PyClustering

Generated by Doxygen 1.8.10

Wed Jul 15 2015 12:09:23

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Namespace Documentation

4.1 pyclustering.cluster Namespace Reference

pyclustering module for cluster analysis.

Namespaces

- agglomerative

    Cluster analysis algorithm: agglomerative algorithm.
- birch

    Cluster analysis algorithm: BIRCH.
- cure

    Cluster analysis algorithm: CURE.
- dbscan

    Cluster analysis algorithm: DBSCAN.
- hsyncnet

    Cluster analysis algorithm: Hierarchical Sync (HSyncNet)
- kmeans

    Cluster analysis algorithm: K-Means.
- kmedians

    Cluster analysis algorithm: K-Medians.
- kmedoids

    Cluster analysis algorithm: K-Medoids (PAM - Partitioning Around Medoids).
- optics

    Cluster analysis algorithm: OPTICS (Ordering Points To Identify Clustering Structure)
- rock

    Cluster analysis algorithm: ROCK.
- syncnet

    Cluster analysis algorithm: Sync.
- syncsom

    Cluster analysis algorithm: SYNC-SOM.
- xmeans

    Cluster analysis algorithm: X-Means.

Classes

- class canvas_cluster_descr

  Description of cluster for representation on canvas.
- class cluster_visualizer

  Common visualizer of clusters on 2D or 3D surface.

4.1.1 Detailed Description

pyclustering module for cluster analysis.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.2 pyclustering.cluster.agglomerative Namespace Reference

Cluster analysis algorithm: agglomerative algorithm.

Classes

- class agglomerative

  Class represents agglomerative algorithm for cluster analysis.
- class type_link

  Enumerator of types of link between clusters.

4.2.1 Detailed Description

Cluster analysis algorithm: agglomerative algorithm.

Implementation based on book:

- K.Anil, J.C.Dubes, R.C.Dubes. Algorithms for Clustering Data. 1988.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.3 pyclustering.cluster.birch Namespace Reference

Cluster analysis algorithm: BIRCH.

Classes

- class birch
  Class represents clustering algorithm BIRCH.

4.3.1 Detailed Description

Cluster analysis algorithm: BIRCH.

Implementation based on article:

- T.Zhang, R.Ramakrishnan, M.Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. 1996.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.4 pyclustering.cluster.cure Namespace Reference

Cluster analysis algorithm: CURE.

Classes

- class cure
  Class represents clustering algorithm CURE.
- class cure_cluster
  Represents data cluster in CURE term.

4.4.1 Detailed Description

Cluster analysis algorithm: CURE.

Implementation based on article:

- S.Guha, R.Rastogi, K.Shim. CURE: An Efficient Clustering Algorithm for Large Databases. 1998.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

4.5 pyclustering.cluster.dbscan Namespace Reference

Cluster analysis algorithm: DBSCAN.

Classes

- class dbscan

  Class represents clustering algorithm DBSCAN.

4.5.1 Detailed Description

Cluster analysis algorithm: DBSCAN.

Implementation based on article:

- M.Ester, H.Kriegel, J.Sander, X.Xiaowei. A density-based algorithm for discovering clusters in large spatial databases with noise. 1996.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

4.6 pyclustering.cluster.hsyncnet Namespace Reference

Cluster analysis algorithm: Hierarchical Sync (HSyncNet)

Classes

- class hsyncnet

  Class represents clustering algorithm HSyncNet.

4.6.1 Detailed Description

Cluster analysis algorithm: Hierarchical Sync (HSyncNet)

Based on article description:

- J.Shao, X.He, C.Bohm, Q.Yang, C.Plant. Synchronization-Inspired Partitioning and Hierarchical Clustering. 2013.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.7 pyclustering.cluster.kmeans Namespace Reference

Cluster analysis algorithm: K-Means.

Classes

- class kmeans

  Class represents clustering algorithm K-Means.

4.7.1 Detailed Description

Cluster analysis algorithm: K-Means.

Based on book description:

- J.B.MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. 1967.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.8 pyclustering.cluster.kmedians Namespace Reference

Cluster analysis algorithm: K-Medians.

Classes

- class kmedians

    Class represents clustering algorithm K-Medians.

4.8.1 Detailed Description

Cluster analysis algorithm: K-Medians.

Based on book description:

- A.K. Jain, R.C Dubes, Algorithms for Clustering Data. 1988.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.9 pyclustering.cluster.kmedoids Namespace Reference

Cluster analysis algorithm: K-Medoids (PAM - Partitioning Around Medoids).

Classes

- class kmedoids

    Class represents clustering algorithm K-Medoids (another one title is PAM - Parti).

4.9.1 Detailed Description

Cluster analysis algorithm: K-Medoids (PAM - Partitioning Around Medoids).

Based on book description:

- A.K. Jain, R.C Dubes, Algorithms for Clustering Data. 1988.
- L. Kaufman, P.J. Rousseeuw, Finding Groups in Data: an Introduction to Cluster Analysis. 1990.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.10 pyclustering.cluster.optics Namespace Reference

Cluster analysis algorithm: OPTICS (Ordering Points To Identify Clustering Structure)

Classes

- class optics

  Class represents clustering algorithm OPTICS (Ordering Points To Identify Clustering Structure).
- class optics_descriptor

  Object description that used by OPTICS algorithm for cluster analysis.

4.10.1 Detailed Description

Cluster analysis algorithm: OPTICS (Ordering Points To Identify Clustering Structure)

Based on article description:

- M.Ankerst, M.Breunig, H.Kriegel, J.Sander. OPTICS: Ordering Points To Identify the Clustering Structure. 1999.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.11 pyclustering.cluster.rock Namespace Reference

Cluster analysis algorithm: ROCK.

Classes

- class rock

  Class represents clustering algorithm ROCK.

4.11.1 Detailed Description

Cluster analysis algorithm: ROCK.

Based on article description:

- S.Guha, R.Rastogi, K.Shim. ROCK: A Robust Clustering Algorithm for Categorical Attributes. 1999.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.12 pyclustering.cluster.syncnet Namespace Reference

Cluster analysis algorithm: Sync.

Classes

- class syncnet

  Class represents clustering algorithm SyncNet.
- class syncnet_analyser

  Performs analysis of output dynamic of the oscillatory network syncnet to extract information about cluster allocation.

4.12.1 Detailed Description

Cluster analysis algorithm: Sync.

Based on article description:

- T.Miyano, T.Tsutsui. Data Synchronization as a Method of Data Mining. 2007.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.13 pyclustering.cluster.syncsom Namespace Reference

Cluster analysis algorithm: SYNC-SOM.

Classes

- class syncsom

  Class represents clustering algorithm SYNC-SOM.

4.13.1 Detailed Description

Cluster analysis algorithm: SYNC-SOM.

Based on article description:

- A.Novikov, E.Benderskaya. SYNC-SOM Double-layer Oscillatory Network for Cluster Analysis. 2014.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.14 pyclustering.cluster.xmeans Namespace Reference

Cluster analysis algorithm: X-Means.

Classes

- class splitting_type

  Enumeration of splitting types that can be used as splitting creation of cluster in X-Means algorithm.
- class xmeans

  Class represents clustering algorithm X-Means.

4.14.1 Detailed Description

Cluster analysis algorithm: X-Means.

Based on article description:

- D.Pelleg, A.Moore. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. 2000.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.15 pyclustering.container Namespace Reference

pyclustering module of data structures (containers).

Namespaces

- cftree
  Data Structure: CF-Tree.
- kdtree
  Data Structure: KD-Tree.

4.15.1 Detailed Description

pyclustering module of data structures (containers).

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

4.16 pyclustering.container.cftree Namespace Reference

Data Structure: CF-Tree.

Classes

- class cfentry
  Clustering feature representation.
- class cfnode
  Representation of node of CF-Tree.
- class cfnode_type
  Enumeration of CF-Node types that are used by CF-Tree.
- class cftree
  CF-Tree representation.
- class leaf_node
  Represents clustering feature leaf node.
- class measurement_type
  Enumeration of measurement types for CF-Tree.
- class non_leaf_node
  Representation of clustering feature non-leaf node.

4.16.1 Detailed Description

Data Structure: CF-Tree.

Based on book description:

- M.Zhang, R.Ramakrishnan, M.Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. 1996.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.17 pyclustering.container.kdtree Namespace Reference

Data Structure: KD-Tree.

Classes

- class kdtree

  Represents KD Tree.
- class node

  Represents node of KD-Tree.

4.17.1 Detailed Description

Data Structure: KD-Tree.

Based on book description:

- M.Samet. The Design And Analysis Of Spatial Data Structures. 1994.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.18 pyclustering.gcolor Namespace Reference

pyclustering module for graph coloring.

Namespaces

- dsatur

  Graph coloring algorithm: DSATUR.
- hysteresis

  Graph coloring algorithm: Algorithm based on Hysteresis Oscillatory Network.
- sync

  Graph coloring algorithm based on Sync Oscillatory Network.

4.18.1 Detailed Description

pyclustering module for graph coloring.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.19 pyclustering.gcolor.dsatur Namespace Reference

Graph coloring algorithm: DSATUR.

Classes

- class dsatur

  Represents DSATUR algorithm for graph coloring problem that uses greedy strategy.

4.19.1 Detailed Description

Graph coloring algorithm: DSATUR.

Based on article description:

- D.Brelaz. New Methods to color the vertices of a graph. 1979.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.20 pyclustering.gcolor.hysteresis Namespace Reference

Graph coloring algorithm: Algorithm based on Hysteresis Oscillatory Network.

Classes

- class hysteresisgcolor

  Class represents graph coloring algorithm based on hysteresis oscillatory network.

4.20.1 Detailed Description

Graph coloring algorithm: Algorithm based on Hysteresis Oscillatory Network.

Based on article description:

- K.Jinno, H.Taguchi, T.Yamamoto, H.Hirose. Dynamical Hysteresis Neural Network for Graph Coloring Problem. 2003.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.21 pyclustering.gcolor.sync Namespace Reference

Graph coloring algorithm based on Sync Oscillatory Network.

Classes

- class syncgcolor

  Oscillatory network based on Kuramoto model with negative and positive connections for graph coloring problem.
- class syncgcolor_analyser

  Analyser of output dynamic of the oscillatory network syncgcolor.

4.21.1 Detailed Description

Graph coloring algorithm based on Sync Oscillatory Network.

Based on article description:

- J.Wu J, L.Jiao, W.Chen. Clustering dynamics of nonlinear oscillator network: Application to graph coloring problem. 2011.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.22 pyclustering.nnet Namespace Reference

Neural and oscillatory network module.

Namespaces

- hhn

  Oscillatory Neural Network based on Hodgkin-Huxley Neuron Model.
- hysteresis

  Neural Network: Hysteresis Oscillatory Network.
- legion

  Neural Network: Local Excitatory Global Inhibitory Oscillatory Network (LEGION)
- pcnn

  Neural Network: Pulse Coupled Neural Network.
- som

  Neural Network: Self-Organized Feature Map.
- sync

  Neural Network: Oscillatory Neural Network based on Kuramoto model.

Classes

- class conn_represent

  Enumerator of internal network connection representation between oscillators.
- class conn_type

  Enumerator of connection types between oscillators.
- class initial_type

  Enumerator of types of oscillator output initialization.
- class network

  Common network description.
- class solve_type

  Enumerator of solver types that are used for network simulation.

4.22.1 Detailed Description

Neural and oscillatory network module.

Consists of models of bio-inspired networks.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.23 pyclustering.nnet.hhn Namespace Reference

Oscillatory Neural Network based on Hodgkin-Huxley Neuron Model.

Classes

- class central_element

  Central element consist of two central neurons that are described by a little bit different dynamic than peripheral.
- class hhn_network

  Oscillatory Neural Network with central element based on Hodgkin-Huxley neuron model.
- class hhn_parameters

  Describes parameters of Hodgkin-Huxley Oscillatory Network.

4.23.1 Detailed Description

Oscillatory Neural Network based on Hodgkin-Huxley Neuron Model.

Based on article description:

- D.Chik, R.Borisyuk, Y.Kazanovich. Selective attention model with spiking elements. 2009.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.24 pyclustering.nnet.hysteresis Namespace Reference

Neural Network: Hysteresis Oscillatory Network.

Classes

- class hysteresis_network

  Hysteresis oscillatory network that uses relaxation oscillators.

4.24.1 Detailed Description

Neural Network: Hysteresis Oscillatory Network.

Based on article description:

- K.Jinno. Oscillatory Hysteresis Associative Memory. 2002.
- K.Jinno, H.Taguchi, T.Yamamoto, H.Hirose. Dynamical Hysteresis Neural Network for Graph Coloring Problem. 2003.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.25 pyclustering.nnet.legion Namespace Reference

Neural Network: Local Excitatory Global Inhibitory Oscillatory Network (LEGION)

Classes

- class legion_dynamic

  Represents output dynamic of LEGION.
- class legion_network

  Local excitatory global inhibitory oscillatory network (LEGION) that uses relaxation oscillator based on Van der Pol model.
- class legion_parameters

  Describes parameters of LEGION.

4.25.1 Detailed Description

Neural Network: Local Excitatory Global Inhibitory Oscillatory Network (LEGION)

Based on article description:

- D.Wang, D.Terman. Image Segmentation Based on Oscillatory Correlation. 1997.
- D.Wang, D.Terman. Locally Excitatory Globally Inhibitory Oscillator Networks. 1995.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.26 pyclustering.nnet.pcnn Namespace Reference

Neural Network: Pulse Coupled Neural Network.

Classes

- class pcnn_dynamic

    Represents output dynamic of PCNN.
- class pcnn_network

    Model of oscillatory network that is based on the Eckhorn model.
- class pcnn_parameters

    Parameters for pulse coupled neural network.
- class pcnn_visualizer

    Visualizer of output dynamic of pulse-coupled neural network (PCNN).

4.26.1 Detailed Description

Neural Network: Pulse Coupled Neural Network.

Based on book description:

- T.Lindblad, J.M.Kinser. Image Processing Using Pulse-Coupled Neural Networks (2nd edition). 2005.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.27 pyclustering.nnet.som Namespace Reference

Neural Network: Self-Organized Feature Map.

Classes

- class som

    Represents self-organized feature map (SOM).
- class som_parameters

    Represents SOM parameters.
- class type_conn

    Enumeration of connection types for SOM.
- class type_init

    Enumeration of initialization types for SOM.

4.27.1 Detailed Description

Neural Network: Self-Organized Feature Map.

Based on article description:

- T.Kohonen. The Self-Organizing Map. 1990.
- T.Kohonen, E.Oja, O.Simula, A.Visa, J.Kangas. Engineering Applications of the Self-Organizing Map. 1996.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.28 pyclustering.nnet.sync Namespace Reference

Neural Network: Oscillatory Neural Network based on Kuramoto model.

Classes

- class sync_dynamic

  Represents output dynamic of Sync.
- class sync_network

  Model of oscillatory network that is based on the Kuramoto model of synchronization.
- class sync_visualizer

  Visualizer of output dynamic of sync network (Sync).

4.28.1 Detailed Description

Neural Network: Oscillatory Neural Network based on Kuramoto model.

Based on article description:

- A.Arenas, Y.Moreno, C.Zhou. Synchronization in complex networks. 2008.
- X.B.Lu. Adaptive Cluster Synchronization in Coupled Phase Oscillators. 2009.
- X.Lou. Adaptive Synchronizability of Coupled Oscillators With Switching. 2012.
- A.Novikov, E.Benderskaya. Oscillatory Neural Networks Based on the Kuramoto Model. 2014.

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.29 pyclustering.utils Namespace Reference

Utils that are used by modules of pyclustering.

Namespaces

- graph

    Graph representation (uses format GRPR).

Functions

- def read_sample (filename)

    Returns sample for cluster analysis.
- def read_image (filename)

    Returns image as N-dimension (depends on the input image) matrix, where one element of list describes pixel.
- def rgb2gray (image_rgb_array)

    Returns image as 1-dimension (gray colored) matrix, where one element of list describes pixel.
- def average_neighbor_distance (points, num_neigh)

    Returns average distance for establish links between specified number of neighbors.
- def geometric_median

    Calculate geometric median of input set of points using Euclidian distance.
- def euclidean_distance (a, b)

    Calculate Euclidian distance between vector a and b.
- def euclidean_distance_sqrt (a, b)

    Calculate square Euclidian distance between vector a and b.
- def manhattan_distance (a, b)

    Calculate Manhattan distance between vector a and b.
- def average_inter_cluster_distance

    Calculates average inter-cluster distance between two clusters.
- def average_intra_cluster_distance

    Calculates average intra-cluster distance between two clusters.
- def variance_increase_distance (cluster1, cluster2, data)

    Calculates variance increase distance between two clusters.
- def heaviside (value)

    Calculates Heaviside function that represents step function.
- def timedcall (executable_function, args)

    Executes specified method or function with measuring of execution time.
- def extract_number_oscillations

    Extracts number of oscillations of specified oscillator.
- def allocate_sync_ensembles

    Allocate clusters in line with ensembles of synchronous oscillators where each synchronous ensemble corresponds to only one cluster.
- def draw_clusters

    Displays clusters for data in 2D or 3D.
- def draw_dynamics

    Draw dynamics of neurons (oscillators) in the network.
- def set_ax_param

    Sets parameters for matplotlib ax.
- def draw_dynamics_set

    Draw lists of dynamics of neurons (oscillators) in the network.

- def draw_image_color_segments

    Shows image segments using colored image.
- def draw_image_mask_segments

    Shows image segments using black masks.
- def linear_sum (list_vector)

    Calculates linear sum of vector that is represented by list, each element can be represented by list - multidimensional elements.
- def square_sum (list_vector)

    Calculates square sum of vector that is represented by list, each element can be represented by list - multidimensional elements.
- def list_math_subtraction (a, b)

    Calculates subtraction of two lists.
- def list_math_substraction_number (a, b)

    Calculates subtraction between list and number.
- def list_math_addition (a, b)

    Addition of two lists.
- def list_math_addition_number (a, b)

    Addition between list and number.
- def list_math_division_number (a, b)

    Division between list and number.
- def list_math_division (a, b)

    Division of two lists.
- def list_math_multiplication_number (a, b)

    Multiplication between list and number.
- def list_math_multiplication (a, b)

    Multiplication of two lists.

4.29.1 Detailed Description

Utils that are used by modules of pyclustering.

Authors

Andrei Novikov (`pyclustering@yandex.ru`)

Date

2014-2015

Copyright

GNU Public License

4.29.2 Function Documentation

4.29.2.1 def pyclustering.utils.allocate_sync_ensembles (*dynamic*, *tolerance* = `0.1`, *threshold* = `1.0`, *ignore* = `None`)

Allocate clusters in line with ensembles of synchronous oscillators where each synchronous ensemble corresponds to only one cluster.

Parameters

in	*dynamic*	(dynamic): Dynamic of each oscillator.
in	*tolerance*	(double): Maximum error for allocation of synchronous ensemble oscillators.
in	*threshold*	(double): Amlitude trigger when spike is taken into account.
in	*ignore*	(bool): Set of indexes that shouldn't be taken into account.

Returns

(list) Grours (lists) of indexes of synchronous oscillators, for example, [[index_osc1, index_osc3], [index_↵ osc2], [index_osc4, index_osc5]].

4.29.2.2 def pyclustering.utils.average_inter_cluster_distance (*cluster1*, *cluster2*, *data* = None)

Calculates average inter-cluster distance between two clusters.

Clusters can be represented by list of coordinates (in this case data shouldn't be specified), or by list of indexes of points from the data (represented by list of points), in this case data should be specified.

Parameters

in	*cluster1*	(list): The first cluster.
in	*cluster2*	(list): The second cluster.
in	*data*	(list): If specified than elements of clusters will be used as indexes, otherwise elements of cluster will be considered as points.

Returns

(double) Average inter-cluster distance between two clusters.

4.29.2.3 def pyclustering.utils.average_intra_cluster_distance (*cluster1*, *cluster2*, *data* = None)

Calculates average intra-cluster distance between two clusters.

Clusters can be represented by list of coordinates (in this case data shouldn't be specified), or by list of indexes of points from the data (represented by list of points), in this case data should be specified.

Parameters

in	*cluster1*	(list): The first cluster.
in	*cluster2*	(list): The second cluster.
in	*data*	(list): If specified than elements of clusters will be used as indexes, otherwise elements of cluster will be considered as points.

Returns

(double) Average intra-cluster distance between two clusters.

4.29.2.4 def pyclustering.utils.average_neighbor_distance (*points*, *num_neigh*)

Returns average distance for establish links between specified number of neighbors.

Parameters

in	*points*	(list): Input data, list of points where each point represented by list.
in	*num_neigh*	(uint): Number of neighbors that should be used for distance calculation.

Returns

(double) Average distance for establish links between 'num_neigh' in data set 'points'.

4.29.2.5 def pyclustering.utils.draw_clusters (*data*, *clusters*, *noise* = `[]`, *marker_descr* = `'.'`, *hide_axes* = `False`)

Displays clusters for data in 2D or 3D.

Parameters

in	*data*	(list): Points that are described by coordinates represented.
in	*clusters*	(list): Clusters that are represented by lists of indexes where each index corresponds to point in data.
in	*noise*	(list): Points that are regarded to noise.
in	*marker_descr*	(string): Marker for displaying points.
in	*hide_axes*	(bool): If True - axes is not displayed.

4.29.2.6 def pyclustering.utils.draw_dynamics (*t*, *dyn*, *x_title* = `None`, *y_title* = `None`, *x_lim* = `None`, *y_lim* = `None`, *x_labels* = `True`, *y_labels* = `True`, *separate* = `False`, *axes* = `None`)

Draw dynamics of neurons (oscillators) in the network.

It draws if matplotlib is not specified (None), othewise it should be performed manually.

