returns k x M matrix

generate_prototype()
The prototypes are updated indirectly using the updated Alpha values.

Returns The matrix of k x M of actual prototype.

initializeMembershipMatrix()
@:var random_num_list: contains the list of size K with random numbers in range [0.0 , 1.0). @:var summation: contains the sum of list. @:var temp_list: contains the list which sums up to 1.

Returns a N x K membership matrix.

initializeTypicalityMatrix()
@:var random_number_list: contains the list of size K with random numbers in range [0.0 , 1.0). :return: a N x K typicality matrix.

local_loss(*i*, *j*)

@:var second_term: float value which contains value of typicality[i][l] to the power of typicality constant and then multiplied with scalar b.

@:var first term: float value which contains value of membership[i][l] to the power of membership constant and then multiplied with scalar a.

Returns a M x k matrix.

neighborhood_function(*j*, *l*)

Parameters

- **j** – contains the index of j^{th} Alpha.
- **l** – contains the index of lth Alpha.

@:var rank_lj: int value returned by the ranking function. @:var sigma: a double value. @:var denominator: a double value. @:var numerator: int value containing (rank_lj)^2. @:var rank_neighborhood: double value containing the neighborhood value.

Returns the neighborhood range according to the winning Alpha.

normaliseAlpha()

Everytime the coefficients are updates, there is a need for normalisation of these coefficient so that the every row sum should be equal to one @:var sum_alpha: takes the sum of row

rank_ij(*j*, *l*)

calculates the rank of the lth Alpha according to j^{th} Alpha.

@:param j: contains the index of j^{th} Alpha. @:param l: contains the index of lth Alpha.

@:var prototype_l: contains the 1 x M vector with index j. @:var prototype_j: contains the 1 x M vector with index l. @:var rank: every time adds 1, when the d(l, j) - d(l, k) > 0 else 0.

@:returns: the rank of lth Alpha according to j^{th} Alpha.

updateMembershipMatrix(*i*, *j*)

updates the membership of the data at index i according to the Alpha at index j.

Parameters

- **i** – index of the i^{th} data.
- **j** – index of the j^{th} Alpha.

@var power_m: float value. @var i: contains data at index i. @var j: contains neuron at index j. @var numerator: calculates the numerator part of updating the membership equation. @var denominator: calculates the denominator part of updating the membership equation. @:var div: contains the result after division of the numerator with denominator.

updateTypicalityMatrix(*i*, *j*)

Parameters

- **i** – index of the i^{th} data.
- **j** – index of the j^{th} neuron.

@var num: calculates the numerator part of updating the typicality equation. @var den: calculates the denominator part of updating the typicality equation @var div: contains the result after division of the numerator with denominator.

update_coefficient(*p*, *q*)

This function takes coefficient matrix and updates the values of the coefficient using the Stochastic gradient descent learning.

Parameters

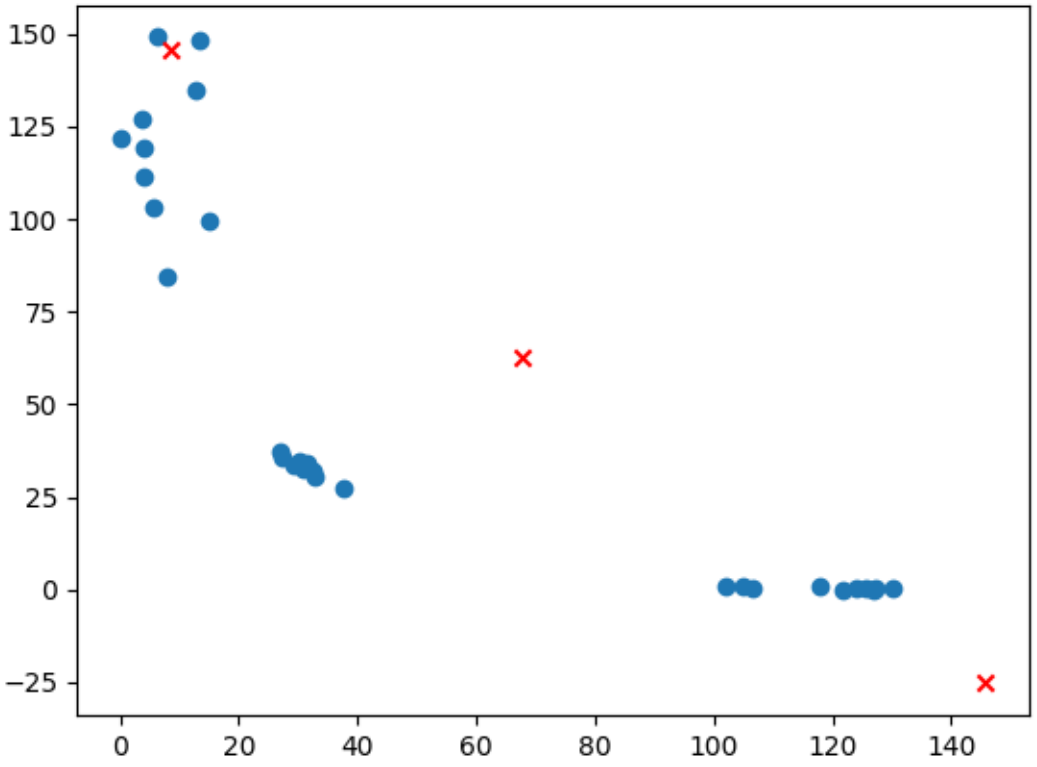
- **q** – index of the qth column of the coefficient matrix.
- **p** – index of the pth row of the coefficient matrix.