Parameters

in	*t*	(list): Values of time (used by x axis).
in	*dyn*	(list): Values of output of oscillators (used by y axis).
in	*x_title*	(string): Title for Y.
in	*y_title*	(string): Title for X.
in	*x_lim*	(double): X limit.
in	*y_lim*	(double): Y limit.
in	*x_labels*	(bool): If True - shows X labels.
in	*y_labels*	(bool): If True - shows Y labels.
in	*separate*	(list): Consists of lists of oscillators where each such list consists of oscillator indexes that will be shown on separated stage.
in	*axes*	(ax): If specified then matplotlib axes will be used for drawing and plot will not be shown.

Returns

(ax) Axes of matplotlib.

4.29.2.7 def pyclustering.utils.draw_dynamics_set (*dynamics*, *xtitle* = `None`, *ytitle* = `None`, *xlim* = `None`, *ylim* = `None`, *xlabels* = `False`, *ylabels* = `False`)

Draw lists of dynamics of neurons (oscillators) in the network.

Parameters

in	*dynamics*	(list): List of network outputs that are represented by values of output of oscillators (used by y axis).
in	*xtitle*	(string): Title for Y.
in	*ytitle*	(string): Title for X.
in	*xlim*	(double): X limit.
in	*ylim*	(double): Y limit.
in	*xlabels*	(bool): If True - shows X labels.
in	*ylabels*	(bool): If True - shows Y labels.

4.29.2.8 def pyclustering.utils.draw_image_color_segments (*source*, *clusters*, *hide_axes* = `True`)

Shows image segments using colored image.

Each color on result image represents allocated segment. The first image is initial and other is result of segmentation.

Parameters

in	*source*	(string): Path to image.
in	*clusters*	(list): List of clusters (allocated segments of image) where each cluster consists of indexes of pixel from source image.
in	*hide_axes*	(bool): If True then axes will not be displayed.

4.29.2.9 def pyclustering.utils.draw_image_mask_segments (*source*, *clusters*, *hide_axes* = `True`)

Shows image segments using black masks.

Each black mask of allocated segment is presented on separate plot. The first image is initial and others are black masks of segments.

Parameters

in	*source*	(string): Path to image.
in	*clusters*	(list): List of clusters (allocated segments of image) where each cluster consists of indexes of pixel from source image.
in	*hide_axes*	(bool): If True then axes will not be displayed.

4.29.2.10 def pyclustering.utils.euclidean_distance (*a*, *b*)

Calculate Euclidian distance between vector a and b.

Parameters

in	*a*	(list): The first vector.
in	*b*	(list): The second vector.

Returns

(double) Euclidian distance between two vectors.

Note

This function for calculation is faster then standard function in ~100 times!

4.29.2.11 def pyclustering.utils.euclidean_distance_sqrt (*a*, *b*)

Calculate square Euclidian distance between vector a and b.

Parameters

in	*a*	(list): The first vector.
in	*b*	(list): The second vector.

Returns

(double) Square Euclidian distance between two vectors.

4.29.2.12 def pyclustering.utils.extract_number_oscillations (*osc_dyn*, *index* = `0`, *amplitude_threshold* = `1.0`)

Extracts number of oscillations of specified oscillator.

Parameters

in	*osc_dyn*	(list): Dynamic of oscillators.
in	*index*	(uint): Index of oscillator in dynamic.
in	*amplitude_↩ threshold*	(double): Amplitude threshold, when oscillator value is greater than threshold then oscillation is incremented.

Returns

(uint) Number of oscillations of specified oscillator.

4.29.2.13 def pyclustering.utils.geometric_median (*points*, *indexes* = `None`)

Calculate geometric median of input set of points using Euclidian distance.

Parameters

in	*points*	(list): Set of points for median calculation.
in	*indexes*	(list): Indexes of objects in input set of points that will be taken into account during median calculation.

Returns

(uint) index of point in input set that corresponds to median.

4.29.2.14 def pyclustering.utils.heaviside (*value*)

Calculates Heaviside function that represents step function.

If input value is greater than 0 then returns 1, otherwise returns 0.

Parameters

in	*value*	(double): Argument of Heaviside function.

Returns

(double) Value of Heaviside function.

4.29.2.15 def pyclustering.utils.linear_sum (*list_vector*)

Calculates linear sum of vector that is represented by list, each element can be represented by list - multidimensional elements.

Parameters

in	*list_vector*	(list): Input vector.

Returns

(list|double) Linear sum of vector that can be represented by list in case of multidimensional elements.

4.29.2.16 def pyclustering.utils.list_math_addition (*a*, *b*)

Addition of two lists.

Each element from list 'a' is added to element from list 'b' accordingly.

Parameters

in	*a*	(list): List of elements that supports mathematic addition..
in	*b*	(list): List of elements that supports mathematic addition..

Returns

(list) Results of addtion of two lists.

4.29.2.17 def pyclustering.utils.list_math_addition_number (*a*, *b*)

Addition between list and number.

Each element from list 'a' is added to number 'b'.

Parameters

in	*a*	(list): List of elements that supports mathematic addition.
in	*b*	(double): Value that supports mathematic addition.

Returns

(list) Result of addtion of two lists.

4.29.2.18 def pyclustering.utils.list_math_division (*a*, *b*)

Division of two lists.

Each element from list 'a' is divided by element from list 'b' accordingly.

Parameters

in	*a*	(list): List of elements that supports mathematic division.
in	*b*	(list): List of elements that supports mathematic division.

Returns

(list) Result of division of two lists.

4.29.2.19 def pyclustering.utils.list_math_division_number (*a*, *b*)

Division between list and number.

Each element from list 'a' is divided by number 'b'.

Parameters

in	*a*	(list): List of elements that supports mathematic division.
in	*b*	(double): Value that supports mathematic division.

Returns

(list) Result of division between list and number.

4.29.2.20 def pyclustering.utils.list_math_multiplication (*a*, *b*)

Multiplication of two lists.

Each element from list 'a' is multiplied by element from list 'b' accordingly.

Parameters

in	*a*	(list): List of elements that supports mathematic multiplication.
in	*b*	(double): Number that supports mathematic multiplication.

Returns

(list) Result of multiplication between list and number.

4.29.2.21 def pyclustering.utils.list_math_multiplication_number (*a*, *b*)

Multiplication between list and number.

Each element from list 'a' is multiplied by number 'b'.

Parameters

in	*a*	(list): List of elements that supports mathematic division.
in	*b*	(double): Number that supports mathematic division.

Returns

(list) Result of division between list and number.

4.29.2.22 def pyclustering.utils.list_math_substraction_number (*a*, *b*)

Calculates subtraction between list and number.

Each element from list 'a' is subtracted by number 'b'.

Parameters

in	*a*	(list): List of elements that supports mathematical subtraction.
in	*b*	(list): Value that supports mathematical subtraction.

Returns

(list) Results of subtraction between list and number.

4.29.2.23 def pyclustering.utils.list_math_subtraction (*a*, *b*)

Calculates subtraction of two lists.

Each element from list 'a' is subtracted by element from list 'b' accordingly.

Parameters

`in`	*a*	(list): List of elements that supports mathematical subtraction.
`in`	*b*	(list): List of elements that supports mathematical subtraction.

Returns

(list) Results of subtraction of two lists.

4.29.2.24 def pyclustering.utils.manhattan_distance (*a, b*)

Calculate Manhattan distance between vector a and b.

Parameters

`in`	*a*	(list): The first vector.
`in`	*b*	(list): The second vector.

Returns

(double) Manhattan distance between two vectors.

4.29.2.25 def pyclustering.utils.read_image (*filename*)

Returns image as N-dimension (depends on the input image) matrix, where one element of list describes pixel.

Parameters

`in`	*filename*	(string): Path to image.

Returns

(list) Pixels where each pixel described by list of RGB-values.

4.29.2.26 def pyclustering.utils.read_sample (*filename*)

Returns sample for cluster analysis.

Parameters

`in`	*filename*	(string): Path to file with data for cluster analysis.

Returns

(list) Points where each point represented by list of coordinates.

4.29.2.27 def pyclustering.utils.rgb2gray (*image_rgb_array*)

Returns image as 1-dimension (gray colored) matrix, where one element of list describes pixel.

Luma coding is used for transformation.

Parameters

in	*image_rgb_array*	(list): Image represented by RGB list.

Returns

(list) Image as gray colored matrix, where one element of list describes pixel.

```
1 colored_image = read_image(file_name);
2 gray_image = rgb2gray(colored_image);
```

See also

read_image()

4.29.2.28 def pyclustering.utils.set_ax_param (*ax*, *x_title* = None, *y_title* = None, *x_lim* = None, *y_lim* = None, *x_labels* = True, *y_labels* = True, *grid* = True)

Sets parameters for matplotlib ax.

Parameters

in	*ax*	(Axes): Axes for which parameters should applied.
in	*x_title*	(string): Title for Y.
in	*y_title*	(string): Title for X.
in	*x_lim*	(double): X limit.
in	*y_lim*	(double): Y limit.
in	*x_labels*	(bool): If True - shows X labels.
in	*y_labels*	(bool): If True - shows Y labels.
in	*grid*	(bool): If True - shows grid.

4.29.2.29 def pyclustering.utils.square_sum (*list_vector*)

Calculates square sum of vector that is represented by list, each element can be represented by list - multidimensional elements.

Parameters

in	*list_vector*	(list): Input vector.

Returns

(double) Square sum of vector.

4.29.2.30 def pyclustering.utils.timedcall (*executable_function*, *args*)

Executes specified method or function with measuring of execution time.

Parameters

in	*executable_function*	(pointer): Pointer to function or method.

in	*args*	(∗): Arguments of called function or method.

Returns

(tuple) Execution time and result of execution of function or method (execution_time, result_execution).

4.29.2.31 def pyclustering.utils.variance_increase_distance (*cluster1*, *cluster2*, *data*)

Calculates variance increase distance between two clusters.

Clusters can be represented by list of coordinates (in this case data shouldn't be specified), or by list of indexes of points from the data (represented by list of points), in this case data should be specified.

Parameters

in	*cluster1*	(list): The first cluster.
in	*cluster2*	(list): The second cluster.
in	*data*	(list): If specified than elements of clusters will be used as indexes, otherwise elements of cluster will be considered as points.

Returns

(double) Average variance increase distance between two clusters.

4.30 pyclustering.utils.graph Namespace Reference

Graph representation (uses format GRPR).

Classes

- class graph

  Graph representation.
- class type_graph_descr

  Enumeration of graph description.

Functions

- def read_graph (filename)

  Read graph from file in GRPR format.
- def draw_graph

  Draw graph.

4.30.1 Detailed Description

Graph representation (uses format GRPR).

Authors

Andrei Novikov (pyclustering@yandex.ru)

Date

2014-2015

Copyright

GNU Public License

4.30.2 Function Documentation

4.30.2.1 def pyclustering.utils.graph.draw_graph (*graph_instance*, *map_coloring* = `None`)

Draw graph.

Parameters

`in`	*graph_instance*	(graph): Graph that should be drawn.
`in`	*map_coloring*	(list): List of color indexes for each vertex. Size of this list should be equal to size of graph (number of vertices). If it's not specified (None) than graph without coloring will be dwarn.

Warning

Graph can be represented if there is space representation for it.

4.30.2.2 def pyclustering.utils.graph.read_graph (*filename*)

Read graph from file in GRPR format.

Parameters

`in`	*filename*	(string): Path to file with graph in GRPR format.

Returns

(graph) Graph that is read from file.

Chapter 5

Class Documentation

5.1 pyclustering.cluster.agglomerative.agglomerative Class Reference

Class represents agglomerative algorithm for cluster analysis.

Public Member Functions

- def __init__ (self, data, number_clusters, link)

  Constructor of clustering algorithm hierarchical.
- def process (self)

  Performs cluster analysis in line with rules of agglomerative algorithm and similarity.
- def get_clusters (self)

  Returns list of allocated clusters, each cluster contains indexes of objects in list of data.

5.1.1 Detailed Description

Class represents agglomerative algorithm for cluster analysis.

Example:

```
# sample for cluster analysis (represented by list)
sample = read_sample(path_to_sample);

# create object that uses python code only
agglomerative_instance = agglomerative(sample, 2, link_type.CENTROID_LINK)

# cluster analysis
agglomerative_instance.process();

# obtain results of clustering
clusters = agglomerative_instance.get_clusters();
```

5.1.2 Constructor & Destructor Documentation

5.1.2.1 def pyclustering.cluster.agglomerative.agglomerative.__init__ (*self, data, number_clusters, link*)

Constructor of clustering algorithm hierarchical.

Parameters

in	*data*	(list): Input data that is presented as a list of points (objects), each point should be represented by a list or tuple.
in	*number_clusters*	(uint): Number of clusters that should be allocated.
in	*link*	(type_link): Link type that is used for calculation similarity between objects and clusters.

5.1.3 Member Function Documentation

5.1.3.1 def pyclustering.cluster.agglomerative.agglomerative.get_clusters (*self*)

Returns list of allocated clusters, each cluster contains indexes of objects in list of data.

Remarks

Results of clustering can be obtained using corresponding gets methods.

Returns

(list) List of allocated clusters, each cluster contains indexes of objects in list of data.

See also

process()

5.1.3.2 def pyclustering.cluster.agglomerative.agglomerative.process (*self*)

Performs cluster analysis in line with rules of agglomerative algorithm and similarity.

See also

get_clusters()

The documentation for this class was generated from the following file:

- pyclustering/cluster/agglomerative.py

5.2 pyclustering.cluster.birch.birch Class Reference

Class represents clustering algorithm BIRCH.

Public Member Functions

- def __init__

  Constructor of clustering algorithm BIRCH.
- def process (self)

  Performs cluster analysis in line with rules of BIRCH algorithm.
- def get_clusters (self)

  Returns list of allocated clusters, each cluster contains indexes of objects in list of data.

5.2.1 Detailed Description

Class represents clustering algorithm BIRCH.

Example:

```
# sample for cluster analysis (represented by list)
sample = read_sample(path_to_sample);

# create object of birch that uses CCORE for processing
birch_instance = birch(sample, 2, 5, 5, 0.05, measurement_type.CENTROID_EUCLIDIAN_DISTANCE, 200, True);

# cluster analysis
birch_instance.process();

# obtain results of clustering
clusters = birch_instance.get_clusters();
```

5.2.2 Constructor & Destructor Documentation

5.2.2.1 def pyclustering.cluster.birch.birch.__init__ (*self*, *data*, *number_clusters*, *branching_factor* = 5, *max_node_entries* = 5, *initial_diameter* = 0.1, *type_measurement* = measurement_type.CENTROID_EUCLIDIAN_DISTANCE, *entry_size_limit* = 200, *ccore* = False)

Constructor of clustering algorithm BIRCH.

Parameters

in	*data*	(list): Input data presented as list of points (objects), where each point should be represented by list or tuple.
in	*number_clusters*	(uint): Number of clusters that should be allocated.
in	*branching_factor*	(uint): Maximum number of successor that might be contained by each non-leaf node in CF-Tree.
in	*max_node_↩ entries*	(uint): Maximum number of entries that might be contained by each leaf node in CF-Tree.
in	*initial_diameter*	(double): Initial diameter that used for CF-Tree construction, it can be increase if entry_size_limit is exceeded.
in	*type_↩ measurement*	(measurement_type): Type measurement used for calculation distance metrics.
in	*entry_size_limit*	(uint): Maximum number of entries that can be stored in CF-Tree, if it is exceeded during creation then diameter is increased and CF-Tree is rebuilt.
in	*ccore*	(bool): If True than DLL CCORE (C++ solution) will be used for solving the problem.

Remarks

Despite eight arguments only the first two is mandatory, others can be ommitted. In this case default values are used for instance creation.

Example:

```
birch_instance1 = birch(sample1, 2);    # two clusters should be allocated
birch_instance2 = birch(sample2, 5);    # five clusters should be allocated

# three clusters should be allocated, but also each leaf node can have maximum 5
# entries and each entry can have maximum 5 descriptors with initial diameter 0.05.
birch_instance3 = birch(sample3, 3, 5, 5, 0.05);
```

5.2.3 Member Function Documentation

5.2.3.1 def pyclustering.cluster.birch.birch.get_clusters (*self*)

Returns list of allocated clusters, each cluster contains indexes of objects in list of data.

Returns

(list) List of allocated clusters.

5.2.3.2 def pyclustering.cluster.birch.birch.process (*self*)

Performs cluster analysis in line with rules of BIRCH algorithm.

Remarks

Results of clustering can be obtained using corresponding gets methods.

See also

get_clusters()

The documentation for this class was generated from the following file:

- pyclustering/cluster/birch.py

5.3 pyclustering.cluster.canvas_cluster_descr Class Reference

Description of cluster for representation on canvas.

Public Member Functions

- def __init__ (self, cluster, data, marker, markersize)

  Constructor of cluster representation on the canvas.

Static Public Attributes

- cluster = None;

  Cluster that may consist of objects or indexes of objects from data.
- data = None;

  Data where objects are stored.
- marker = None;

  Marker that is used for drawing objects.
- markersize = None;

  Size of marker that is used for drawing objects.

5.3.1 Detailed Description

Description of cluster for representation on canvas.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 def pyclustering.cluster.canvas_cluster_descr.__init__ (*self, cluster, data, marker, markersize*)

Constructor of cluster representation on the canvas.

Parameters

in	*cluster*	(list): Single cluster that consists of objects or indexes from data.
in	*data*	(list): Objects that should be displayed, can be None if clusters consist of objects instead of indexes.
in	*marker*	(string): Type of marker that is used for drawing objects.
in	*markersize*	(uint): Size of marker that is used for drawing objects.

5.3.3 Member Data Documentation

5.3.3.1 pyclustering.cluster.canvas_cluster_descr.cluster = None; [static]

Cluster that may consist of objects or indexes of objects from data.

5.3.3.2 pyclustering.cluster.canvas_cluster_descr.data = None; [static]

Data where objects are stored.

It can be None if clusters consist of objects instead of indexes.

5.3.3.3 pyclustering.cluster.canvas_cluster_descr.marker = None; [static]

Marker that is used for drawing objects.

5.3.3.4 pyclustering.cluster.canvas_cluster_descr.markersize = None; [static]

Size of marker that is used for drawing objects.

The documentation for this class was generated from the following file:

- pyclustering/cluster/__init__.py

5.4 pyclustering.nnet.hhn.central_element Class Reference

Central element consist of two central neurons that are described by a little bit different dynamic than peripheral.

Public Member Functions

- def __init__ (self)
  Constructor of central element.
- def __repr__ (self)
  Returns string that represents central element.

Static Public Attributes

- float membrane_potential = 0.0
  Membrane potential of cenral neuron (V).
- float active_cond_sodium = 0.0
  Activation conductance of the sodium channel (m).
- float inactive_cond_sodium = 0.0
  Inactivaton conductance of the sodium channel (h).

- float active_cond_potassium = 0.0

    Inactivaton conductance of the sodium channel (h)
- pulse_generation_time = None;

    Times of pulse generation by central neuron.
- pulse_generation = False;

    Spike generation of central neuron.

5.4.1 Detailed Description

Central element consist of two central neurons that are described by a little bit different dynamic than peripheral.

See also

hhn_network

5.4.2 Member Data Documentation

5.4.2.1 float pyclustering.nnet.hhn.central_element.active_cond_sodium = 0.0 [static]

Activation conductance of the sodium channel (m).

5.4.2.2 float pyclustering.nnet.hhn.central_element.inactive_cond_sodium = 0.0 [static]

Inactivaton conductance of the sodium channel (h).

5.4.2.3 float pyclustering.nnet.hhn.central_element.membrane_potential = 0.0 [static]

Membrane potential of cenral neuron (V).

5.4.2.4 pyclustering.nnet.hhn.central_element.pulse_generation = False; [static]

Spike generation of central neuron.

5.4.2.5 pyclustering.nnet.hhn.central_element.pulse_generation_time = None; [static]

Times of pulse generation by central neuron.

The documentation for this class was generated from the following file:

- pyclustering/nnet/hhn.py

5.5 pyclustering.container.cftree.cfentry Class Reference

Clustering feature representation.

Public Member Functions

- def number_points (self)

    Returns number of points that are encoded.
- def linear_sum (self)

Returns linear sum.

- def square_sum (self)

    Returns square sum.

- def __init__ (self, number_points, linear_sum, square_sum)

    CF-entry constructor.

- def __copy__ (self)
- def __repr__ (self)
- def __str__ (self)

    Default cfentry string representation.

- def __add__ (self, entry)

    Overloaded operator add.

- def __sub__ (self, entry)

    Overloaded operator sub.

- def __eq__ (self, entry)

    Overloaded operator eq.

- def get_distance (self, entry, type_measurement)

    Calculates distance between two clusters in line with measurement type.

- def get_centroid (self)

    Calculates centroid of cluster that is represented by the entry.

- def get_radius (self)

    Calculates radius of cluster that is represented by the entry.

- def get_diameter (self)

    Calculates diameter of cluster that is represented by the entry.

Public Attributes

- **number_points**

5.5.1 Detailed Description

Clustering feature representation.

See also

cfnode
cftree

5.5.2 Constructor & Destructor Documentation

5.5.2.1 def pyclustering.container.cftree.cfentry.__init__ (*self, number_points, linear_sum, square_sum*)

CF-entry constructor.

Parameters

in	*number_points*	(uint): Number of objects that is represented by the entry.
in	*linear_sum*	(list): Linear sum of values that represent objects in each dimension.
in	*square_sum*	(double): Square sum of values that represent objects.

5.5.3 Member Function Documentation

5.5.3.1 def pyclustering.container.cftree.cfentry.__add__ (*self, entry*)

Overloaded operator add.

Performs addition of two clustering features.

Parameters

in	*entry*	(cfentry): Entry that is added to the current.

Returns

(cfentry) Result of addition of two clustering features.

5.5.3.2 def pyclustering.container.cftree.cfentry.__copy__ (*self*)

Returns

(cfentry) Makes copy of the cfentry instance.

5.5.3.3 def pyclustering.container.cftree.cfentry.__eq__ (*self*, *entry*)

Overloaded operator eq.

Performs comparison of two clustering features.

Parameters

in	*entry*	(cfentry): Entry that is used for comparison with current.

Returns

(bool) True is both clustering features are equals in line with tolerance, otherwise False.

5.5.3.4 def pyclustering.container.cftree.cfentry.__repr__ (*self*)

Returns

(string) Default cfentry representation.

5.5.3.5 def pyclustering.container.cftree.cfentry.__sub__ (*self*, *entry*)

Overloaded operator sub.

Performs substraction of two clustering features.

Substraction can't be performed with clustering feature whose description is less then substractor.

Parameters

in	*entry*	(cfentry): Entry that is substracted from the current.

Returns

(cfentry) Result of substraction of two clustering features.

5.5.3.6 def pyclustering.container.cftree.cfentry.get_centroid (*self*)

Calculates centroid of cluster that is represented by the entry.

It's calculated once when it's requested after the last changes.

Returns

(list) Centroid of cluster that is represented by the entry.

5.5.3.7 def pyclustering.container.cftree.cfentry.get_diameter (*self*)

Calculates diameter of cluster that is represented by the entry.

It's calculated once when it's requested after the last changes.

Returns

(double) Diameter of cluster that is represented by the entry.

5.5.3.8 def pyclustering.container.cftree.cfentry.get_distance (*self, entry, type_measurement*)

Calculates distance between two clusters in line with measurement type.

Parameters

in	*entry*	(cfentry): Clustering feature to which distance should be obtained.
in	*type_measurement*	(measurement_type): Distance measurement algorithm between two clusters.

Returns

(double) Distance between two clusters.

5.5.3.9 def pyclustering.container.cftree.cfentry.get_radius (*self*)

Calculates radius of cluster that is represented by the entry.

It's calculated once when it's requested after the last changes.

Returns

(double) Radius of cluster that is represented by the entry.

5.5.3.10 def pyclustering.container.cftree.cfentry.linear_sum (*self*)

Returns linear sum.

Returns

(list) Linear sum.

5.5.3.11 def pyclustering.container.cftree.cfentry.number_points (*self*)

Returns number of points that are encoded.

Returns

(uint) Number of encoded points.

5.5.3.12 def pyclustering.container.cftree.cfentry.square_sum (*self*)

Returns square sum.

Returns

(double) Square sum.

The documentation for this class was generated from the following file:

- pyclustering/container/cftree.py

5.6 pyclustering.container.cftree.cfnode Class Reference

Representation of node of CF-Tree.

Inheritance diagram for pyclustering.container.cftree.cfnode:

Public Member Functions

- def __init__ (self, feature, parent, payload)
  Constructor of abstract CF node.
- def __repr__ (self)
- def __str__ (self)
- def get_distance (self, node, type_measurement)
  Calculates distance between nodes in line with specified type measurement.

Static Public Attributes

- feature = None;
  Clustering feature of the node.
- parent = None;
  Pointer to the parent node (None for root).
- type = None;
  Type node (leaf or non-leaf).
- payload = None;
  Payload of node where user data can be stored.

5.6.1 Detailed Description

Representation of node of CF-Tree.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 def pyclustering.container.cftree.cfnode.__init__ (*self, feature, parent, payload*)

Constructor of abstract CF node.

Parameters

in	*feature*	(cfentry): Clustering feature of the created node.

in	*parent*	(cfnode): Parent of the created node.
in	*payload*	(∗): Data that is stored by the node.

5.6.3 Member Function Documentation

5.6.3.1 def pyclustering.container.cftree.cfnode.__repr__ (*self*)

Returns

(string) Default representation of CF node.

5.6.3.2 def pyclustering.container.cftree.cfnode.__str__ (*self*)

Returns

(string) String representation of CF node.

5.6.3.3 def pyclustering.container.cftree.cfnode.get_distance (*self, node, type_measurement*)

Calculates distance between nodes in line with specified type measurement.

Parameters

in	*node*	(cfnode): CF-node that is used for calculation distance to the current node.
in	*type_↩ measurement*	(measurement_type): Measurement type that is used for calculation distance.

Returns

(double) Distance between two nodes.

5.6.4 Member Data Documentation

5.6.4.1 pyclustering.container.cftree.cfnode.feature = None; [static]

Clustering feature of the node.

5.6.4.2 pyclustering.container.cftree.cfnode.parent = None; [static]

Pointer to the parent node (None for root).

5.6.4.3 pyclustering.container.cftree.cfnode.payload = None; [static]

Payload of node where user data can be stored.

5.6.4.4 pyclustering.container.cftree.cfnode.type = None; [static]

Type node (leaf or non-leaf).

The documentation for this class was generated from the following file:

- pyclustering/container/cftree.py

5.7 pyclustering.container.cftree.cfnode_type Class Reference

Enumeration of CF-Node types that are used by CF-Tree.

Inheritance diagram for pyclustering.container.cftree.cfnode_type:

Static Public Attributes

- int CFNODE_DUMMY = 0

  Undefined node.
- int CFNODE_LEAF = 1

  Leaf node hasn't got successors, only entries.
- int CFNODE_NONLEAF = 2

  Non-leaf node has got successors and hasn't got entries.

5.7.1 Detailed Description

Enumeration of CF-Node types that are used by CF-Tree.

See also

cfnode
cftree

5.7.2 Member Data Documentation

5.7.2.1 int pyclustering.container.cftree.cfnode_type.CFNODE_DUMMY = 0 [static]

Undefined node.

5.7.2.2 int pyclustering.container.cftree.cfnode_type.CFNODE_LEAF = 1 [static]

Leaf node hasn't got successors, only entries.

5.7.2.3 int pyclustering.container.cftree.cfnode_type.CFNODE_NONLEAF = 2 [static]

Non-leaf node has got successors and hasn't got entries.

The documentation for this class was generated from the following file:

- pyclustering/container/cftree.py

5.8 pyclustering.container.cftree.cftree Class Reference

CF-Tree representation.

Public Member Functions

- def root (self)
- def leafes (self)
- def amount_nodes (self)
- def amount_entries (self)
- def height (self)
- def branch_factor (self)
- def threshold (self)
- def max_entries (self)
- def type_measurement (self)
- def __init__

  Create CF-tree.
- def insert_cluster (self, cluster)

  Insert cluster that is represented as list of points where each point is represented by list of coordinates.
- def insert (self, entry)

  Insert clustering feature to the tree.
- def find_nearest_leaf

  Search nearest leaf to the specified clustering feature.

5.8.1 Detailed Description

CF-Tree representation.

A CF-tree is a height-balanced tree with two parameters: branching factor and threshold.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 def pyclustering.container.cftree.cftree.__init__ (*self, branch_factor, max_entries, threshold, type_measurement =* measurement_type.CENTROID_EUCLIDIAN_DISTANCE)

Create CF-tree.

Parameters

in	*branch_factor*	(uint): Maximum number of children for non-leaf nodes.
in	*max_entries*	(uint): Maximum number of entries for leaf nodes.
in	*threshold*	(double): Maximum diameter of feature clustering for each leaf node.
in	*type_↩ measurement*	(measurement_type): Measurement type that is used for calculation distance metrics.

5.8.3 Member Function Documentation

5.8.3.1 def pyclustering.container.cftree.cftree.amount_entries (*self*)

Returns

(uint) Number of entries in the tree.

5.8.3.2 def pyclustering.container.cftree.cftree.amount_nodes (*self*)

Returns

(unit) Number of nodes (leaf and non-leaf) in the tree.

5.8.3.3 def pyclustering.container.cftree.cftree.branch_factor (*self*)

Returns

(uint) Branching factor of the tree.

Branching factor defines maximum number of successors in each non-leaf node.

5.8.3.4 def pyclustering.container.cftree.cftree.find_nearest_leaf (*self*, *entry*, *search_node* = `None`)

Search nearest leaf to the specified clustering feature.

Parameters

in	*entry*	(cfentry): Clustering feature.
in	*search_node*	(cfnode): Node from that searching should be started, if None then search process will be started for the root.

Returns

(leaf_node) Nearest node to the specified clustering feature.

5.8.3.5 def pyclustering.container.cftree.cftree.height (*self*)

Returns

(uint) Height of the tree.

5.8.3.6 def pyclustering.container.cftree.cftree.insert (*self*, *entry*)

Insert clustering feature to the tree.

Parameters

in	*entry*	(cfentry): Clustering feature that should be inserted.

5.8.3.7 def pyclustering.container.cftree.cftree.insert_cluster (*self*, *cluster*)

Insert cluster that is represented as list of points where each point is represented by list of coordinates.

Clustering feature is created for that cluster and inserted to the tree.

Parameters

in	*cluster*	(list): Cluster that is represented by list of points that should be inserted to the tree.

5.8.3.8 def pyclustering.container.cftree.cftree.leafes (*self*)

Returns

(list) List of all non-leaf nodes in the tree.

5.8.3.9 def pyclustering.container.cftree.cftree.max_entries (*self*)

Returns

(uint) Maximum number of entries in each leaf node.

5.8.3.10 def pyclustering.container.cftree.cftree.root (*self*)

Returns

(cfnode) Root of the tree.

5.8.3.11 def pyclustering.container.cftree.cftree.threshold (*self*)

Returns

(double) Threshold of the tree that represents maximum diameter of sub-clusters that is formed by leaf node entries.

5.8.3.12 def pyclustering.container.cftree.cftree.type_measurement (*self*)

Returns

(measurement_type) Type that is used for measuring.

The documentation for this class was generated from the following file:

- pyclustering/container/cftree.py

5.9 pyclustering.cluster.cluster_visualizer Class Reference

Common visualizer of clusters on 2D or 3D surface.

Public Member Functions

- def __init__

  Constructor of cluster visualizer.
- def append_cluster

  Appends cluster to canvas for drawing.
- def append_clusters

  Appends list of cluster to canvas for drawing.
- def set_canvas_title (self, text, canvas)

  Set title for specified canvas.
- def show

  Shows clusters (visualize).

5.9.1 Detailed Description

Common visualizer of clusters on 2D or 3D surface.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 def pyclustering.cluster.cluster_visualizer.__init__ (*self*, *number_canvases* = `1`)

Constructor of cluster visualizer.

Parameters

`in`	*number_canvases*	(uint): Number of canvases that is used for visualization.

5.9.3 Member Function Documentation

5.9.3.1 def pyclustering.cluster.cluster_visualizer.append_cluster (*self, cluster, data =* `None`, *canvas =* `0`, *marker =* `'.'`, *markersize =* `5`)

Appends cluster to canvas for drawing.

Parameters

`in`	*cluster*	(list): cluster that may consist of indexes of objects from the data or object itself.
`in`	*data*	(list): If defines that each element of cluster is considered as a index of object from the data.
`in`	*canvas*	(uint): Number of canvas that should be used for displaying cluster.
`in`	*marker*	(string): Marker that is used for displaying objects from cluster on the canvas.
`in`	*markersize*	(uint): Size of marker.

5.9.3.2 def pyclustering.cluster.cluster_visualizer.append_clusters (*self, clusters, data =* `None`, *canvas =* `0`, *marker =* `'.'`, *markersize =* `5`)

Appends list of cluster to canvas for drawing.

Parameters

`in`	*clusters*	(list): List of clusters where each cluster may consist of indexes of objects from the data or object itself.
`in`	*data*	(list): If defines that each element of cluster is considered as a index of object from the data.
`in`	*canvas*	(uint): Number of canvas that should be used for displaying clusters.
`in`	*marker*	(string): Marker that is used for displaying objects from clusters on the canvas.
`in`	*markersize*	(uint): Size of marker.

5.9.3.3 def pyclustering.cluster.cluster_visualizer.set_canvas_title (*self, text, canvas*)

Set title for specified canvas.

Parameters

`in`	*text*	(string): Title for canvas.
`in`	*canvas*	(uint): Index of canvas where title should be displayed.

5.9.3.4 def pyclustering.cluster.cluster_visualizer.show (*self, visible_axis =* `True`, *visible_grid =* `True`)

Shows clusters (visualize).

Parameters

in	*visible_axis*	(bool): Defines visibility of axes on each canvas, if True - axes are invisible.
in	*visible_grid*	(bool): Defines visibility of axes on each canvas, if True - grid is displayed.

The documentation for this class was generated from the following file:

- pyclustering/cluster/__init__.py

5.10 pyclustering.nnet.conn_represent Class Reference

Enumerator of internal network connection representation between oscillators.

Inheritance diagram for pyclustering.nnet.conn_represent:

Static Public Attributes

- int LIST = 0

  Each oscillator has list of his neighbors.
- int MATRIX = 1

  Connections are represented my matrix connection NxN, where N is number of oscillators.

5.10.1 Detailed Description

Enumerator of internal network connection representation between oscillators.

5.10.2 Member Data Documentation

5.10.2.1 int pyclustering.nnet.conn_represent.LIST = 0 [static]

Each oscillator has list of his neighbors.

5.10.2.2 int pyclustering.nnet.conn_represent.MATRIX = 1 [static]

Connections are represented my matrix connection NxN, where N is number of oscillators.

The documentation for this class was generated from the following file:

- pyclustering/nnet/__init__.py

5.11 pyclustering.nnet.conn_type Class Reference

Enumerator of connection types between oscillators.

Inheritance diagram for pyclustering.nnet.conn_type:

Static Public Attributes

- int NONE = 0

  No connection between oscillators.
- int ALL_TO_ALL = 1

  All oscillators have connection with each other.
- int GRID_FOUR = 2

  Connections between oscillators represent grid where one oscillator can be connected with four neighbor oscillators: right, upper, left, lower.
- int GRID_EIGHT = 3

  Connections between oscillators represent grid where one oscillator can be connected with eight neighbor oscillators: right, right-upper, upper, upper-left, left, left-lower, lower, lower-right.
- int LIST_BIDIR = 4

  Connections between oscillators represent bidirectional list.
- int DYNAMIC = 5

  Connections are defined by user or by network during simulation.

5.11.1 Detailed Description

Enumerator of connection types between oscillators.

5.11.2 Member Data Documentation

5.11.2.1 int pyclustering.nnet.conn_type.ALL_TO_ALL = 1 `[static]`

All oscillators have connection with each other.

5.11.2.2 int pyclustering.nnet.conn_type.DYNAMIC = 5 `[static]`

Connections are defined by user or by network during simulation.

5.11.2.3 int pyclustering.nnet.conn_type.GRID_EIGHT = 3 `[static]`

Connections between oscillators represent grid where one oscillator can be connected with eight neighbor oscillators: right, right-upper, upper, upper-left, left, left-lower, lower, lower-right.

5.11.2.4 int pyclustering.nnet.conn_type.GRID_FOUR = 2 `[static]`

Connections between oscillators represent grid where one oscillator can be connected with four neighbor oscillators: right, upper, left, lower.

5.11.2.5 int pyclustering.nnet.conn_type.LIST_BIDIR = 4 `[static]`

Connections between oscillators represent bidirectional list.

5.11.2.6 int pyclustering.nnet.conn_type.NONE = 0 `[static]`

No connection between oscillators.

The documentation for this class was generated from the following file:

- pyclustering/nnet/__init__.py

5.12 pyclustering.cluster.cure.cure Class Reference

Class represents clustering algorithm CURE.

Public Member Functions

- def __init__

  Constructor of clustering algorithm CURE.
- def process (self)

  Performs cluster analysis in line with rules of CURE algorithm.
- def get_clusters (self)

  Returns list of allocated clusters, each cluster contains indexes of objects in list of data.

5.12.1 Detailed Description

Class represents clustering algorithm CURE.

Example:

```
# read data for clustering from some file
sample = read_sample(path_to_data);

# create instance of cure algorithm for cluster analysis
# request for allocation of two clusters.
cure_instance = cure(sample, 2, 5, 0.5, True);

# run cluster analysis
cure_instance.process();

# get results of clustering
clusters = cure_instance.get_clusters();
```

5.12.2 Constructor & Destructor Documentation

5.12.2.1 def pyclustering.cluster.cure.cure.__init__ (*self*, *data*, *number_cluster*, *number_represent_points* = `5`, *compression* = `0.5`, *ccore* = `False`)

Constructor of clustering algorithm CURE.

Parameters

in	*data*	(list): Input data that is presented as list of points (objects), each point should be represented by list or tuple.
in	*number_cluster*	(uint): Number of clusters that should be allocated.
in	*number_↩ represent_points*	(uint): Number of representative points for each cluster.
in	*compression*	(double): Coefficient defines level of shrinking of representation points toward the mean of the new created cluster after merging on each step. Usually it destributed from 0 to 1.
in	*ccore*	(bool): If True than DLL CCORE (C++ solution) will be used for solving.

5.12.3 Member Function Documentation

5.12.3.1 def pyclustering.cluster.cure.cure.get_clusters (*self*)

Returns list of allocated clusters, each cluster contains indexes of objects in list of data.

Returns

(list) List of allocated clusters.

See also

process()

5.12.3.2 def pyclustering.cluster.cure.cure.process (*self*)

Performs cluster analysis in line with rules of CURE algorithm.

Remarks

Results of clustering can be obtained using corresponding get methods.

See also

get_clusters()

The documentation for this class was generated from the following file:

- pyclustering/cluster/cure.py

5.13 pyclustering.cluster.cure.cure_cluster Class Reference

Represents data cluster in CURE term.

Public Member Functions

- def __init__
  Constructor of CURE cluster.
- def __repr__ (self)
  Displays distance to closest cluster and points that are contained by current cluster.

Static Public Attributes

- points = None;
  List of points that make up cluster.
- mean = None;
  Mean of points that make up cluster.
- rep = None;
  List of points that represents clusters.
- closest = None;
  Pointer to the closest cluster.
- distance = None;
  Distance to the closest cluster.

5.13.1 Detailed Description

Represents data cluster in CURE term.

CURE cluster is described by points of cluster, representation points of the cluster and by the cluster center.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 def pyclustering.cluster.cure.cure_cluster.__init__ (*self*, *point* = None)

Constructor of CURE cluster.

Parameters

in	*point*	(list): Point represented by list of coordinates.

5.13.3 Member Data Documentation

5.13.3.1 pyclustering.cluster.cure.cure_cluster.closest = None; [static]

Pointer to the closest cluster.

5.13.3.2 pyclustering.cluster.cure.cure_cluster.distance = None; [static]

Distance to the closest cluster.

5.13.3.3 pyclustering.cluster.cure.cure_cluster.mean = None; [static]

Mean of points that make up cluster.

5.13.3.4 pyclustering.cluster.cure.cure_cluster.points = None; [static]

List of points that make up cluster.

5.13.3.5 pyclustering.cluster.cure.cure_cluster.rep = None; [static]

List of points that represents clusters.

The documentation for this class was generated from the following file:

- pyclustering/cluster/cure.py

5.14 pyclustering.cluster.dbscan.dbscan Class Reference

Class represents clustering algorithm DBSCAN.

Public Member Functions

- def __init__ (self, data, eps, neighbors, ccore)

  Constructor of clustering algorithm DBSCAN.
- def process (self)

  Performs cluster analysis in line with rules of DBSCAN algorithm.
- def get_clusters (self)

  Returns allocated clusters.
- def get_noise (self)

  Returns allocated noise.

5.14.1 Detailed Description

Class represents clustering algorithm DBSCAN.

Example:

```
# sample for cluster analysis (represented by list)
sample = read_sample(path_to_sample);

# create object that uses CCORE for processing
dbscan_instance = dbscan(sample, 0.5, 3, True);

# cluster analysis
dbscan_instance.process();

# obtain results of clustering
clusters = dbscan_instance.get_clusters();
noise = dbscan_instance.get_noise();
```

5.14.2 Constructor & Destructor Documentation

5.14.2.1 def pyclustering.cluster.dbscan.dbscan.__init__ (*self*, *data*, *eps*, *neighbors*, *ccore*)

Constructor of clustering algorithm DBSCAN.

Parameters

in	*data*	(list): Input data that is presented as list of points (objects), each point should be represented by list or tuple.
in	*eps*	(double): Connectivity radius between points, points may be connected if distance between them less then the radius.
in	*neighbors*	(uint): minimum number of shared neighbors that is required for establish links between points.
in	*ccore*	(bool): if True than DLL CCORE (C++ solution) will be used for solving the problem.

5.14.3 Member Function Documentation

5.14.3.1 def pyclustering.cluster.dbscan.dbscan.get_clusters (*self*)

Returns allocated clusters.

Remarks

Allocated clusters can be returned only after data processing (use method process()). Otherwise empty list is returned.

Returns

(list) List of allocated clusters, each cluster contains indexes of objects in list of data.

See also

process()
get_noise()

5.14.3.2 def pyclustering.cluster.dbscan.dbscan.get_noise (*self*)

Returns allocated noise.

Remarks

Allocated noise can be returned only after data processing (use method process() before). Otherwise empty list is returned.

Returns

(list) List of indexes that are marked as a noise.

See also

process()
get_clusters()

5.14.3.3 def pyclustering.cluster.dbscan.dbscan.process (*self*)

Performs cluster analysis in line with rules of DBSCAN algorithm.

See also

get_clusters()
get_noise()

The documentation for this class was generated from the following file:

- pyclustering/cluster/dbscan.py

5.15 pyclustering.gcolor.dsatur.dsatur Class Reference

Represents DSATUR algorithm for graph coloring problem that uses greedy strategy.

Public Member Functions

- def __init__ (self, data)
  Constructor of DSATUR algorithm.
- def process (self)
  Perform graph coloring using DSATUR algorithm.
- def get_colors (self)
  Returns results of graph coloring.

5.15.1 Detailed Description

Represents DSATUR algorithm for graph coloring problem that uses greedy strategy.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 def pyclustering.gcolor.dsatur.dsatur.__init__ (*self, data*)

Constructor of DSATUR algorithm.