visualise_update()

When the coefficients gets updates, this functions visualises where the prototypes are moving.

Returns An image as output which contains scatter plots.

5.1 Example: Random sample from three Gaussian



CHAPTER

SIX

MEDIAN DATA

class Median.**MedianNeuralGas**(*data*, *epochs*)

Bases: object

Median Neural Gas

This is a reserved method in python classes. It is called as a constructor in object oriented terminology. This method is called when an object of Median_Data is created and it allows the class to initialize the attributes of the class.

para self: represents the instance of the class. param file: contains the link where, the data is stored. param number_of_prototype: predefined int type value which is required to represent the data.

updateMembershipTypicality()

This function updates the membership values and typicality values after the every iteration of the prototype update.

class Median.**Median_Data**(*file*, *number_of_prototype*)

Bases: object

This class calculates various metric to perform task of Vectorial data clustering.

CenteringData

"Vector of length N x 1 dimension.

D

Converts the data as an array

Distance

self.number_of_prototype int value which contains the number of prototype k to represent the data.

b

self.gamma is a constant.

beta

Initialisation of Membership and Typicality lists.

beta_ij()

This function calculates the degree of belongingness of a data point to the prototypes.

var second_term: float value which contains value of typicality[i][l] to the power of typicality constant
and then multiplied with scalar b.

var first term: float value which contains value of membership[i][l] to the power of membership constant
and then multiplied with scalar a.

return: a (N-k) x k matrix.

delDistance
self.mem is the hyper-parameter that controls how fuzzy the cluster will be, ranges usually between 1.2 to 2.0.

deleteDistance(*j*)
This function deletes the distances which are now acting as a prototype.

param j: Index of new prototype which is needed to be replaced from dissimilarity matrix.

gamma
initialises an (N-k) x k dimension matrix with zeros.

Type self.beta

generate_prototype()
This function chooses random k rows which act as an initial prototypes.

Returns The matrix of k x M of actual prototype.

initializeMembershipMatrix()
Initializes the random values in range (0,1), with the condition that the sum of elements of any row is 1.

var random_num_list: contains the list of size k with random numbers in range [0.0 , 1.0). var summation: contains the sum of list. var temp_list: contains the list which sums up to 1.

return: a (N-k) x k membership matrix.

initializeTypicalityMatrix()
Initializes the random values in range [0,1), but the row wise element sum need not be equal to 1.

var random_number_list: contains the list of size k with random numbers in range [0 , 1).

return: a (N-k) x k typicality matrix.

lenDist
self.length contains the the length of data excluding prototypes.

length
initialised self.Distance which acts as an array without prototypes.

local_loss(*i*, *j*)
The function calculates the error sum, between data points at i^{th} index and prototypes at j^{th} index.

param i: contains the index of the i^{th} data. param j: contains the index of the j^{th} prototype.

var local_error: a double value which contains the sum of loss between data at i^{th} index and all the prototypes.

return: local loss between the data at index i and prototype at index j

mem
self.typ is the hyper-parameter that controls the degree of typicality values.

membership_matrix
Initialises Centering N x N Matrix

neighborhood_function(*j*, *l*)
The function calculates range of prototype at j^{th} index in comparison with prototype at lth index using rank.

param j: contains the index of j^{th} prototype. param l: contains the index of lth prototype.

var rank_lj: int value returned by the ranking function. var sigma: a double value. var denominator: a double value. var numerator: int value containing (rank_lj)^2. var rank_neighborhood: double value containing the neighborhood value.

returns: the winner prototype which is needed to be updated. rtype: float

newPrototype(*j*, *l_dash*)

This function generates float values which is then used to decide the next prototype using argmin.

param j: Index of prototype to be updated. param l_dash: data length

return: returns M x 1 vector.

number_of_prototypes

self.lenDist contains the length of the Dissimilarity matrix.

rank_ij(*j*, *l*)

calculates the rank of the lth Alpha according to j^{th} Alpha.