Parameters

in	*data*	(list): Matrix graph representation.

5.15.3 Member Function Documentation

5.15.3.1 def pyclustering.gcolor.dsatur.dsatur.get_colors (*self*)

Returns results of graph coloring.

Returns

(list) list with assigned colors where each element corresponds to node in the graph, for example [1, 2, 2, 1, 3, 4, 1].

See also

process()

5.15.3.2 def pyclustering.gcolor.dsatur.dsatur.process (*self*)

Perform graph coloring using DSATUR algorithm.

See also

get_colors()

The documentation for this class was generated from the following file:

- pyclustering/gcolor/dsatur.py

5.16 pyclustering.utils.graph.graph Class Reference

Graph representation.

Public Member Functions

- def __init__

  Constructor of graph.
- def __len__ (self)
- def data (self)
- def space_description (self)
- def comments (self)
- def type_graph_descr (self)

5.16.1 Detailed Description

Graph representation.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 def pyclustering.utils.graph.graph.__init__ (*self*, *data*, *type_graph* = `None`, *space_descr* = `None`, *comments* = `None`)

Constructor of graph.

Parameters

in	*data*	(list): Representation of graph. Considered as matrix if 'type_graph' is not specified.
in	*type_graph*	(type_graph_descr): Type of graph representation in 'data'.
in	*space_descr*	(list): Coordinates of each vertex that are used for graph drawing (can be omitted).
in	*comments*	(string): Comments related to graph.

5.16.3 Member Function Documentation

5.16.3.1 def pyclustering.utils.graph.graph.__len__ (*self*)

Returns

(uint) Size of graph defined by number of vertices.

5.16.3.2 def pyclustering.utils.graph.graph.comments (*self*)

Returns

(string) Comments.

5.16.3.3 def pyclustering.utils.graph.graph.data (*self*)

Returns

(list) Graph representation.

5.16.3.4 def pyclustering.utils.graph.graph.space_description (*self*)

Returns

(list) Space description.

5.16.3.5 def pyclustering.utils.graph.graph.type_graph_descr (*self*)

Returns

(type_graph_descr) Type of graph representation.

The documentation for this class was generated from the following file:

- pyclustering/utils/graph.py

5.17 pyclustering.nnet.hhn.hhn_network Class Reference

Oscillatory Neural Network with central element based on Hodgkin-Huxley neuron model.

Inheritance diagram for pyclustering.nnet.hhn.hhn_network:

Public Member Functions

- def __init__

    Constructor of oscillatory network based on Hodgkin-Huxley meuron model.
- def simulate

    Performs static simulation of oscillatory network based on Hodgkin-Huxley neuron model.
- def simulate_static

    Performs static simulation of oscillatory network based on Hodgkin-Huxley neuron model.
- def hnn_state (self, inputs, t, argv)

    Returns new values of excitatory and inhibitory parts of oscillator and potential of oscillator.
- def allocate_sync_ensembles

    Allocates clusters in line with ensembles of synchronous oscillators where each.

5.17.1 Detailed Description

Oscillatory Neural Network with central element based on Hodgkin-Huxley neuron model.

Interaction between oscillators is performed via central element (no connection between oscillators that are called as peripheral). Peripheral oscillators receive external stimulus. Central element consist of two oscillators: the first is used for synchronization some ensemble of oscillators and the second controls synchronization of the first cental oscillator with verious ensembles.

Example:

```
# change period of time when high strength value of synaptic connection exists from CN2 to PN.
params = hhn_parameters();
params.deltah = 400;

# create oscillatory network with stimulus
net = hhn_network(6, [0, 0, 25, 25, 47, 47], params);

# simulate network
(t, dyn) = net.simulate(1200, 600);

# draw network output during simulation
draw_dynamics(t, dyn, x_title = "Time", y_title = "V", separate = True);
```

5.17.2 Constructor & Destructor Documentation

5.17.2.1 def pyclustering.nnet.hhn.hhn_network.__init__ (*self*, *num_osc*, *stimulus* = None, *parameters* = None, *type_conn* = None, *type_conn_represent* = conn_represent.MATRIX)

Constructor of oscillatory network based on Hodgkin-Huxley meuron model.

Parameters

in	*num_osc*	(uint): Number of peripheral oscillators in the network.
in	*stimulus*	(list): List of stimulus for oscillators, number of stimulus should be equal to number of peripheral oscillators.
in	*parameters*	(hhn_parameters): Parameters of the network.
in	*type_conn*	(conn_type): Type of connections between oscillators in the network (ignored for this type of network).
in	*type_conn_↩represent*	(conn_represent): Internal representation of connection in the network: matrix or list.

5.17.3 Member Function Documentation

5.17.3.1 def pyclustering.nnet.hhn.hhn_network.allocate_sync_ensembles (*self*, *tolerance* = 0.1)

Allocates clusters in line with ensembles of synchronous oscillators where each.

Synchronous ensemble corresponds to only one cluster.

Parameters

in	*tolerance*	(double): maximum error for allocation of synchronous ensemble oscillators.

Returns

(list) Grours (lists) of indexes of synchronous oscillators. For example [[index_osc1, index_osc3], [index_osc2], [index_osc4, index_osc5]].

5.17.3.2 def pyclustering.nnet.hhn.hhn_network.hnn_state (*self, inputs, t, argv*)

Returns new values of excitatory and inhibitory parts of oscillator and potential of oscillator.

Parameters

in	*inputs*	(list): States of oscillator for integration [v, m, h, n] (see description below).
in	*t*	(double): Current time of simulation.
in	*argv*	(tuple): Extra arguments that are not used for integration - index of oscillator.

Returns

(list) new values of oscillator [v, m, h, n], where: v - membrane potential of oscillator, m - activation conductance of the sodium channel, h - inactication conductance of the sodium channel, n - activation conductance of the potassium channel.

5.17.3.3 def pyclustering.nnet.hhn.hhn_network.simulate (*self, steps, time, solution* = solve_type.RK4, *collect_dynamic* = `True`)

Performs static simulation of oscillatory network based on Hodgkin-Huxley neuron model.

Parameters

in	*steps*	(uint): Number steps of simulations during simulation.
in	*time*	(double): Time of simulation.
in	*solution*	(solve_type): Type of solver for differential equations.
in	*collect_dynamic*	(bool): If True - returns whole dynamic of oscillatory network, otherwise returns only last values of dynamics.

Returns

(list) Dynamic of oscillatory network. If argument 'collect_dynamic' = True, than return dynamic for the whole simulation time, otherwise returns only last values (last step of simulation) of dynamic.

5.17.3.4 def pyclustering.nnet.hhn.hhn_network.simulate_static (*self, steps, time, solution* = solve_type.RK4, *collect_dynamic* = `False`)

Performs static simulation of oscillatory network based on Hodgkin-Huxley neuron model.

Parameters

in	*steps*	(uint): Number steps of simulations during simulation.
in	*time*	(double): Time of simulation.

in	*solution*	(solve_type): Type of solver for differential equations.
in	*collect_dynamic*	(bool): If True - returns whole dynamic of oscillatory network, otherwise returns only last values of dynamics.

Returns

(list) Dynamic of oscillatory network. If argument 'collect_dynamic' = True, than return dynamic for the whole simulation time, otherwise returns only last values (last step of simulation) of dynamic.

The documentation for this class was generated from the following file:

- pyclustering/nnet/hhn.py

5.18 pyclustering.nnet.hhn.hhn_parameters Class Reference

Describes parameters of Hodgkin-Huxley Oscillatory Network.

Static Public Attributes

- tuple nu = random.random()

  Intrinsic noise.
- float gNa = 120.0

  Maximal conductivity for sodium current.
- float gK = 36.0

  Maximal conductivity for potassium current.
- float gL = 0.3

  Maximal conductivity for leakage current.
- float vNa = 50.0

  Reverse potential of sodium current [mV].
- float vK = -77.0

  Reverse potential of potassium current [mV].
- float vL = -54.4

  Reverse potantial of leakage current [mV].
- float vRest = -65.0

  Rest potential [mV].
- float Icn1 = 5.0

  External current [mV] for central element 1.
- float Icn2 = 30.0

  External current [mV] for central element 2.
- float Vsyninh = -80.0

  Synaptic reversal potential [mV] for inhibitory effects.
- float Vsynexc = 0.0

  Synaptic reversal potential [mV] for exciting effects.
- float alfa_inhibitory = 6.0

  Alfa-parameter for alfa-function for inhibitory effect.
- float betta_inhibitory = 0.3

  Betta-parameter for alfa-function for inhibitory effect.
- float alfa_excitatory = 40.0

  Alfa-parameter for alfa-function for excitatoty effect.
- float betta_excitatory = 2.0

Betta-parameter for alfa-function for excitatoty effect.
- float w1 = 0.1

  Strength of the synaptic connection from PN to CN1.
- float w2 = 9.0

  Strength of the synaptic connection from CN1 to PN.
- float w3 = 5.0

  Strength of the synaptic connection from CN2 to PN.
- float deltah = 650.0

  Period of time [ms] when high strength value of synaptic connection exists from CN2 to PN.
- int threshold = -10

  Threshold of the membrane potential that should exceeded by oscillator to be considered as an active.
- float eps = 0.16

  Affects pulse counter.

5.18.1 Detailed Description

Describes parameters of Hodgkin-Huxley Oscillatory Network.

See also

hhn_network

5.18.2 Member Data Documentation

5.18.2.1 float pyclustering.nnet.hhn.hhn_parameters.alfa_excitatory = 40.0 [static]

Alfa-parameter for alfa-function for excitatoty effect.

5.18.2.2 float pyclustering.nnet.hhn.hhn_parameters.alfa_inhibitory = 6.0 [static]

Alfa-parameter for alfa-function for inhibitory effect.

5.18.2.3 float pyclustering.nnet.hhn.hhn_parameters.betta_excitatory = 2.0 [static]

Betta-parameter for alfa-function for excitatoty effect.

5.18.2.4 float pyclustering.nnet.hhn.hhn_parameters.betta_inhibitory = 0.3 [static]

Betta-parameter for alfa-function for inhibitory effect.

5.18.2.5 float pyclustering.nnet.hhn.hhn_parameters.deltah = 650.0 [static]

Period of time [ms] when high strength value of synaptic connection exists from CN2 to PN.

5.18.2.6 float pyclustering.nnet.hhn.hhn_parameters.eps = 0.16 [static]

Affects pulse counter.

5.18.2.7 float pyclustering.nnet.hhn.hhn_parameters.gK = 36.0 [static]

Maximal conductivity for potassium current.

5.18.2.8 float pyclustering.nnet.hhn.hhn_parameters.gL = 0.3 [static]

Maximal conductivity for leakage current.

5.18.2.9 float pyclustering.nnet.hhn.hhn_parameters.gNa = 120.0 [static]

Maximal conductivity for sodium current.

5.18.2.10 float pyclustering.nnet.hhn.hhn_parameters.Icn1 = 5.0 [static]

External current [mV] for central element 1.

5.18.2.11 float pyclustering.nnet.hhn.hhn_parameters.Icn2 = 30.0 [static]

External current [mV] for central element 2.

5.18.2.12 tuple pyclustering.nnet.hhn.hhn_parameters.nu = random.random() [static]

Intrinsic noise.

5.18.2.13 int pyclustering.nnet.hhn.hhn_parameters.threshold = -10 [static]

Threshold of the membrane potential that should exceeded by oscillator to be considered as an active.

5.18.2.14 float pyclustering.nnet.hhn.hhn_parameters.vK = -77.0 [static]

Reverse potential of potassium current [mV].

5.18.2.15 float pyclustering.nnet.hhn.hhn_parameters.vL = -54.4 [static]

Reverse potantial of leakage current [mV].

5.18.2.16 float pyclustering.nnet.hhn.hhn_parameters.vNa = 50.0 [static]

Reverse potential of sodium current [mV].

5.18.2.17 float pyclustering.nnet.hhn.hhn_parameters.vRest = -65.0 [static]

Rest potential [mV].

5.18.2.18 float pyclustering.nnet.hhn.hhn_parameters.Vsynexc = 0.0 [static]

Synaptic reversal potential [mV] for exciting effects.

5.18.2.19 float pyclustering.nnet.hhn.hhn_parameters.Vsyninh = -80.0 [static]

Synaptic reversal potential [mV] for inhibitory effects.

5.18.2.20 float pyclustering.nnet.hhn.hhn_parameters.w1 = 0.1 [static]

Strength of the synaptic connection from PN to CN1.

5.18.2.21 float pyclustering.nnet.hhn.hhn_parameters.w2 = 9.0 [static]

Strength of the synaptic connection from CN1 to PN.

5.18.2.22 float pyclustering.nnet.hhn.hhn_parameters.w3 = 5.0 [static]

Strength of the synaptic connection from CN2 to PN.

The documentation for this class was generated from the following file:

- pyclustering/nnet/hhn.py

5.19 pyclustering.cluster.hsyncnet.hsyncnet Class Reference

Class represents clustering algorithm HSyncNet.

Inheritance diagram for pyclustering.cluster.hsyncnet.hsyncnet:

Public Member Functions

- def __init__

  Costructor of the oscillatory network hSyncNet for cluster analysis.
- def __del__ (self)

  Destructor of oscillatory network HSyncNet.
- def process

  Performs clustering of input data set in line with input parameters.

5.19.1 Detailed Description

Class represents clustering algorithm HSyncNet.

HSyncNet is bio-inspired algorithm that is based on oscillatory network that uses modified Kuramoto model.

Example:

```
# read list of points for cluster analysis
sample = read_sample(file);

# create network for allocation three clusters using CCORE (C++ implementation)
network = hsyncnet(sample, 3, ccore = True);

# run cluster analysis and output dynamic of the network
(time, dynamic) = network.process(0.995, collect_dynamic = True);

# get allocated clusters
clusters = network.get_clusters();

# show output dynamic of the network
draw_dynamics(time, dynamic);
```

5.19.2 Constructor & Destructor Documentation

5.19.2.1 def pyclustering.cluster.hsyncnet.hsyncnet.__init__ (*self*, *source_data*, *number_clusters*, *osc_initial_phases* = initial_type.RANDOM_GAUSSIAN, *ccore* = `False`)

Costructor of the oscillatory network hSyncNet for cluster analysis.

Parameters

in	*source_data*	(list): Input data set defines structure of the network.
in	*number_clusters*	(uint): Number of clusters that should be allocated.
in	*osc_initial_↩ phases*	(initial_type): Type of initialization of initial values of phases of oscillators.
in	*ccore*	(bool): If True than DLL CCORE (C++ solution) will be used for solving.

5.19.3 Member Function Documentation

5.19.3.1 def pyclustering.cluster.hsyncnet.hsyncnet.process (*self*, *order* = `0.998`, *solution* = **solve_type.FAST**, *collect_dynamic* = `False`)

Performs clustering of input data set in line with input parameters.

Parameters

in	*order*	(double): Level of local synchronization between oscillator that defines end of synchronization process, range [0..1].
in	*solution*	(solve_type) Type of solving differential equation.
in	*collect_dynamic*	(bool): If True - returns whole history of process synchronization otherwise - only final state (when process of clustering is over).

Returns

(tuple) Returns dynamic of the network as tuple of lists on each iteration (time, oscillator_phases) that depends on collect_dynamic parameter.

See also

get_clusters()

The documentation for this class was generated from the following file:

- pyclustering/cluster/hsyncnet.py

5.20 pyclustering.nnet.hysteresis.hysteresis_network Class Reference

Hysteresis oscillatory network that uses relaxation oscillators.

Inheritance diagram for pyclustering.nnet.hysteresis.hysteresis_network:

Public Member Functions

- def outputs (self)

  Returns current outputs of neurons.
- def outputs (self, values)

  Sets outputs of neurons.
- def states (self)

  Return current states of neurons.
- def states (self, values)

  Set current states of neurons.

- def __init__

  Constructor of hysteresis oscillatory network.
- def simulate

  Performs static simulation of hysteresis oscillatory network.
- def simulate_static

  Performs static simulation of hysteresis oscillatory network.
- def allocate_sync_ensembles

  Allocate clusters in line with ensembles of synchronous oscillators where each synchronous ensemble corresponds to only one cluster.

5.20.1 Detailed Description

Hysteresis oscillatory network that uses relaxation oscillators.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 def pyclustering.nnet.hysteresis.hysteresis_network.__init__ (*self, num_osc, own_weight =* -4, *neigh_weight =* -1, *type_conn =* conn_type.ALL_TO_ALL, *type_conn_represent =* conn_represent.MATRIX)

Constructor of hysteresis oscillatory network.

Parameters

in	*num_osc*	(uint): Number of oscillators in the network.
in	*own_weight*	(double): Weight of connection from oscillator to itself - own weight.
in	*neigh_weight*	(double): Weight of connection between oscillators.
in	*type_conn*	(conn_type): Type of connection between oscillators in the network.
in	*type_conn_↩ represent*	(conn_represent): Internal representation of connection in the network: matrix or list.

5.20.3 Member Function Documentation

5.20.3.1 def pyclustering.nnet.hysteresis.hysteresis_network.allocate_sync_ensembles (*self, tolerance =* 0.1)

Allocate clusters in line with ensembles of synchronous oscillators where each synchronous ensemble corresponds to only one cluster.

Parameters

in	*tolerance*	(double): Maximum error for allocation of synchronous ensemble oscillators.

Returns

(list) Grours of indexes of synchronous oscillators, for example, [[index_osc1, index_osc3], [index_osc2], [index_osc4, index_osc5]]."

5.20.3.2 def pyclustering.nnet.hysteresis.hysteresis_network.outputs (*self*)

Returns current outputs of neurons.

Returns

(list) Current outputs of neurons.

5.20.3.3 def pyclustering.nnet.hysteresis.hysteresis_network.simulate (*self*, *steps*, *time*, *solution* = solve_type.RK4, *collect_dynamic* = True)

Performs static simulation of hysteresis oscillatory network.

Parameters

in	*steps*	(uint): Number steps of simulations during simulation.
in	*time*	(double): Time of simulation.
in	*solution*	(solve_type): Type of solution (solving).
in	*collect_dynamic*	(bool): If True - returns whole dynamic of oscillatory network, otherwise returns only last values of dynamics.

Returns

(list) Dynamic of oscillatory network. If argument 'collect_dynamic' = True, than return dynamic for the whole simulation time, otherwise returns only last values (last step of simulation) of dynamic.

5.20.3.4 def pyclustering.nnet.hysteresis.hysteresis_network.simulate_static (*self*, *steps*, *time*, *solution* = solve_type.RK4, *collect_dynamic* = `False`)

Performs static simulation of hysteresis oscillatory network.

Parameters

in	*steps*	(uint): Number steps of simulations during simulation.
in	*time*	(double): Time of simulation.
in	*solution*	(solve_type): Type of solution (solving).
in	*collect_dynamic*	(bool): If True - returns whole dynamic of oscillatory network, otherwise returns only last values of dynamics.

Returns

(list) Dynamic of oscillatory network. If argument 'collect_dynamic' = True, than return dynamic for the whole simulation time, otherwise returns only last values (last step of simulation) of dynamic.

5.20.3.5 def pyclustering.nnet.hysteresis.hysteresis_network.states (*self*)

Return current states of neurons.

Returns

(list) States of neurons.

The documentation for this class was generated from the following file:

- pyclustering/nnet/hysteresis.py

5.21 pyclustering.gcolor.hysteresis.hysteresisgcolor Class Reference

Class represents graph coloring algorithm based on hysteresis oscillatory network.

Inheritance diagram for pyclustering.gcolor.hysteresis.hysteresisgcolor:

Public Member Functions

- def __init__ (self, graph_matrix, alpha, eps)

Constructor of hysteresis oscillatory network for graph coloring.

- def get_clusters

  Returns list of clusters where each cluster represents ensemble of synchronous oscillators and each each cluster denotes set of oscillators that correspond to only one color.

- def get_map_coloring

  Returns list of color indexes that are assigned to each object from input data space accordingly.

5.21.1 Detailed Description

Class represents graph coloring algorithm based on hysteresis oscillatory network.

This is bio-inspired algorithm where the network uses relaxation oscillators that is regarded as a multi-vibrator. Each ensemble of synchronous oscillators corresponds to only one color.

Example

```
# load graph from a file
graph = read_graph(filename);

# create oscillatory network for solving graph coloring problem
network = hysteresisgcolor(graph.data, alpha, eps);

# perform simulation of the network
(t, dyn) = network.simulate(2000, 20);

# show dynamic of the network
draw_dynamics(t, dyn, x_title = "Time", y_title = "State");

# obtain results of graph coloring and display results
coloring_map = network.get_map_coloring();
draw_graph(graph, coloring_map);
```

5.21.2 Constructor & Destructor Documentation

5.21.2.1 def pyclustering.gcolor.hysteresis.hysteresisgcolor.__init__ (*self, graph_matrix, alpha, eps*)

Constructor of hysteresis oscillatory network for graph coloring.

Parameters

in	*graph_matrix*	(list): Matrix representation of a graph.
in	*alpha*	(double): Positive constant (affect weight between two oscillators w[i][j]).
in	*eps*	(double): Positive constant (affect feedback to itself (i = j) of each oscillator w[i][j] = -alpha - eps).

5.21.3 Member Function Documentation

5.21.3.1 def pyclustering.gcolor.hysteresis.hysteresisgcolor.get_clusters (*self, tolerance* = `0.1`)

Returns list of clusters where each cluster represents ensemble of synchronous oscillators and each each cluster denotes set of oscillators that correspond to only one color.

Parameters

in	*tolerance*	(double): Tolerance level that define maximal difference between outputs of oscillators in one synchronous ensemble.

Remarks

Results can be obtained only after network simulation (graph processing by the network).

Returns

(list) Lists of ensembles of synchronous oscillators that consist of indexes of oscillators, for example [[0, 2, 5], [1, 3, 4]].

See also

simulate()
get_map_coloring()

5.21.3.2 def pyclustering.gcolor.hysteresis.hysteresisgcolor.get_map_coloring (*self*, *tolerance* = `0.1`)

Returns list of color indexes that are assigned to each object from input data space accordingly.

Parameters

`in`	*tolerance*	(double): Tolerance level that define maximal difference between outputs of oscillators in one synchronous ensemble.

Remarks

Results can be obtained only after network simulation (graph processing by the network).

Returns

(list) Color indexes that are assigned to each object from input data space accordingly.

See also

simulate()
get_clusters()

The documentation for this class was generated from the following file:

- pyclustering/gcolor/hysteresis.py

5.22 pyclustering.nnet.initial_type Class Reference

Enumerator of types of oscillator output initialization.

Inheritance diagram for pyclustering.nnet.initial_type:

Static Public Attributes

- int RANDOM_GAUSSIAN = 0
  Output of oscillators are random in line with gaussian distribution.
- int EQUIPARTITION = 1
  Output of oscillators are equidistant from each other (uniformly distributed, not randomly).

5.22.1 Detailed Description

Enumerator of types of oscillator output initialization.

5.22.2 Member Data Documentation

5.22.2.1 int pyclustering.nnet.initial_type.EQUIPARTITION = 1 [static]

Output of oscillators are equidistant from each other (uniformly distributed, not randomly).

5.22.2.2 int pyclustering.nnet.initial_type.RANDOM_GAUSSIAN = 0 [static]

Output of oscillators are random in line with gaussian distribution.

The documentation for this class was generated from the following file:

- pyclustering/nnet/__init__.py

5.23 pyclustering.container.kdtree.kdtree Class Reference

Represents KD Tree.

Public Member Functions

- def __init__
  Create kd-tree from list of points and from according list of payloads.
- def insert (self, point, payload)
  Insert new point with payload to kd-tree.
- def remove (self, point)
  Remove specified point from kd-tree.
- def find_minimal_node (self, node, discriminator)
  Find minimal node in line with coordinate that is defined by discriminator.
- def find_node
  Find node with coordinates that are defined by specified point.
- def find_nearest_dist_node
  Find nearest neighbor in area with radius = distance.
- def find_nearest_dist_nodes (self, point, distance)
  Find neighbors that are located in area that is covered by specified distance.
- def children (self, node)
  Returns list of children of node.
- def traverse
  Traverses all nodes of subtree that is defined by node specified in input parameter.
- def show (self)
  Display tree on the console.

5.23.1 Detailed Description

Represents KD Tree.

5.23.2 Constructor & Destructor Documentation

5.23.2.1 def pyclustering.container.kdtree.kdtree.__init__ (*self*, *data_list* = None, *payload_list* = None)

Create kd-tree from list of points and from according list of payloads.

If lists were not specified then empty kd-tree will be created.

Parameters

in	*data_list*	(list): Insert points from the list to created KD tree.
in	*payload_list*	(list): Insert payload from the list to created KD tree, length should be equal to length of data_list if it is specified.

5.23.3 Member Function Documentation

5.23.3.1 def pyclustering.container.kdtree.kdtree.children (*self*, *node*)

Returns list of children of node.

Parameters

in	*node*	(node): Node whose children are required.

Returns

(list) Children of node. If node haven't got any child then None is returned.

5.23.3.2 def pyclustering.container.kdtree.kdtree.find_minimal_node (*self*, *node*, *discriminator*)

Find minimal node in line with coordinate that is defined by discriminator.

Parameters

in	*node*	(node): Node of KD tree from that search should be started.
in	*discriminator*	(uint): Coordinate number that is used for comparison.

Returns

(node) Minimal node in line with descriminator from the specified node.

5.23.3.3 def pyclustering.container.kdtree.kdtree.find_nearest_dist_node (*self*, *point*, *distance*, *retdistance* = `False`)

Find nearest neighbor in area with radius = distance.

Parameters

in	*point*	(list): Maximum distance where neighbors are searched.
in	*distance*	(double): Maximum distance where neighbors are searched.
in	*retdistance*	(bool): If True - returns neighbors with distances to them, otherwise only neighbors is returned.

Returns

(list) Neighbors, if redistance is True then neighbors with distances to them will be returned.

5.23.3.4 def pyclustering.container.kdtree.kdtree.find_nearest_dist_nodes (*self*, *point*, *distance*)

Find neighbors that are located in area that is covered by specified distance.

Parameters

in	*point*	(list): Coordinates that is considered as centroind for searching.
in	*distance*	(double): Distance from the center where seaching is performed.

Returns

(list) Neighbors in area that is specified by point (center) and distance (radius).

5.23.3.5 def pyclustering.container.kdtree.kdtree.find_node (*self*, *point*, *cur_node* = None)

Find node with coordinates that are defined by specified point.

If node does not exist then None will be returned. Otherwise required node will be returned.

Parameters

in	*point*	(list): Coordinates of the point whose node should be found.
in	*cur_node*	(node): Node from which search should be started.

Returns

(node) Node in case of existance of node with specified coordinates, otherwise it return None.

5.23.3.6 def pyclustering.container.kdtree.kdtree.insert (*self*, *point*, *payload*)

Insert new point with payload to kd-tree.

Parameters

in	*point*	(list): Coordinates of the point of inserted node.
in	*payload*	(*): Payload of inserted node.

5.23.3.7 def pyclustering.container.kdtree.kdtree.remove (*self*, *point*)

Remove specified point from kd-tree.

Parameters

in	*point*	(list): Coordinates of the point of removed node.

Returns

(node) Root if node has been successfully removed, otherwise None.

5.23.3.8 def pyclustering.container.kdtree.kdtree.traverse (*self*, *start_node* = None, *level* = None)

Traverses all nodes of subtree that is defined by node specified in input parameter.

Parameters

in	*start_node*	(node): Node from that travering of subtree is performed.

in,out	*level*	(uint): Should be ignored by application.

Returns

(list) All nodes of the subtree.

The documentation for this class was generated from the following file:

- pyclustering/container/kdtree.py

5.24 pyclustering.cluster.kmeans.kmeans Class Reference

Class represents clustering algorithm K-Means.

Public Member Functions

- def __init__

  Constructor of clustering algorithm K-Means.
- def process (self)

  Performs cluster analysis in line with rules of K-Means algorithm.
- def get_clusters (self)

  Returns list of allocated clusters, each cluster contains indexes of objects in list of data.
- def get_centers (self)

  Returns list of centers of allocated clusters.

5.24.1 Detailed Description

Class represents clustering algorithm K-Means.

Example:

```
# load list of points for cluster analysis
sample = read_sample(path);

# create instance of K-Means algorithm
kmeans_instance = kmeans(sample, [ [0.0, 0.1], [2.5, 2.6] ]);

# run cluster analysis and obtain results
kmeans_instance.process();
kmeans_instance.get_clusters();
```

5.24.2 Constructor & Destructor Documentation

5.24.2.1 def pyclustering.cluster.kmeans.kmeans.__init__ (*self, data, initial_centers, tolerance =* 0.25, *ccore =* False)

Constructor of clustering algorithm K-Means.

Parameters

in	*data*	(list): Input data that is presented as list of points (objects), each point should be represented by list or tuple.

in	*initial_centers*	(list): Initial coordinates of centers of clusters that are represented by list↩ : [center1, center2, ...].
in	*tolerance*	(double): Stop condition: if maximum value of change of centers of clusters is less than tolerance than algorithm will stop processing
in	*ccore*	(bool): Defines should be CCORE library (C++ pyclustering library) used instead of Python code or not.

5.24.3 Member Function Documentation

5.24.3.1 def pyclustering.cluster.kmeans.kmeans.get_centers (*self*)

Returns list of centers of allocated clusters.

See also

process()
get_clusters()

5.24.3.2 def pyclustering.cluster.kmeans.kmeans.get_clusters (*self*)

Returns list of allocated clusters, each cluster contains indexes of objects in list of data.

See also

process()
get_centers()

5.24.3.3 def pyclustering.cluster.kmeans.kmeans.process (*self*)

Performs cluster analysis in line with rules of K-Means algorithm.

Remarks

Results of clustering can be obtained using corresponding get methods.

See also

get_clusters()
get_centers()

The documentation for this class was generated from the following file:

- pyclustering/cluster/kmeans.py

5.25 pyclustering.cluster.kmedians.kmedians Class Reference

Class represents clustering algorithm K-Medians.

Public Member Functions

- def __init__

  Constructor of clustering algorithm K-Medians.

- def process (self)

  Performs cluster analysis in line with rules of K-Medians algorithm.
- def get_clusters (self)

  Returns list of allocated clusters, each cluster contains indexes of objects in list of data.
- def get_medians (self)

  Returns list of centers of allocated clusters.

5.25.1 Detailed Description

Class represents clustering algorithm K-Medians.

The algorithm is less sensitive to outliers tham K-Means.

Example:

```
# load list of points for cluster analysis
sample = read_sample(path);

# create instance of K-Medians algorithm
kmedians_instance = kmedians(sample, [ [0.0, 0.1], [2.5, 2.6] ]);

# run cluster analysis and obtain results
kmedians_instance.process();
kmedians_instance.get_clusters();
```

5.25.2 Constructor & Destructor Documentation

5.25.2.1 def pyclustering.cluster.kmedians.kmedians.__init__ (*self, data, initial_centers, tolerance =* `0.25`)

Constructor of clustering algorithm K-Medians.

Parameters

in	*data*	(list): Input data that is presented as list of points (objects), each point should be represented by list or tuple.
in	*initial_centers*	(list): Initial coordinates of centers of clusters that are represented by list↩ : [center1, center2, ...].
in	*tolerance*	(double): Stop condition: if maximum value of change of centers of clusters is less than tolerance than algorithm will stop processing

5.25.3 Member Function Documentation

5.25.3.1 def pyclustering.cluster.kmedians.kmedians.get_clusters (*self*)

Returns list of allocated clusters, each cluster contains indexes of objects in list of data.

See also

process()
get_medians()

5.25.3.2 def pyclustering.cluster.kmedians.kmedians.get_medians (*self*)

Returns list of centers of allocated clusters.

See also

process()
get_clusters()

5.25.3.3 def pyclustering.cluster.kmedians.kmedians.process (*self*)

Performs cluster analysis in line with rules of K-Medians algorithm.

Remarks

Results of clustering can be obtained using corresponding get methods.

See also

get_clusters()
get_medians()

The documentation for this class was generated from the following file:

- pyclustering/cluster/kmedians.py

5.26 pyclustering.cluster.kmedoids.kmedoids Class Reference

Class represents clustering algorithm K-Medoids (another one title is PAM - Parti).

Public Member Functions

- def __init__

  Constructor of clustering algorithm K-Medoids.
- def process (self)

  Performs cluster analysis in line with rules of K-Medoids algorithm.
- def get_clusters (self)

  Returns list of allocated clusters, each cluster contains indexes of objects in list of data.
- def get_medoids (self)

  Returns list of centers of allocated clusters.

5.26.1 Detailed Description

Class represents clustering algorithm K-Medoids (another one title is PAM - Parti).

The algorithm is less sensitive to outliers tham K-Means. The principle difference between K-Medoids and K-Medians is that K-Medoids uses existed points from input data space as medoids, but median in K-Medians can be unreal object (not from input data space).

Example:

```
# load list of points for cluster analysis
sample = read_sample(path);

# create instance of K-Medoids algorithm
kmedians_instance = kmedians(sample, [1, 10]);

# run cluster analysis and obtain results
kmedians_instance.process();
kmedians_instance.get_clusters();
```

5.26.2 Constructor & Destructor Documentation

5.26.2.1 def pyclustering.cluster.kmedoids.kmedoids.__init__ (*self, data, initial_index_medoids, tolerance =* `0.25`)

Constructor of clustering algorithm K-Medoids.

Parameters

in	*data*	(list): Input data that is presented as list of points (objects), each point should be represented by list or tuple.
in	*initial_index_↩ medoids*	(list): Indexes of intial medoids (indexes of points in input data).
in	*tolerance*	(double): Stop condition: if maximum value of change of centers of clusters is less than tolerance than algorithm will stop processing

5.26.3 Member Function Documentation

5.26.3.1 def pyclustering.cluster.kmedoids.kmedoids.get_clusters (*self*)

Returns list of allocated clusters, each cluster contains indexes of objects in list of data.

See also

process()
get_medoids()

5.26.3.2 def pyclustering.cluster.kmedoids.kmedoids.get_medoids (*self*)

Returns list of centers of allocated clusters.

See also

process()
get_clusters()

5.26.3.3 def pyclustering.cluster.kmedoids.kmedoids.process (*self*)

Performs cluster analysis in line with rules of K-Medoids algorithm.

Remarks

Results of clustering can be obtained using corresponding get methods.

See also

get_clusters()
get_medoids()

The documentation for this class was generated from the following file:

- pyclustering/cluster/kmedoids.py

5.27 pyclustering.container.cftree.leaf_node Class Reference

Represents clustering feature leaf node.

Inheritance diagram for pyclustering.container.cftree.leaf_node:

Public Member Functions

- def entries (self)
- def __init__ (self, feature, parent, entries, payload)

  Create CF Leaf node.
- def __repr__ (self)
- def __str__ (self)
- def insert_entry (self, entry)

  Insert new clustering feature to the leaf node.
- def remove_entry (self, entry)

  Remove clustering feature from the leaf node.
- def merge (self, node)

  Merge leaf node to the current.
- def get_farthest_entries (self, type_measurement)

  Find pair of farthest entries of the node.
- def get_nearest_index_entry (self, entry, type_measurement)

  Find nearest index of nearest entry of node for the specified entry.
- def get_nearest_entry (self, entry, type_measurement)

  Find nearest entry of node for the specified entry.

Public Attributes

- **type**

Additional Inherited Members

5.27.1 Detailed Description

Represents clustering feature leaf node.

5.27.2 Constructor & Destructor Documentation

5.27.2.1 def pyclustering.container.cftree.leaf_node.__init__ (*self, feature, parent, entries, payload*)

Create CF Leaf node.

Parameters

in	*feature*	(cfentry): Clustering feature of the created node.
in	*parent*	(non_leaf_node): Parent of the created node.
in	*entries*	(list): List of entries of the node.
in	*payload*	(∗): Data that is stored by the node.

5.27.3 Member Function Documentation

5.27.3.1 def pyclustering.container.cftree.leaf_node.__repr__ (*self*)

Returns

(string) Default leaf node represenation.

5.27.3.2 def pyclustering.container.cftree.leaf_node.__str__ (*self*)

Returns

(string) String leaf node representation.

5.27.3.3 def pyclustering.container.cftree.leaf_node.entries (*self*)

Returns

(list) List of leaf nodes.

5.27.3.4 def pyclustering.container.cftree.leaf_node.get_farthest_entries (*self, type_measurement*)

Find pair of farthest entries of the node.

Parameters

in	*type_measurement*	(measurement_type): Measurement type that is used for obtaining farthest entries.

Returns

(list) Pair of farthest entries of the node that are represented by list.

5.27.3.5 def pyclustering.container.cftree.leaf_node.get_nearest_entry (*self, entry, type_measurement*)

Find nearest entry of node for the specified entry.

Parameters

in	*entry*	(cfentry): Entry that is used for calculation distance.
in	*type_measurement*	(measurement_type): Measurement type that is used for obtaining nearest entry to the specified.

Returns

(cfentry) Nearest entry of node for the specified entry.

5.27.3.6 def pyclustering.container.cftree.leaf_node.get_nearest_index_entry (*self, entry, type_measurement*)

Find nearest index of nearest entry of node for the specified entry.

Parameters

in	*entry*	(cfentry): Entry that is used for calculation distance.
in	*type_measurement*	(measurement_type): Measurement type that is used for obtaining nearest entry to the specified.

Returns

(uint) Index of nearest entry of node for the specified entry.

5.27.3.7 def pyclustering.container.cftree.leaf_node.insert_entry (*self, entry*)

Insert new clustering feature to the leaf node.

Parameters

in	*entry*	(cfentry): Clustering feature.

5.27.3.8 def pyclustering.container.cftree.leaf_node.merge (*self, node*)

Merge leaf node to the current.

Parameters

in	*node*	(leaf_node): Leaf node that should be merged with current.

5.27.3.9 def pyclustering.container.cftree.leaf_node.remove_entry (*self, entry*)

Remove clustering feature from the leaf node.

Parameters

in	*entry*	(cfentry): Clustering feature.

The documentation for this class was generated from the following file:

- pyclustering/container/cftree.py

5.28 pyclustering.nnet.legion.legion_dynamic Class Reference

Represents output dynamic of LEGION.

Public Member Functions

- def output (self)

  Returns output dynamic of the network.
- def inhibitor (self)

  Returns output dynamic of the global inhibitor of the network.
- def time (self)

  Returns simulation time.
- def __init__

  Constructor of legion dynamic.
- def __del__ (self)

  Destructor of the dynamic of the legion network.
- def __len__ (self)

  Returns length of output dynamic.
- def allocate_sync_ensembles

  Allocate clusters in line with ensembles of synchronous oscillators where each synchronous ensemble corresponds to only one cluster.

5.28.1 Detailed Description

Represents output dynamic of LEGION.

5.28.2 Constructor & Destructor Documentation

5.28.2.1 def pyclustering.nnet.legion.legion_dynamic.__init__ (*self*, *output*, *inhibitor*, *time*, *ccore* = `None`)

Constructor of legion dynamic.

Parameters

in	*output*	(list): Output dynamic of the network represented by excitatory values of oscillators.
in	*inhibitor*	(list): Output dynamic of the global inhibitor of the network.
in	*time*	(list): Simulation time.
in	*ccore*	(POINTER): Pointer to CCORE legion_dynamic. If it is specified then others arguments can be omitted.

5.28.3 Member Function Documentation

5.28.3.1 def pyclustering.nnet.legion.legion_dynamic.allocate_sync_ensembles (*self, tolerance* = `0.1`)

Allocate clusters in line with ensembles of synchronous oscillators where each synchronous ensemble corresponds to only one cluster.

Parameters

in	*tolerance*	(double): Maximum error for allocation of synchronous ensemble oscillators.

Returns

(list) Grours of indexes of synchronous oscillators, for example, [[index_osc1, index_osc3], [index_osc2], [index_osc4, index_osc5]].

The documentation for this class was generated from the following file:

- pyclustering/nnet/legion.py

5.29 pyclustering.nnet.legion.legion_network Class Reference

Local excitatory global inhibitory oscillatory network (LEGION) that uses relaxation oscillator based on Van der Pol model.

Inheritance diagram for pyclustering.nnet.legion.legion_network:

Public Member Functions

- def __init__

  Constructor of oscillatory network LEGION (local excitatory global inhibitory oscillatory network).
- def simulate

  Performs static simulation of LEGION oscillatory network.

5.29.1 Detailed Description

Local excitatory global inhibitory oscillatory network (LEGION) that uses relaxation oscillator based on Van der Pol model.

The model uses global inhibitor to de-synchronize synchronous ensembles of oscillators.

Example:

```
# Create parameters of the network
parameters = legion_parameters();
parameters.Wt = 4.0;

# Create stimulus
stimulus = [1, 1, 0, 0, 0, 1, 1, 1];

# Create the network (use CCORE for fast solving)
net = legion_network(len(stimulus), parameters, conn_type.GRID_FOUR, ccore = True);

# Simulate network - result of simulation is output dynamic of the network
output_dynamic = net.simulate(1000, 750, stimulus);

# Draw output dynamic
draw_dynamics(output_dynamic.time, output_dynamic.output, x_title = "Time", y_title = "x(t)");
```

5.29.2 Constructor & Destructor Documentation

5.29.2.1 def pyclustering.nnet.legion.legion_network.__init__ (*self*, *num_osc*, *parameters* = None, *type_conn* = conn_type.ALL_TO_ALL, *type_conn_represent* = conn_represent.MATRIX, *ccore* = False)

Constructor of oscillatory network LEGION (local excitatory global inhibitory oscillatory network).

Parameters

in	*num_osc*	(uint): Number of oscillators in the network.
in	*parameters*	(legion_parameters): Parameters of the network that are defined by structure 'legion_parameters'.
in	*type_conn*	(conn_type): Type of connection between oscillators in the network.
in	*type_conn_↩ represent*	(conn_represent): Internal representation of connection in the network: matrix or list.
in	*ccore*	(bool): If True then all interaction with object will be performed via CCORE library (C++ implementation of pyclustering).

5.29.3 Member Function Documentation

5.29.3.1 def pyclustering.nnet.legion.legion_network.simulate (*self*, *steps*, *time*, *stimulus*, *solution* = solve_type.RK4, *collect_dynamic* = True)

Performs static simulation of LEGION oscillatory network.

Parameters

in	*steps*	(uint): Number steps of simulations during simulation.
in	*time*	(double): Time of simulation.
in	*stimulus*	(list): Stimulus for oscillators, number of stimulus should be equal to number of oscillators, example of stimulus for 5 oscillators [0, 0, 1, 1, 0], value of stimulus is defined by parameter 'I'.
in	*solution*	(solve_type): Type of solution (solving).
in	*collect_dynamic*	(bool): If True - returns whole dynamic of oscillatory network, otherwise returns only last values of dynamics.

Returns

(list) Dynamic of oscillatory network. If argument 'collect_dynamic' = True, than return dynamic for the whole simulation time, otherwise returns only last values (last step of simulation) of dynamic.

The documentation for this class was generated from the following file:

- pyclustering/nnet/legion.py

5.30 pyclustering.nnet.legion.legion_parameters Class Reference

Describes parameters of LEGION.