param j: contains the index of j^{th} Alpha. param l: contains the index of lth Alpha.

var prototype_l: contains the 1 x M vector with index j. var prototype_j: contains the 1 x M vector with index l. var rank: everytime adds 1, when the d(l, j) - d(l, k) > 0 else 0.

returns: the rank of lth prototype according to j^{th} Alpha.

sig

self.a and self.b are positive independent scalars not necessary to sum up to 1.

sumBetaTyp

Initialises the membership, typicality matrices and generates prototypes

typ

self.sig is a constant value used in calculation of neighborhood.

updateMembershipMatrix(*i*, *j*)

This function updates the membership value of the data at index i according to the prototype at index j.

param i: index of the i^{th} data. param j: index of the j^{th} prototype.

var power_m: a float value. var i: contains data at i^{th} index. var j: contains prototype at j^{th} index. var numerator: calculates the numerator part of updating the membership equation. var denominator: calculates the denominator part of updating the membership equation. var div: contains the result after division of the numerator with denominator.

updateTypicalityMatrix(*i*, *j*)

This function updates the typicality value of the data at index i according to the prototype at index j.

param i: index of the i^{th} data. param j: index of the j^{th} prototype.

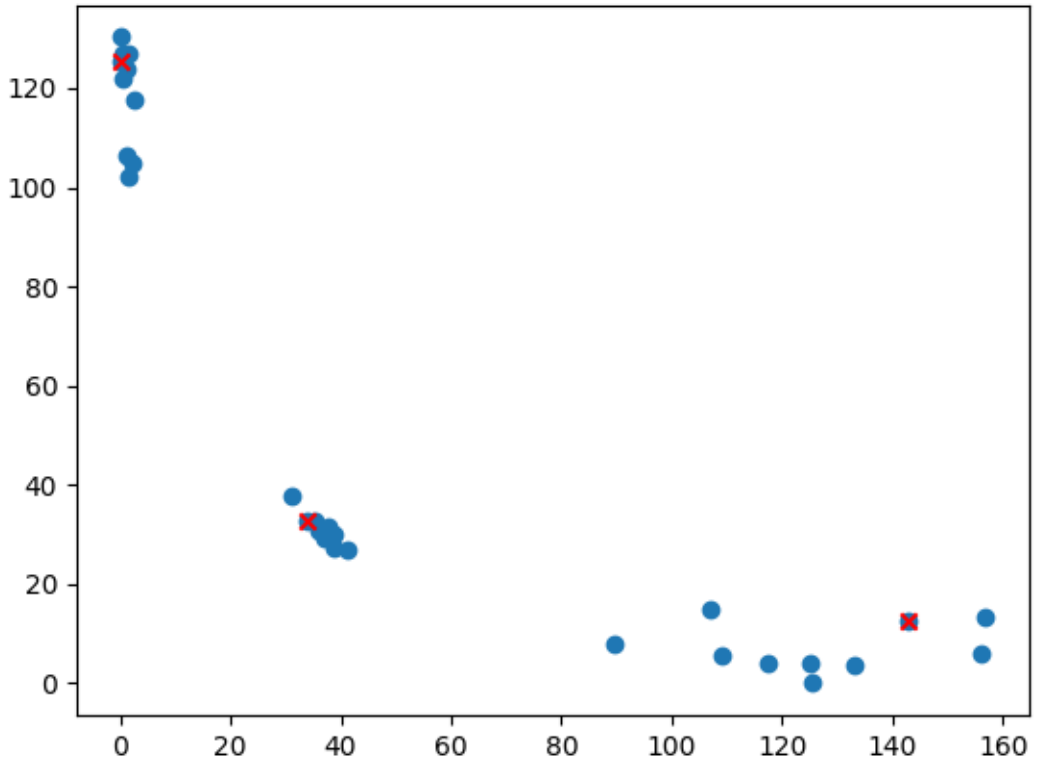
var num: calculates the numerator part of updating the typicality equation. var den: calculates the denominator part of updating the typicality equation var div: contains the result after division of the numerator with denominator.

visualise_update()

This function does the visualisation of movement of prototypes.

Returns An image as output which contains scatter plots.

6.1 Example: Random sample from three Gaussian



BIBLIOGRAPHY

[1] T. Geweniger and T. Villmann, "Variants of fuzzy neural gas," in *Advances in Intelligent Systems and Computing*, pp. 261–270, Springer International Publishing, apr 2019.

[2] B. Hammer and A. Hasenfuss, "Relational neural gas," in *Lecture Notes in Computer Science*, pp. 190–204, Springer Berlin Heidelberg.

[3] T. Geweniger and T. Villmann, "Relational and median variants of possibilistic fuzzy c-means," in *2017 12th International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)*, IEEE, jun 2017.