Static Public Attributes

- float eps = 0.02

  Coefficient that affects intrinsic inhibitor of each oscillator.
- float alpha = 0.005

  Coefficient is chosen to be on the same order of magnitude as 'eps'.
- float gamma = 6.0

  Coefficient that is used to control the ratio of the times that the solution spends in these two phases.
- float betta = 0.1

  Coefficient that affects on intrinsic inhibitor of each oscillator.
- float lamda = 0.1

  Scale coefficient that is used by potential, should be greater than 0.
- float teta = 0.9

  Threshold that should be exceeded by a potential to switch on potential.
- float teta_x = -1.5

  Threshold that should be exceeded by a single oscillator to affect its neighbors.
- float teta_p = 1.5

  Threshold that should be exceeded to activate potential.
- float teta_xz = 0.1

  Threshold that should be exceeded by any oscillator to activate global inhibitor.
- float teta_zx = 0.1

  Threshold that should be exceeded to affect on a oscillator by the global inhibitor.
- float T = 2.0

  Weight of permanent connections.
- float mu = 0.01

  Defines time scaling of relaxing of oscillator potential.
- float Wz = 1.5

  Weight of global inhibitory connections.
- float Wt = 8.0

  Total dynamic weights to a single oscillator from neighbors.
- float fi = 3.0

  Rate at which the global inhibitor reacts to the stimulation from the oscillator network.
- float ro = 0.02

  Multiplier of oscillator noise.
- float I = 0.2

  Value of external stimulus.
- ENABLE_POTENTIONAL = True;

  Defines whether to use potentional of oscillator or not.

5.30.1 Detailed Description

Describes parameters of LEGION.

Contained parameters affect on output dynamic of each oscillator of the network.

See also

legion_network

5.30.2 Member Data Documentation

5.30.2.1 float pyclustering.nnet.legion.legion_parameters.alpha = 0.005 [static]

Coefficient is chosen to be on the same order of magnitude as 'eps'.

Affects on exponential function that decays on a slow time scale.

5.30.2.2 float pyclustering.nnet.legion.legion_parameters.betta = 0.1 [static]

Coefficient that affects on intrinsic inhibitor of each oscillator.

Specifies the steepness of the sigmoid function.

5.30.2.3 pyclustering.nnet.legion.legion_parameters.ENABLE_POTENTIONAL = True; [static]

Defines whether to use potentional of oscillator or not.

5.30.2.4 float pyclustering.nnet.legion.legion_parameters.eps = 0.02 [static]

Coefficient that affects intrinsic inhibitor of each oscillator.

Should be the same as 'alpha'.

5.30.2.5 float pyclustering.nnet.legion.legion_parameters.fi = 3.0 [static]

Rate at which the global inhibitor reacts to the stimulation from the oscillator network.

5.30.2.6 float pyclustering.nnet.legion.legion_parameters.gamma = 6.0 [static]

Coefficient that is used to control the ratio of the times that the solution spends in these two phases.

For a larger value of g, the solution spends a shorter time in the active phase.

5.30.2.7 float pyclustering.nnet.legion.legion_parameters.I = 0.2 [static]

Value of external stimulus.

5.30.2.8 float pyclustering.nnet.legion.legion_parameters.lamda = 0.1 [static]

Scale coefficient that is used by potential, should be greater than 0.

5.30.2.9 float pyclustering.nnet.legion.legion_parameters.mu = 0.01 [static]

Defines time scaling of relaxing of oscillator potential.

5.30.2.10 float pyclustering.nnet.legion.legion_parameters.ro = 0.02 [static]

Multiplier of oscillator noise.

Plays important role in desynchronization process.

5.30.2.11 float pyclustering.nnet.legion.legion_parameters.T = 2.0 [static]

Weight of permanent connections.

5.30.2.12 float pyclustering.nnet.legion.legion_parameters.teta = 0.9 [static]

Threshold that should be exceeded by a potential to switch on potential.

5.30.2.13 float pyclustering.nnet.legion.legion_parameters.teta_p = 1.5 [static]

Threshold that should be exceeded to activate potential.

If potential less than the threshold then potential is relaxed to 0 on time scale 'mu'.

5.30.2.14 float pyclustering.nnet.legion.legion_parameters.teta_x = -1.5 [static]

Threshold that should be exceeded by a single oscillator to affect its neighbors.

5.30.2.15 float pyclustering.nnet.legion.legion_parameters.teta_xz = 0.1 [static]

Threshold that should be exceeded by any oscillator to activate global inhibitor.

5.30.2.16 float pyclustering.nnet.legion.legion_parameters.teta_zx = 0.1 [static]

Threshold that should be exceeded to affect on a oscillator by the global inhibitor.

5.30.2.17 float pyclustering.nnet.legion.legion_parameters.Wt = 8.0 [static]

Total dynamic weights to a single oscillator from neighbors.

Sum of weights of dynamic connections to a single oscillator can not be bigger than Wt.

5.30.2.18 float pyclustering.nnet.legion.legion_parameters.Wz = 1.5 [static]

Weight of global inhibitory connections.

The documentation for this class was generated from the following file:

- pyclustering/nnet/legion.py

5.31 pyclustering.container.cftree.measurement_type Class Reference

Enumeration of measurement types for CF-Tree.

Inheritance diagram for pyclustering.container.cftree.measurement_type:

Static Public Attributes

- int CENTROID_EUCLIDIAN_DISTANCE = 0
  Euclidian distance between centroids of clustering features.

- int CENTROID_MANHATTAN_DISTANCE = 1

  Manhattan distance between centroids of clustering features.
- int AVERAGE_INTER_CLUSTER_DISTANCE = 2

  Average distance between all objects from clustering features.
- int AVERAGE_INTRA_CLUSTER_DISTANCE = 3

  Average distance between all objects within clustering features and between them.
- int VARIANCE_INCREASE_DISTANCE = 4

  Variance based distance between clustering features.

5.31.1 Detailed Description

Enumeration of measurement types for CF-Tree.

See also

cftree

5.31.2 Member Data Documentation

5.31.2.1 int pyclustering.container.cftree.measurement_type.AVERAGE_INTER_CLUSTER_DISTANCE = 2 [static]

Average distance between all objects from clustering features.

5.31.2.2 int pyclustering.container.cftree.measurement_type.AVERAGE_INTRA_CLUSTER_DISTANCE = 3 [static]

Average distance between all objects within clustering features and between them.

5.31.2.3 int pyclustering.container.cftree.measurement_type.CENTROID_EUCLIDIAN_DISTANCE = 0 [static]

Euclidian distance between centroids of clustering features.

5.31.2.4 int pyclustering.container.cftree.measurement_type.CENTROID_MANHATTAN_DISTANCE = 1 [static]

Manhattan distance between centroids of clustering features.

5.31.2.5 int pyclustering.container.cftree.measurement_type.VARIANCE_INCREASE_DISTANCE = 4 [static]

Variance based distance between clustering features.

The documentation for this class was generated from the following file:

- pyclustering/container/cftree.py

5.32 pyclustering.nnet.network Class Reference

Common network description.

Inheritance diagram for pyclustering.nnet.network:

Public Member Functions

- def __init__

  Constructor of the network.
- def __len__ (self)

  Returns size of the network that is defined by amount of oscillators.
- def has_connection (self, i, j)

  Returns strength of connection between i and j oscillators.
- def get_neighbors (self, index)

  Find neighbors of the oscillator with specified index.

5.32.1 Detailed Description

Common network description.

5.32.2 Constructor & Destructor Documentation

5.32.2.1 def pyclustering.nnet.network.__init__ (*self*, *num_osc*, *type_conn* = conn_type.ALL_TO_ALL, *conn_represent* = conn_represent.MATRIX)

Constructor of the network.

Parameters

in	*num_osc*	(uint): Number of oscillators in the network.
in	*type_conn*	(conn_type): Type of connections that are used in the network between oscillators.
in	*conn_represent*	(conn_represent): Type of representation of connections.

5.32.3 Member Function Documentation

5.32.3.1 def pyclustering.nnet.network.get_neighbors (*self*, *index*)

Find neighbors of the oscillator with specified index.

Parameters

in	*index*	(uint): index of oscillator in the network.

Returns

(list) Neighbors of the oscillator.

5.32.3.2 def pyclustering.nnet.network.has_connection (*self*, *i*, *j*)

Returns strength of connection between i and j oscillators.

Return 0 - if connection doesn't exist.

The documentation for this class was generated from the following file:

- pyclustering/nnet/__init__.py

5.33 pyclustering.container.kdtree.node Class Reference

Represents node of KD-Tree.

Public Member Functions

- def __init__
- def __repr__ (self)
- def __str__ (self)

Public Attributes

- data

    Data point that is presented as list of coodinates.
- payload

    Payload of node that can be used by user for storing specific information in the node.
- left

    Left node successor of the node.
- right

    Right node successor of the node.
- disc

    Index of dimension.
- parent

    Parent node of the node.

5.33.1 Detailed Description

Represents node of KD-Tree.

5.33.2 Constructor & Destructor Documentation

5.33.2.1 def pyclustering.container.kdtree.node.__init__ (*self*, *data* = None, *payload* = None, *left* = None, *right* = None, *disc* = None, *parent* = None)

Parameters

in	*data*	(list): Data point that is presented as list of coodinates.
in	*payload*	(*): Payload of node (pointer to essense that is attached to this node).
in	*left*	(node): Node of KD-Tree that is represented left successor.
in	*right*	(node): Node of KD-Tree that is represented right successor.
in	*disc*	(uint): Index of dimension of that node.
in	*parent*	(node): Node of KD-Tree that is represented parent.

5.33.3 Member Function Documentation

5.33.3.1 def pyclustering.container.kdtree.node.__repr__ (*self*)

Returns

(string) Default representation of the node.

5.33.3.2 def pyclustering.container.kdtree.node.__str__ (self)

Returns

(string) String representation of the node.

5.33.4 Member Data Documentation

5.33.4.1 pyclustering.container.kdtree.node.data

Data point that is presented as list of coodinates.

5.33.4.2 pyclustering.container.kdtree.node.disc

Index of dimension.

5.33.4.3 pyclustering.container.kdtree.node.left

Left node successor of the node.

5.33.4.4 pyclustering.container.kdtree.node.parent

Parent node of the node.

5.33.4.5 pyclustering.container.kdtree.node.payload

Payload of node that can be used by user for storing specific information in the node.

5.33.4.6 pyclustering.container.kdtree.node.right

Right node successor of the node.

The documentation for this class was generated from the following file:

- pyclustering/container/kdtree.py

5.34 pyclustering.container.cftree.non_leaf_node Class Reference

Representation of clustering feature non-leaf node.

Inheritance diagram for pyclustering.container.cftree.non_leaf_node:

Public Member Functions

- def successors (self)
- def __init__ (self, feature, parent, successors, payload)

  Create CF Non-leaf node.
- def __repr__ (self)
- def __str__ (self)
- def insert_successor (self, successor)

Insert successor to the node.
- def remove_successor (self, successor)

  Remove successor from the node.
- def merge (self, node)

  Merge non-leaf node to the current.
- def get_farthest_successors (self, type_measurement)

  Find pair of farthest successors of the node in line with measurement type.
- def get_nearest_successors (self, type_measurement)

  Find pair of nearest successors of the node in line with measurement type.

Public Attributes

- **type**

Additional Inherited Members

5.34.1 Detailed Description

Representation of clustering feature non-leaf node.

5.34.2 Constructor & Destructor Documentation

5.34.2.1 def pyclustering.container.cftree.non_leaf_node.__init__ (*self, feature, parent, successors, payload*)

Create CF Non-leaf node.

Parameters

in	*feature*	(cfentry): Clustering feature of the created node.
in	*parent*	(non_leaf_node): Parent of the created node.
in	*successors*	(list): List of successors of the node.
in	*payload*	(∗): Data that is stored by the node.

5.34.3 Member Function Documentation

5.34.3.1 def pyclustering.container.cftree.non_leaf_node.__repr__ (*self*)

Returns

(string) Representation of non-leaf node representation.

5.34.3.2 def pyclustering.container.cftree.non_leaf_node.__str__ (*self*)

Returns

(string) String non-leaf representation.

5.34.3.3 def pyclustering.container.cftree.non_leaf_node.get_farthest_successors (*self, type_measurement*)

Find pair of farthest successors of the node in line with measurement type.

Parameters

in	*type_measurement*	(measurement_type): Measurement type that is used for obtaining farthest successors.

Returns

(list) Pair of farthest successors represented by list [cfnode1, cfnode2].

5.34.3.4 def pyclustering.container.cftree.non_leaf_node.get_nearest_successors (*self, type_measurement*)

Find pair of nearest successors of the node in line with measurement type.

Parameters

in	*type_measurement*	(measurement_type): Measurement type that is used for obtaining nearest successors.

Returns

(list) Pair of nearest successors represented by list.

5.34.3.5 def pyclustering.container.cftree.non_leaf_node.insert_successor (*self, successor*)

Insert successor to the node.

Parameters

in	*successor*	(cfnode): Successor for adding.

5.34.3.6 def pyclustering.container.cftree.non_leaf_node.merge (*self, node*)

Merge non-leaf node to the current.

Parameters

in	*node*	(non_leaf_node): Non-leaf node that should be merged with current.

5.34.3.7 def pyclustering.container.cftree.non_leaf_node.remove_successor (*self, successor*)

Remove successor from the node.

Parameters

in	*successor*	(cfnode): Successor for removing.

5.34.3.8 def pyclustering.container.cftree.non_leaf_node.successors (*self*)

Returns

(list) List of successors of the node.

The documentation for this class was generated from the following file:

- pyclustering/container/cftree.py

5.35 pyclustering.cluster.optics.optics Class Reference

Class represents clustering algorithm OPTICS (Ordering Points To Identify Clustering Structure).

Public Member Functions

- def __init__ (self, sample, eps, minpts)

  Constructor of clustering algorithm OPTICS.
- def process (self)

  Performs cluster analysis in line with rules of OPTICS algorithm.
- def get_clusters (self)

  Returns list of allocated clusters, where each cluster contains indexes of objects and each cluster is represented by list.
- def get_noise (self)

  Returns list of noise that contains indexes of objects that corresponds to input data.
- def get_cluster_ordering (self)

  Returns clustering ordering information about the input data set.

5.35.1 Detailed Description

Class represents clustering algorithm OPTICS (Ordering Points To Identify Clustering Structure).

OPTICS is a density-based algorithm. Purpose of the algorithm is to provide explicit clusters, but create clustering-ordering representation of the input data. Clustering-ordering information contains information about internal structures of data set in terms of density.

Example:

```
# Read sample for clustering from some file
sample = read_sample(path_sample);

# Create OPTICS algorithm for cluster analysis
optics_instance = optics(sample, 0.5, 6);

# Run cluster analysis
optics_instance.process();

# Obtain results of clustering
clusters = optics_instance.get_clusters();
noise = optics_instance.get_noise();

# Obtain rechability-distances
ordering = optics_instance.get_cluster_ordering();

# Visualization of cluster ordering in line with reachability distance.
indexes = [i for i in range(0, len(ordering))];
plt.bar(indexes, ordering);
plt.show();
```

5.35.2 Constructor & Destructor Documentation

5.35.2.1 def pyclustering.cluster.optics.optics.__init__ (*self, sample, eps, minpts*)

Constructor of clustering algorithm OPTICS.

Parameters

in	*sample*	(list): Input data that is presented as a list of points (objects), where each point is represented by list or tuple.

in	*eps*	(double): Connectivity radius between points, points may be connected if distance between them less than the radius.
in	*minpts*	(uint): Minimum number of shared neighbors that is required for establishing links between points.

5.35.3 Member Function Documentation

5.35.3.1 def pyclustering.cluster.optics.optics.get_cluster_ordering (*self*)

Returns clustering ordering information about the input data set.

Clustering ordering of data-set contains the information about the internal clustering structure in line with connectivity radius.

Returns

(list) List of reachability distances (clustering ordering).

See also

process()
get_clusters()
get_noise()

5.35.3.2 def pyclustering.cluster.optics.optics.get_clusters (*self*)

Returns list of allocated clusters, where each cluster contains indexes of objects and each cluster is represented by list.

Returns

(list) List of allocated clusters.

See also

process()
get_noise()
get_cluster_ordering()

5.35.3.3 def pyclustering.cluster.optics.optics.get_noise (*self*)

Returns list of noise that contains indexes of objects that corresponds to input data.

Returns

(list) List of allocated noise objects.

See also

process()
get_clusters()
get_cluster_ordering()

5.35.3.4 def pyclustering.cluster.optics.optics.process (*self*)

Performs cluster analysis in line with rules of OPTICS algorithm.

Remarks

Results of clustering can be obtained using corresponding gets methods.

See also

get_clusters()
get_noise()
get_cluster_ordering()

The documentation for this class was generated from the following file:

- pyclustering/cluster/optics.py

5.36 pyclustering.cluster.optics.optics_descriptor Class Reference

Object description that used by OPTICS algorithm for cluster analysis.

Public Member Functions

- def __init__

  Constructor of object description in optics terms.
- def __repr__ (self)

  Returns string representation of the optics descriptor.

Static Public Attributes

- reachability_distance = None;

  Reachability distance - the smallest distance to be reachable by core object.
- core_distance = None;

  Core distance - the smallest distance to reach specified number of neighbors that is not greater then connectivity radius.
- processed = None;

  True is object has been already traversed.
- index_object = None;

  Index of object from the input data.

5.36.1 Detailed Description

Object description that used by OPTICS algorithm for cluster analysis.

5.36.2 Constructor & Destructor Documentation

5.36.2.1 def pyclustering.cluster.optics.optics_descriptor.__init__ (*self*, *index*, *core_distance* = None, *reachability_distance* = None)

Constructor of object description in optics terms.

Parameters

in	*index*	(uint): Index of the object in the data set.
in	*core_distance*	(double): Core distance that is minimum distance to specified number of neighbors.
in	*reachability_←distance*	(double): Reachability distance to this object.

5.36.3 Member Data Documentation

5.36.3.1 pyclustering.cluster.optics.optics_descriptor.core_distance = None; [static]

Core distance - the smallest distance to reach specified number of neighbors that is not greater then connectivity radius.

5.36.3.2 pyclustering.cluster.optics.optics_descriptor.index_object = None; [static]

Index of object from the input data.

5.36.3.3 pyclustering.cluster.optics.optics_descriptor.processed = None; [static]

True is object has been already traversed.

5.36.3.4 pyclustering.cluster.optics.optics_descriptor.reachability_distance = None; [static]

Reachability distance - the smallest distance to be reachable by core object.

The documentation for this class was generated from the following file:

- pyclustering/cluster/optics.py

5.37 pyclustering.nnet.pcnn.pcnn_dynamic Class Reference

Represents output dynamic of PCNN.

Public Member Functions

- def output (self)

  (list) Returns outputs of oscillator during simulation.
- def time (self)

  (list) Returns sampling times when dynamic is measured during simulation.
- def __init__

  Constructor of PCNN dynamic.
- def __del__ (self)

  Default destructor of PCNN dynamic.
- def __len__ (self)

  (uint) Returns number of simulation steps that are stored in dynamic.
- def allocate_sync_ensembles (self)

  Allocate clusters in line with ensembles of synchronous oscillators where each synchronous ensemble corresponds to only one cluster.
- def allocate_spike_ensembles (self)

Analyses output dynamic of network and allocates spikes on each iteration as a list of indexes of oscillators.

- def allocate_time_signal (self)

    Analyses output dynamic and calculates time signal (signal vector information) of network output.

5.37.1 Detailed Description

Represents output dynamic of PCNN.

5.37.2 Constructor & Destructor Documentation

5.37.2.1 def pyclustering.nnet.pcnn.pcnn_dynamic.__init__ (*self*, *dynamic*, *ccore* = None)

Constructor of PCNN dynamic.

Parameters

in	*dynamic*	(list): Dynamic of oscillators on each step of simulation. If ccore pointer is specified than it can be ignored.
in	*ccore*	(ctypes.pointer): Pointer to CCORE pcnn_dynamic instance in memory.

5.37.3 Member Function Documentation

5.37.3.1 def pyclustering.nnet.pcnn.pcnn_dynamic.allocate_spike_ensembles (*self*)

Analyses output dynamic of network and allocates spikes on each iteration as a list of indexes of oscillators.

Each allocated spike ensemble represents list of indexes of oscillators whose output is active.

Returns

(list) Spike ensembles of oscillators.

5.37.3.2 def pyclustering.nnet.pcnn.pcnn_dynamic.allocate_sync_ensembles (*self*)

Allocate clusters in line with ensembles of synchronous oscillators where each synchronous ensemble corresponds to only one cluster.

Returns

(list) Grours (lists) of indexes of synchronous oscillators. For example, [[index_osc1, index_osc3], [index_↩ osc2], [index_osc4, index_osc5]].

5.37.3.3 def pyclustering.nnet.pcnn.pcnn_dynamic.allocate_time_signal (*self*)

Analyses output dynamic and calculates time signal (signal vector information) of network output.

Returns

(list) Time signal of network output.

The documentation for this class was generated from the following file:

- pyclustering/nnet/pcnn.py

5.38 pyclustering.nnet.pcnn.pcnn_network Class Reference

Model of oscillatory network that is based on the Eckhorn model.

Inheritance diagram for pyclustering.nnet.pcnn.pcnn_network:

Public Member Functions

- def __init__

  Constructor of oscillatory network is based on Kuramoto model.
- def __del__ (self)

  Default destructor of PCNN.
- def __len__ (self)

  (uint) Returns size of oscillatory network.
- def simulate (self, steps, stimulus)

  Performs static simulation of pulse coupled neural network using.

5.38.1 Detailed Description

Model of oscillatory network that is based on the Eckhorn model.

Example:

```
# Create pulse-coupled neural network:
# - 9 oscillators.
# - default parameters.
# - grid type of connections (each oscillator has connection with four neighbors).
net = pcnn_network(9, None, conn_type.GRID_FOUR, ccore = ccore_flag);

# Create external stimulus. Number of stimulus should be equal to number of neurons.
stimulus = [1, 1, 1, 0, 0, 0, 1, 1, 1];

# Simulate dynamic of the network during 40 iterations
dynamic = net.simulate(40, stimulus);

# Allocate synchronous oscillators
ensembles = dynamic.allocate_sync_ensembles();
print(ensembles);

# Show output dynamic of the network
pcnn_visualizer.show_output_dynamic(dynamic);

# Show time signal vector information
pcnn_visualizer.show_time_signal(dynamic);
```

5.38.2 Constructor & Destructor Documentation

5.38.2.1 def pyclustering.nnet.pcnn.pcnn_network.__init__ (*self*, *num_osc*, *parameters* = None, *type_conn* = conn_type.ALL_TO_ALL, *type_conn_represent* = conn_represent.MATRIX, *ccore* = False)

Constructor of oscillatory network is based on Kuramoto model.

Parameters

in	*num_osc*	(uint): Number of oscillators in the network.
in	*parameters*	(pcnn_parameters): Parameters of the network.

in	*type_conn*	(conn_type): Type of connection between oscillators in the network (all-to-all, grid, bidirectional list, etc.).
in	*type_conn_↩ represent*	(conn_represent): Internal representation of connection in the network: matrix or list.
in	*ccore*	(bool): If True then all interaction with object will be performed via CCORE library (C++ implementation of pyclustering).

5.38.3 Member Function Documentation

5.38.3.1 def pyclustering.nnet.pcnn.pcnn_network.simulate (*self, steps, stimulus*)

Performs static simulation of pulse coupled neural network using.

Parameters

in	*steps*	(uint): Number steps of simulations during simulation.
in	*stimulus*	(list): Stimulus for oscillators, number of stimulus should be equal to number of oscillators.

Returns

(pcnn_dynamic) Dynamic of oscillatory network - output of each oscillator on each step of simulation.

The documentation for this class was generated from the following file:

- pyclustering/nnet/pcnn.py

5.39 pyclustering.nnet.pcnn.pcnn_parameters Class Reference

Parameters for pulse coupled neural network.

Static Public Attributes

- float VF = 1.0

  Multiplier for the feeding compartment at the current step.
- float VL = 1.0

  Multiplier for the linking compartment at the current step.
- float VT = 10.0

  Multiplier for the threshold at the current step.
- float AF = 0.1

  Multiplier for the feeding compartment at the previous step.
- float AL = 0.1

  Multiplier for the linking compartment at the previous step.
- float AT = 0.5

  Multiplier for the threshold at the previous step.
- float W = 1.0

  Synaptic weight - neighbours influence on linking compartment.
- float M = 1.0

  Synaptic weight - neighbours influence on feeding compartment.
- float B = 0.1

  Linking strength in the network.
- FAST_LINKING = False;

  Enable/disable Fast-Linking mode.

5.39.1 Detailed Description

Parameters for pulse coupled neural network.

5.39.2 Member Data Documentation

5.39.2.1 float pyclustering.nnet.pcnn.pcnn_parameters.AF = 0.1 [static]

Multiplier for the feeding compartment at the previous step.

5.39.2.2 float pyclustering.nnet.pcnn.pcnn_parameters.AL = 0.1 [static]

Multiplier for the linking compartment at the previous step.

5.39.2.3 float pyclustering.nnet.pcnn.pcnn_parameters.AT = 0.5 [static]

Multiplier for the threshold at the previous step.

5.39.2.4 float pyclustering.nnet.pcnn.pcnn_parameters.B = 0.1 [static]

Linking strength in the network.

5.39.2.5 pyclustering.nnet.pcnn.pcnn_parameters.FAST_LINKING = False; [static]

Enable/disable Fast-Linking mode.

Fast linking helps to overcome some of the effects of time quantisation. This process allows the linking wave to progress a lot faster than the feeding wave.

5.39.2.6 float pyclustering.nnet.pcnn.pcnn_parameters.M = 1.0 [static]

Synaptic weight - neighbours influence on feeding compartment.

5.39.2.7 float pyclustering.nnet.pcnn.pcnn_parameters.VF = 1.0 [static]

Multiplier for the feeding compartment at the current step.

5.39.2.8 float pyclustering.nnet.pcnn.pcnn_parameters.VL = 1.0 [static]

Multiplier for the linking compartment at the current step.

5.39.2.9 float pyclustering.nnet.pcnn.pcnn_parameters.VT = 10.0 [static]

Multiplier for the threshold at the current step.

The documentation for this class was generated from the following file:

- pyclustering/nnet/pcnn.py

5.40 pyclustering.nnet.pcnn.pcnn_visualizer Class Reference

Visualizer of output dynamic of pulse-coupled neural network (PCNN).

Static Public Member Functions

- def show_time_signal (pcnn_output_dynamic)

  Shows time signal (signal vector information) using network dynamic during simulation.
- def show_output_dynamic

  Shows output dynamic (output of each oscillator) during simulation.
- def animate_spike_ensembles (pcnn_output_dynamic, image_size)

  Shows animation of output dynamic (output of each oscillator) during simulation.

5.40.1 Detailed Description

Visualizer of output dynamic of pulse-coupled neural network (PCNN).

5.40.2 Member Function Documentation

5.40.2.1 def pyclustering.nnet.pcnn.pcnn_visualizer.animate_spike_ensembles (*pcnn_output_dynamic*, *image_size*) [static]

Shows animation of output dynamic (output of each oscillator) during simulation.

Parameters

in	*pcnn_output_dynamic*	(pcnn_dynamic): Output dynamic of the pulse-coupled neural network.
in	*image_size*	(list): Image size represented as [height, width].

5.40.2.2 def pyclustering.nnet.pcnn.pcnn_visualizer.show_output_dynamic (*pcnn_output_dynamic*, *separate_representation* = False) [static]

Shows output dynamic (output of each oscillator) during simulation.

Parameters

in	*pcnn_output_dynamic*	(pcnn_dynamic): Output dynamic of the pulse-coupled neural network.
in	*separate_representation*	(list): Consists of lists of oscillators where each such list consists of oscillator indexes that will be shown on separated stage.

5.40.2.3 def pyclustering.nnet.pcnn.pcnn_visualizer.show_time_signal (*pcnn_output_dynamic*) [static]

Shows time signal (signal vector information) using network dynamic during simulation.

Parameters

in	*pcnn_output_dynamic*	(pcnn_dynamic): Output dynamic of the pulse-coupled neural network.

The documentation for this class was generated from the following file:

- pyclustering/nnet/pcnn.py

5.41 pyclustering.cluster.rock.rock Class Reference

Class represents clustering algorithm ROCK.

Public Member Functions

- def __init__

  Constructor of clustering algorithm ROCK.
- def process (self)

  Performs cluster analysis in line with rules of ROCK algorithm.
- def get_clusters (self)

  Returns list of allocated clusters, each cluster contains indexes of objects in list of data.

5.41.1 Detailed Description

Class represents clustering algorithm ROCK.

Example:

```
# Read sample for clustering from some file
sample = read_sample(path_to_sample);

# Create instance of ROCK algorithm for cluster analysis
# Five clusters should be allocated
rock_instance = rock(sample, 1.0, 5);

# Run cluster analysis
rock_instance.process();

# Obtain results of clustering
clusters = rock_instance.get_clusters();
```

5.41.2 Constructor & Destructor Documentation

5.41.2.1 def pyclustering.cluster.rock.rock.__init__ (*self*, *data*, *eps*, *number_clusters*, *threshold* = `0.5`, *ccore* = `False`)

Constructor of clustering algorithm ROCK.

Parameters

in	*data*	(list): Input data - list of points where each point is represented by list of coordinates.
in	*eps*	(double): Connectivity radius (similarity threshold), points are neighbors if distance between them is less than connectivity radius.
in	*number_clusters*	(uint): Defines number of clusters that should be allocated from the input data set.
in	*threshold*	(double): Value that defines degree of normalization that influences on choice of clusters for merging during processing.
in	*ccore*	(bool): Defines should be CCORE (C++ pyclustering library) used instead of Python code or not.

5.41.3 Member Function Documentation

5.41.3.1 def pyclustering.cluster.rock.rock.get_clusters (*self*)

Returns list of allocated clusters, each cluster contains indexes of objects in list of data.

Returns

(list) List of allocated clusters, each cluster contains indexes of objects in list of data.

See also

process()

5.41.3.2 def pyclustering.cluster.rock.rock.process (*self*)

Performs cluster analysis in line with rules of ROCK algorithm.

Remarks

Results of clustering can be obtained using corresponding get methods.

See also

get_clusters()

The documentation for this class was generated from the following file:

- pyclustering/cluster/rock.py

5.42 pyclustering.nnet.solve_type Class Reference

Enumerator of solver types that are used for network simulation.

Inheritance diagram for pyclustering.nnet.solve_type:

Static Public Attributes

- int FAST = 0

  Forward Euler first-order method.
- int RK4 = 1

  Classic fourth-order Runge-Kutta method (fixed step).
- int RKF45 = 2

  Runge-Kutta-Fehlberg method with order 4 and 5 (float step).

5.42.1 Detailed Description

Enumerator of solver types that are used for network simulation.

5.42.2 Member Data Documentation

5.42.2.1 int pyclustering.nnet.solve_type.FAST = 0 `[static]`

Forward Euler first-order method.

5.42.2.2 int pyclustering.nnet.solve_type.RK4 = 1 [static]

Classic fourth-order Runge-Kutta method (fixed step).

5.42.2.3 int pyclustering.nnet.solve_type.RKF45 = 2 [static]

Runge-Kutta-Fehlberg method with order 4 and 5 (float step).

"

The documentation for this class was generated from the following file:

- pyclustering/nnet/__init__.py

5.43 pyclustering.nnet.som.som Class Reference

Represents self-organized feature map (SOM).

Public Member Functions

- def size (self)
- def weights (self)
- def awards (self)
- def capture_objects (self)
- def __init__

  Constructor of self-organized map.
- def __del__ (self)

  Destructor of the self-organized feature map.
- def __len__ (self)
- def train

  Trains self-organized feature map (SOM).
- def simulate (self, input_pattern)

  Processes input pattern (no learining) and returns index of neuron-winner.
- def get_winner_number (self)

  Calculates number of winner at the last step of learning process.
- def show_distance_matrix (self)

  Shows gray visualization of U-matrix (distance matrix).
- def get_distance_matrix (self)

  Calculates distance matrix (U-matrix).
- def show_density_matrix

  Show density matrix (P-matrix) using kernel density estimation.
- def get_density_matrix

  Calculates density matrix (P-Matrix).
- def show_winner_matrix (self)

  Show winner matrix where each element corresponds to neuron and value represents amount of won objects from input dataspace at the last training iteration.
- def show_network

  Shows neurons in the dimension of data.

5.43.1 Detailed Description

Represents self-organized feature map (SOM).

Example:

```
# sample for training
sample_train = read_sample(file_train_sample);

# create self-organized feature map with size 5x5
network = som(5, 5, sample_train, 100);

# train network
network.train();

# simulate using another sample
sample = read_sample(file_sample);
index_winner = network.simulate(sample);

# check what it is (what it looks like?)
index_similar_objects = network.capture_objects[index_winner];
```

5.43.2 Constructor & Destructor Documentation

5.43.2.1 def pyclustering.nnet.som.som.__init__ (*self*, *rows*, *cols*, *conn_type* = type_conn.grid_eight, *parameters* = None, *ccore* = False)

Constructor of self-organized map.

Parameters

in	*rows*	(uint): Number of neurons in the column (number of rows).
in	*cols*	(uint): Number of neurons in the row (number of columns).
in	*conn_type*	(type_conn): Type of connection between oscillators in the network (grid four, grid eight, honeycomb, function neighbour).
in	*parameters*	(som_parameters): Other specific parameters.
in	*ccore*	(bool): If True simulation is performed by CCORE library (C++ implementation of pyclustering).

5.43.3 Member Function Documentation

5.43.3.1 def pyclustering.nnet.som.som.__len__ (*self*)

Returns

(uint) Size of self-organized map (number of neurons).

5.43.3.2 def pyclustering.nnet.som.som.awards (*self*)

Returns

(list) Numbers of captured objects by each neuron.

5.43.3.3 def pyclustering.nnet.som.som.capture_objects (*self*)

Returns

(list) Indexes of captured objects by each neuron.

5.43.3.4 def pyclustering.nnet.som.som.get_density_matrix (*self*, *surface_divider* = `20.0`)

Calculates density matrix (P-Matrix).

Parameters

in	*surface_divider*	(double): Divider in each dimension that affect radius for density measurement.

Returns

(list) Density matrix (P-Matrix).

See also

get_distance_matrix()

5.43.3.5 def pyclustering.nnet.som.som.get_distance_matrix (*self*)

Calculates distance matrix (U-matrix).

The U-Matrix visualizes based on the distance in input space between a weight vector and its neighbors on map.

Returns

(list) Distance matrix (U-matrix).

See also

show_distance_matrix()
get_density_matrix()

5.43.3.6 def pyclustering.nnet.som.som.get_winner_number (*self*)

Calculates number of winner at the last step of learning process.

Returns

(uint) Number of winner.

5.43.3.7 def pyclustering.nnet.som.som.show_density_matrix (*self*, *surface_divider* = `20.0`)

Show density matrix (P-matrix) using kernel density estimation.

Parameters

in	*surface_divider*	(double): Divider in each dimension that affect radius for density measurement.

See also

show_distance_matrix()

5.43.3.8 def pyclustering.nnet.som.som.show_distance_matrix (*self*)

Shows gray visualization of U-matrix (distance matrix).

See also

get_distance_matrix()

5.43.3.9 def pyclustering.nnet.som.som.show_network (*self*, *awards* = `False`, *belongs* = `False`, *coupling* = `True`, *dataset* = `True`, *marker_type* = `'o'`)

Shows neurons in the dimension of data.

Parameters

in	*awards*	(bool): If True - displays how many objects won each neuron.
in	*belongs*	(bool): If True - marks each won object by according index of neuron-winner (only when dataset is displayed too).
in	*coupling*	(bool): If True - displays connections between neurons (except case when function neighbor is used).
in	*dataset*	(bool): If True - displays inputs data set.
in	*marker_type*	(string): Defines marker that is used for dispaying neurons in the network.

5.43.3.10 def pyclustering.nnet.som.som.show_winner_matrix (*self*)

Show winner matrix where each element corresponds to neuron and value represents amount of won objects from input dataspace at the last training iteration.

See also

show_distance_matrix()

5.43.3.11 def pyclustering.nnet.som.som.simulate (*self, input_pattern*)

Processes input pattern (no learining) and returns index of neuron-winner.

Using index of neuron winner catched object can be obtained using property capture_objects.

Parameters

in	*input_pattern*	(list): Input pattern.

Returns

(uint) Returns index of neuron-winner.

See also

capture_objects

5.43.3.12 def pyclustering.nnet.som.som.size (*self*)

Returns

(uint) Size of self-organized map (number of neurons).

5.43.3.13 def pyclustering.nnet.som.som.train (*self, data, epochs, autostop* = `False`)

Trains self-organized feature map (SOM).

Parameters

in	*data*	(list): Input data - list of points where each point is represented by list of features, for example coordinates.

in	*epochs*	(uint): Number of epochs for training.
in	*autostop*	(bool): Automatic termination of learining process when adaptation is not occurred.

Returns

(uint) Number of learining iterations.

5.43.3.14 def pyclustering.nnet.som.som.weights (*self*)

Returns

(list) Weights of each neuron.

The documentation for this class was generated from the following file:

- pyclustering/nnet/som.py

5.44 pyclustering.nnet.som.som_parameters Class Reference

Represents SOM parameters.

Static Public Attributes

- init_type = type_init.uniform_grid;

  Type of initialization of initial neuron weights (random, random in center of the input data, random distributed in data, ditributed in line with uniform grid).
- init_radius = None;

  Initial radius (if not specified then will be calculated by SOM).
- float init_learn_rate = 0.1

  Rate of learning.
- float adaptation_threshold = 0.001

  Condition when learining process should be stoped.

5.44.1 Detailed Description

Represents SOM parameters.

5.44.2 Member Data Documentation

5.44.2.1 float pyclustering.nnet.som.som_parameters.adaptation_threshold = 0.001 [static]

Condition when learining process should be stoped.

It's used when autostop mode is used.

5.44.2.2 float pyclustering.nnet.som.som_parameters.init_learn_rate = 0.1 [static]

Rate of learning.

5.44.2.3 pyclustering.nnet.som.som_parameters.init_radius = None; [static]

Initial radius (if not specified then will be calculated by SOM).

5.44.2.4 pyclustering.nnet.som.som_parameters.init_type = type_init.uniform_grid; [static]

Type of initialization of initial neuron weights (random, random in center of the input data, random distributed in data, ditributed in line with uniform grid).

The documentation for this class was generated from the following file:

- pyclustering/nnet/som.py

5.45 pyclustering.cluster.xmeans.splitting_type Class Reference

Enumeration of splitting types that can be used as splitting creation of cluster in X-Means algorithm.

Inheritance diagram for pyclustering.cluster.xmeans.splitting_type:

Static Public Attributes

- int BAYESIAN_INFORMATION_CRITERION = 0
  Bayesian information criterion to approximate the correct number of clusters.
- int MINIMUM_NOISELESS_DESCRIPTION_LENGTH = 1
  Minimum noiseless description length to approximate the correct number of clusters.

5.45.1 Detailed Description

Enumeration of splitting types that can be used as splitting creation of cluster in X-Means algorithm.

5.45.2 Member Data Documentation

5.45.2.1 int pyclustering.cluster.xmeans.splitting_type.BAYESIAN_INFORMATION_CRITERION = 0 [static]

Bayesian information criterion to approximate the correct number of clusters.

5.45.2.2 int pyclustering.cluster.xmeans.splitting_type.MINIMUM_NOISELESS_DESCRIPTION_LENGTH = 1 [static]

Minimum noiseless description length to approximate the correct number of clusters.

The documentation for this class was generated from the following file:

- pyclustering/cluster/xmeans.py

5.46 pyclustering.nnet.sync.sync_dynamic Class Reference

Represents output dynamic of Sync.

Inheritance diagram for pyclustering.nnet.sync.sync_dynamic:

Public Member Functions

- def output (self)

    (list) Returns outputs of oscillator during simulation.
- def time (self)

    (list) Returns sampling times when dynamic is measured during simulation.
- def __init__

    Constructor of Sync dynamic.
- def __del__ (self)

    Default destructor of Sync dynamic.
- def __len__ (self)

    (uint) Returns number of simulation steps that are stored in dynamic.
- def allocate_sync_ensembles

    Allocate clusters in line with ensembles of synchronous oscillators where each synchronous ensemble corresponds to only one cluster.
- def allocate_correlation_matrix

    Allocate correlation matrix between oscillators at the specified step of simulation.

5.46.1 Detailed Description

Represents output dynamic of Sync.

5.46.2 Constructor & Destructor Documentation

5.46.2.1 def pyclustering.nnet.sync.sync_dynamic.__init__ (*self, phase, time, ccore* = `None`)

Constructor of Sync dynamic.

Parameters

`in`	*phase*	(list): Dynamic of oscillators on each step of simulation. If ccore pointer is specified than it can be ignored.
`in`	*time*	(list): Simulation time.
`in`	*ccore*	(ctypes.pointer): Pointer to CCORE sync_dynamic instance in memory.

5.46.3 Member Function Documentation

5.46.3.1 def pyclustering.nnet.sync.sync_dynamic.allocate_correlation_matrix (*self, iteration* = `None`)

Allocate correlation matrix between oscillators at the specified step of simulation.

Parameters

`in`	*iteration*	(uint): Number of iteration of simulation for which correlation matrix should be allocated. If iternation number is not specified, the last step of simulation is used for the matrix allocation.

Returns

(list) Correlation matrix between oscillators with size [number_oscillators x number_oscillators].

5.46.3.2 def pyclustering.nnet.sync.sync_dynamic.allocate_sync_ensembles (*self, tolerance* = `0.01`)

Allocate clusters in line with ensembles of synchronous oscillators where each synchronous ensemble corresponds to only one cluster.

Parameters

in	*tolerance*	(double): Maximum error for allocation of synchronous ensemble oscillators.

Returns

(list) Grours (lists) of indexes of synchronous oscillators. For example [[index_osc1, index_osc3], [index_osc2], [index_osc4, index_osc5]].

The documentation for this class was generated from the following file:

- pyclustering/nnet/sync.py

5.47 pyclustering.nnet.sync.sync_network Class Reference

Model of oscillatory network that is based on the Kuramoto model of synchronization.

Inheritance diagram for pyclustering.nnet.sync.sync_network:

Public Member Functions

- def __init__

  Constructor of oscillatory network is based on Kuramoto model.
- def __del__ (self)

  Destructor of oscillatory network is based on Kuramoto model.
- def sync_order (self)

  Calculates level of global synchorization in the network.
- def sync_local_order (self)

  Calculates level of local (partial) synchronization in the network.
- def simulate

  Performs static simulation of Sync oscillatory network.
- def simulate_dynamic

  Performs dynamic simulation of the network until stop condition is not reached.
- def simulate_static

  Performs static simulation of oscillatory network.

5.47.1 Detailed Description

Model of oscillatory network that is based on the Kuramoto model of synchronization.

5.47.2 Constructor & Destructor Documentation

5.47.2.1 def pyclustering.nnet.sync.sync_network.__init__ (*self*, *num_osc*, *weight* = 1, *frequency* = 0, *type_conn* = conn_type.ALL_TO_ALL, *conn_represent* = conn_represent.MATRIX, *initial_phases* = initial_type.RANDOM_GAUSSIAN, *ccore* = False)

Constructor of oscillatory network is based on Kuramoto model.

Parameters

in	*num_osc*	(uint): Number of oscillators in the network.
in	*weight*	(double): Coupling strength of the links between oscillators.
in	*frequency*	(double): Multiplier of internal frequency of the oscillators.
in	*type_conn*	(conn_type): Type of connection between oscillators in the network (all-to-all, grid, bidirectional list, etc.).
in	*conn_represent*	(conn_represent): Internal representation of connection in the network: matrix or list.
in	*initial_phases*	(initial_type): Type of initialization of initial phases of oscillators (random, uniformly distributed, etc.).
in	*ccore*	(bool): If True simulation is performed by CCORE library (C++ implementation of pyclustering).

5.47.3 Member Function Documentation

5.47.3.1 def pyclustering.nnet.sync.sync_network.simulate (*self, steps, time, solution* = solve_type.FAST, *collect_dynamic* = `True`)

Performs static simulation of Sync oscillatory network.

Parameters

in	*steps*	(uint): Number steps of simulations during simulation.
in	*time*	(double): Time of simulation.
in	*solution*	(solve_type): Type of solution (solving).
in	*collect_dynamic*	(bool): If True - returns whole dynamic of oscillatory network, otherwise returns only last values of dynamics.

Returns

(list) Dynamic of oscillatory network. If argument 'collect_dynamic' = True, than return dynamic for the whole simulation time, otherwise returns only last values (last step of simulation) of dynamic.

See also

simulate_dynamic()
simulate_static()

5.47.3.2 def pyclustering.nnet.sync.sync_network.simulate_dynamic (*self, order* = `0.998`, *solution* = solve_type.FAST, *collect_dynamic* = `False`, *step* = `0.1`, *int_step* = `0.01`, *threshold_changes* = `0.0000001`)

Performs dynamic simulation of the network until stop condition is not reached.

Stop condition is defined by input argument 'order'.

Parameters

in	*order*	(double): Order of process synchronization, destributed 0..1.
in	*solution*	(solve_type): Type of solution.
in	*collect_dynamic*	(bool): If True - returns whole dynamic of oscillatory network, otherwise returns only last values of dynamics.

in	*step*	(double): Time step of one iteration of simulation.
in	*int_step*	(double): Integration step, should be less than step.
in	*threshold_changes*	(double): Additional stop condition that helps prevent infinite simulation, defines limit of changes of oscillators between current and previous steps.

Returns

(list) Dynamic of oscillatory network. If argument 'collect_dynamic' = True, than return dynamic for the whole simulation time, otherwise returns only last values (last step of simulation) of dynamic.

See also

simulate()
simulate_static()

5.47.3.3 def pyclustering.nnet.sync.sync_network.simulate_static (*self, steps, time, solution* = solve_type.FAST, *collect_dynamic* = False)

Performs static simulation of oscillatory network.

Parameters

in	*steps*	(uint): Number steps of simulations during simulation.
in	*time*	(double): Time of simulation.
in	*solution*	(solve_type): Type of solution.
in	*collect_dynamic*	(bool): If True - returns whole dynamic of oscillatory network, otherwise returns only last values of dynamics.

Returns

(list) Dynamic of oscillatory network. If argument 'collect_dynamic' = True, than return dynamic for the whole simulation time, otherwise returns only last values (last step of simulation) of dynamic.

See also

simulate()
simulate_dynamic()

5.47.3.4 def pyclustering.nnet.sync.sync_network.sync_local_order (*self*)

Calculates level of local (partial) synchronization in the network.

Returns

(double) Level of local (partial) synchronization.

See also

sync_order()

5.47.3.5 def pyclustering.nnet.sync.sync_network.sync_order (*self*)

Calculates level of global synchorization in the network.

Returns

(double) Level of global synchronization.

See also

sync_local_order()

The documentation for this class was generated from the following file:

- pyclustering/nnet/sync.py

5.48 pyclustering.nnet.sync.sync_visualizer Class Reference

Visualizer of output dynamic of sync network (Sync).

Static Public Member Functions

- def show_output_dynamic (sync_output_dynamic)
  Shows output dynamic (output of each oscillator) during simulation.
- def show_correlation_matrix
  Shows correlation matrix between oscillators at the specified iteration.
- def animate_output_dynamic
  Shows animation of output dynamic (output of each oscillator) during simulation on a circle from [0; 2pi].
- def animate_correlation_matrix
  Shows animation of correlation matrix between oscillators during simulation.

5.48.1 Detailed Description

Visualizer of output dynamic of sync network (Sync).

5.48.2 Member Function Documentation

5.48.2.1 def pyclustering.nnet.sync.sync_visualizer.animate_correlation_matrix (*sync_output_dynamic, animation_velocity =* 75) [static]

Shows animation of correlation matrix between oscillators during simulation.

Parameters

in	*sync_output_dynamic*	(sync_dynamic): Output dynamic of the Sync network.
in	*animation_velocity*	(uint): Interval between frames in milliseconds.

5.48.2.2 def pyclustering.nnet.sync.sync_visualizer.animate_output_dynamic (*sync_output_dynamic, animation_velocity =* 75) [static]

Shows animation of output dynamic (output of each oscillator) during simulation on a circle from [0; 2pi].

Parameters

in	*sync_output_↩ dynamic*	(sync_dynamic): Output dynamic of the Sync network.
in	*animation_↩ velocity*	(uint): Interval between frames in milliseconds.

5.48.2.3 def pyclustering.nnet.sync.sync_visualizer.show_correlation_matrix (*sync_output_dynamic*, *iteration* = None) [static]

Shows correlation matrix between oscillators at the specified iteration.

Parameters

in	*sync_output_↩ dynamic*	(sync_dynamic): Output dynamic of the Sync network.
in	*iteration*	(uint): Number of interation of simulation for which correlation matrix should be allocated. If iternation number is not specified, the last step of simulation is used for the matrix allocation.

5.48.2.4 def pyclustering.nnet.sync.sync_visualizer.show_output_dynamic (*sync_output_dynamic*) [static]

Shows output dynamic (output of each oscillator) during simulation.

Parameters

in	*sync_output_↩ dynamic*	(sync_dynamic): Output dynamic of the Sync network.

The documentation for this class was generated from the following file:

- pyclustering/nnet/sync.py

5.49 pyclustering.gcolor.sync.syncgcolor Class Reference

Oscillatory network based on Kuramoto model with negative and positive connections for graph coloring problem.

Inheritance diagram for pyclustering.gcolor.sync.syncgcolor:

Public Member Functions

- def __init__

  Constructor of the oscillatory network syncgcolor for graph coloring problem.
- def process

  Performs simulation of the network (performs solving of graph coloring problem).

5.49.1 Detailed Description

Oscillatory network based on Kuramoto model with negative and positive connections for graph coloring problem.

5.49.2 Constructor & Destructor Documentation

5.49.2.1 def pyclustering.gcolor.sync.syncgcolor.__init__ (*self, graph_matrix, positive_weight, negative_weight, reduction* = `None`)

Constructor of the oscillatory network syncgcolor for graph coloring problem.

Parameters

in	*graph_matrix*	(list): Graph represented by matrix.
in	*positive_weight*	(double): Value of weight of positive connections.
in	*negative_weight*	(double): Value of weight of negative connections.
in	*reduction*	(bool): Inverse degree of the processed graph.

5.49.3 Member Function Documentation

5.49.3.1 def pyclustering.gcolor.sync.syncgcolor.process (*self*, *order* = 0.998, *solution* = solve_type.FAST, *collect_dynamic* = False)

Performs simulation of the network (performs solving of graph coloring problem).

Parameters

in	*order*	(double): Defines when process of synchronization in the network is over, range from 0 to 1.
in	*solution*	(solve_type): defines type (method) of solving diff. equation.
in	*collect_dynamic*	(bool): If True - return full dynamic of the network, otherwise - last state of phases.

Returns

(syncnet_analyser) Returns analyser of results of coloring.

The documentation for this class was generated from the following file:

- pyclustering/gcolor/sync.py

5.50 pyclustering.gcolor.sync.syncgcolor_analyser Class Reference

Analyser of output dynamic of the oscillatory network syncgcolor.

Inheritance diagram for pyclustering.gcolor.sync.syncgcolor_analyser:

Public Member Functions

- def __init__ (self, phase, time, pointer_sync_analyser)

  Constructor of the analyser.
- def allocate_color_clusters

  Allocates clusters, when one cluster defines only one color.
- def allocate_map_coloring

  Allocates coloring map for graph that has been processed.

5.50.1 Detailed Description

Analyser of output dynamic of the oscillatory network syncgcolor.

5.50.2 Constructor & Destructor Documentation

5.50.2.1 def pyclustering.gcolor.sync.syncgcolor_analyser.__init__ (*self*, *phase*, *time*, *pointer_sync_analyser*)

Constructor of the analyser.

Parameters

in	*phase*	(list): Output dynamic of the oscillatory network, where one iteration consists of all phases of oscillators.
in	*time*	(list): Simulation time.
in	*pointer_sync_↩ analyser*	(POINTER): Pointer to CCORE analyser, if specified then other arguments can be omitted.

5.50.3 Member Function Documentation

5.50.3.1 def pyclustering.gcolor.sync.syncgcolor_analyser.allocate_color_clusters (*self, tolerance =* `0.1`)

Allocates clusters, when one cluster defines only one color.

Parameters

in	*tolerance*	(double): Defines maximum deviation between phases.

Returns

(list) Clusters [vertices with color 1], [vertices with color 2], ..., [vertices with color n].

5.50.3.2 def pyclustering.gcolor.sync.syncgcolor_analyser.allocate_map_coloring (*self, tolerance =* `0.1`)

Allocates coloring map for graph that has been processed.

Parameters

in	*tolerance*	(double): Defines maximum deviation between phases.

Returns

(list) Colors for each node (index of node in graph), for example [color1, color2, color2, ...].

The documentation for this class was generated from the following file:

- pyclustering/gcolor/sync.py

5.51 pyclustering.cluster.syncnet.syncnet Class Reference

Class represents clustering algorithm SyncNet.

Inheritance diagram for pyclustering.cluster.syncnet.syncnet:

Public Member Functions

- def __init__
  Contructor of the oscillatory network SYNC for cluster analysis.
- def __del__ (self)
  Destructor of oscillatory network is based on Kuramoto model.
- def process
  Peforms cluster analysis using simulation of the oscillatory network.
- def show_network (self)
  Shows connections in the network.

5.51.1 Detailed Description

Class represents clustering algorithm SyncNet.

SyncNet is bio-inspired algorithm that is based on oscillatory network that uses modified Kuramoto model.

Example:

```
# read sample for clustering from some file
sample = read_sample(path_to_file);

# create oscillatory network with connectivity radius 0.5 using CCORE (C++ implementation of pyclustering)
network = syncnet(sample, 0.5, ccore = True);

# run cluster analysis and collect output dynamic of the oscillatory network,
# network simulation is performed by Runge Kutta Fehlberg 45.
(dyn_time, dyn_phase) = network.process(0.998, solve_type.RFK45, True);

# show oscillatory network
network.show_network();

# obtain clustering results
clusters = network.get_clusters();

# show clusters
draw_clusters(sample, clusters);
```

5.51.2 Constructor & Destructor Documentation

5.51.2.1 def pyclustering.cluster.syncnet.syncnet.__init__ (*self*, *sample*, *radius*, *conn_repr* = conn_represent.MATRIX, *initial_phases* = initial_type.RANDOM_GAUSSIAN, *enable_conn_weight* = `False`, *ccore* = `False`)

Contructor of the oscillatory network SYNC for cluster analysis.

Parameters

in	*sample*	(list): Input data that is presented as list of points (objects), each point should be represented by list or tuple.
in	*radius*	(double): Connectivity radius between points, points should be connected if distance between them less then the radius.
in	*conn_repr*	(conn_represent): Internal representation of connection in the network: matrix or list. Ignored in case of usage of CCORE library.
in	*initial_phases*	(initial_type): Type of initialization of initial phases of oscillators (random, uniformly distributed, etc.).
in	*enable_conn_↵ weight*	(bool): If True - enable mode when strength between oscillators depends on distance between two oscillators. If False - all connection between oscillators have the same strength that equals to 1 (True).
in	*ccore*	(bool): Defines should be CCORE C++ library used instead of Python code or not.

5.51.3 Member Function Documentation

5.51.3.1 def pyclustering.cluster.syncnet.syncnet.process (*self*, *order* = `0.998`, *solution* = solve_type.FAST, *collect_dynamic* = `True`)

Peforms cluster analysis using simulation of the oscillatory network.

Parameters

in	*order*	(double): Order of synchronization that is used as indication for stopping processing.

in	*solution*	(solve_type): Specified type of solving diff. equation.
in	*collect_dynamic*	(bool): Specified requirement to collect whole dynamic of the network.

Returns

(syncnet_analyser) Returns analyser of results of clustering.

5.51.3.2 def pyclustering.cluster.syncnet.syncnet.show_network (*self*)

Shows connections in the network.

It supports only 2-d and 3-d representation.

The documentation for this class was generated from the following file:

- pyclustering/cluster/syncnet.py

5.52 pyclustering.cluster.syncnet.syncnet_analyser Class Reference

Performs analysis of output dynamic of the oscillatory network syncnet to extract information about cluster allocation.

Inheritance diagram for pyclustering.cluster.syncnet.syncnet_analyser:

Public Member Functions

- def __init__ (self, phase, time, pointer_sync_analyser)

  Constructor of the analyser.
- def __del__ (self)

  Desctructor of the analyser.
- def allocate_clusters

  Returns list of clusters in line with state of ocillators (phases).
- def allocate_noise (self)

  Returns allocated noise.

5.52.1 Detailed Description

Performs analysis of output dynamic of the oscillatory network syncnet to extract information about cluster allocation.

5.52.2 Constructor & Destructor Documentation

5.52.2.1 def pyclustering.cluster.syncnet.syncnet_analyser.__init__ (*self, phase, time, pointer_sync_analyser*)

Constructor of the analyser.

Parameters

in	*phase*	(list): Output dynamic of the oscillatory network, where one iteration consists of all phases of oscillators.

in	*time*	(list): Simulation time.
in	*pointer_sync_←analyser*	(POINTER): Pointer to CCORE analyser, if specified then other arguments can be omitted.

5.52.3 Member Function Documentation

5.52.3.1 def pyclustering.cluster.syncnet.syncnet_analyser.allocate_clusters (*self, eps* = `0.01`)

Returns list of clusters in line with state of oscillators (phases).

Parameters

in	*eps*	(double): Tolerance level that define maximal difference between phases of oscillators in one cluster.

Returns

(list) List of clusters, for example [[cluster1], [cluster2], ...].

See also

allocate_noise()

5.52.3.2 def pyclustering.cluster.syncnet.syncnet_analyser.allocate_noise (*self*)

Returns allocated noise.

Remarks

Allocated noise can be returned only after data processing (use method process() before). Otherwise empty list is returned.

Returns

(list) List of indexes that are marked as a noise.

See also

allocate_clusters()

The documentation for this class was generated from the following file:

- pyclustering/cluster/syncnet.py

5.53 pyclustering.cluster.syncsom.syncsom Class Reference

Class represents clustering algorithm SYNC-SOM.

Public Member Functions

- def som_layer (self)

  The first layer of the oscillatory network - self-organized feature map.
- def sync_layer (self)

The second layer of the oscillatory network based on Kuramoto model.

- def __init__ (self, data, rows, cols)

  Constructor of the double layer oscillatory network SYNC-SOM.
- def process

  Performs simulation of the oscillatory network.
- def get_som_clusters

  Returns clusters with SOM neurons that encode input features in line with result of synchronization in the second (Sync) layer.
- def get_clusters

  Returns clusters in line with ensembles of synchronous oscillators where each synchronous ensemble corresponds to only one cluster.
- def show_som_layer (self)

  Shows visual representation of the first (SOM) layer.
- def show_sync_layer (self)

  Shows visual representation of the second (Sync) layer.

5.53.1 Detailed Description

Class represents clustering algorithm SYNC-SOM.

SYNC-SOM is bio-inspired algorithm that is based on oscillatory network that uses self-organized feature map as the first layer.

Example:

```
# read sample for clustering
sample = read_sample(file);

# create oscillatory network for cluster analysis where the first layer has size 10x10
network = syncsom(sample, 10, 10);

# simulate network (perform cluster analysis) and collect output dynamic
(dyn_time, dyn_phase) = network.process(5, True, 0.998);

# obtain encoded clusters
encoded_clusters = network.get_som_clusters();

# obtain real clusters
clusters = network.get_clusters();

# show the first layer of the network
network.show_som_layer();

# show the second layer of the network
network.show_sync_layer();
```

5.53.2 Constructor & Destructor Documentation

5.53.2.1 def pyclustering.cluster.syncsom.syncsom.__init__ (*self, data, rows, cols*)

Constructor of the double layer oscillatory network SYNC-SOM.

Parameters

in	*data*	(list): Input data that is presented as list of points (objects), each point should be represented by list or tuple.
in	*rows*	(uint): Rows of neurons (number of neurons in column) in the input layer (self-organized feature map).

in	*cols*	(uint): Columns of neurons (number of neurons in row) in the input later (self-organized feature map).

5.53.3 Member Function Documentation

5.53.3.1 def pyclustering.cluster.syncsom.syncsom.get_clusters (*self*, *eps* = 0.1)

Returns clusters in line with ensembles of synchronous oscillators where each synchronous ensemble corresponds to only one cluster.

Parameters

in	*eps*	(double): Maximum error for allocation of synchronous ensemble oscillators.

Returns

(list) List of grours (lists) of indexes of synchronous oscillators that corresponds to index of objects.

See also

process()
get_som_clusters()

5.53.3.2 def pyclustering.cluster.syncsom.syncsom.get_som_clusters (*self*, *eps* = 0.1)

Returns clusters with SOM neurons that encode input features in line with result of synchronization in the second (Sync) layer.

Parameters

in	*eps*	(double): Maximum error for allocation of synchronous ensemble oscillators.

Returns

(list) List of clusters that are represented by lists of indexes of neurons that encode input data.

See also

process()
get_clusters()

5.53.3.3 def pyclustering.cluster.syncsom.syncsom.process (*self*, *number_neighbours*, *collect_dynamic* = False, *order* = 0.999)

Performs simulation of the oscillatory network.

Parameters

in	*number_↩ neighbours*	(uint): Number of neighbours that should be used for calculation average distance and creation connections between oscillators.

in	*collect_dynamic*	(bool): If True - returns whole dynamic of oscillatory network, otherwise returns only last values of dynamics.
in	*order*	(double): Order of process synchronization that should be considered as end of clustering, destributed 0..1.

Returns

(tuple) Dynamic of oscillatory network. If argument 'collect_dynamic' = True, than return dynamic for the whole simulation time, otherwise returns only last values (last step of simulation) of dynamic.

See also

get_som_clusters()
get_clusters()

The documentation for this class was generated from the following file:

- pyclustering/cluster/syncsom.py

5.54 pyclustering.nnet.som.type_conn Class Reference

Enumeration of connection types for SOM.

Inheritance diagram for pyclustering.nnet.som.type_conn:

Static Public Attributes

- int grid_four = 0

  Grid type of connections when each oscillator has connections with left, upper, right, lower neighbors.
- int grid_eight = 1

  Grid type of connections when each oscillator has connections with left, upper-left, upper, upper-right, right, right-lower, lower, lower-left neighbors.
- int honeycomb = 2

  Grid type of connections when each oscillator has connections with left, upper-left, upper-right, right, right-lower, lower-left neighbors.
- int func_neighbor = 3

  Grid type of connections when existance of each connection is defined by the SOM rule on each step of simulation.

5.54.1 Detailed Description

Enumeration of connection types for SOM.

See also

som

5.54.2 Member Data Documentation

5.54.2.1 int pyclustering.nnet.som.type_conn.func_neighbor = 3 [static]

Grid type of connections when existance of each connection is defined by the SOM rule on each step of simulation.

5.54.2.2 int pyclustering.nnet.som.type_conn.grid_eight = 1 [static]

Grid type of connections when each oscillator has connections with left, upper-left, upper, upper-right, right, right-lower, lower, lower-left neighbors.

5.54.2.3 int pyclustering.nnet.som.type_conn.grid_four = 0 [static]

Grid type of connections when each oscillator has connections with left, upper, right, lower neighbors.

5.54.2.4 int pyclustering.nnet.som.type_conn.honeycomb = 2 [static]

Grid type of connections when each oscillator has connections with left, upper-left, upper-right, right, right-lower, lower-left neighbors.

The documentation for this class was generated from the following file:

- pyclustering/nnet/som.py

5.55 pyclustering.utils.graph.type_graph_descr Class Reference

Enumeration of graph description.

Inheritance diagram for pyclustering.utils.graph.type_graph_descr:

Static Public Attributes

- int GRAPH_UNKNOWN = 0

  Unknown graph representation.
- int GRAPH_MATRIX_DESCR = 1

  Matrix graph representation.
- int GRAPH_VECTOR_DESCR = 2

  Vector graph representation.

5.55.1 Detailed Description

Enumeration of graph description.

Matrix representation is list of lists where number of rows equals number of columns and each element of square matrix determines whether there is connection between two vertices. For example: [[0, 1, 1], [1, 0, 1], [1, 1, 0]].

Vector representation is list of lists where index of row corresponds to index of vertex and elements of row consists of indexes of connected vertices. For example: [[1, 2], [0, 2], [0, 1]].

5.55.2 Member Data Documentation

5.55.2.1 int pyclustering.utils.graph.type_graph_descr.GRAPH_MATRIX_DESCR = 1 [static]

Matrix graph representation.

5.55.2.2 int pyclustering.utils.graph.type_graph_descr.GRAPH_UNKNOWN = 0 [static]

Unknown graph representation.

5.55.2.3 int pyclustering.utils.graph.type_graph_descr.GRAPH_VECTOR_DESCR = 2 [static]

Vector graph representation.

The documentation for this class was generated from the following file:

- pyclustering/utils/graph.py

5.56 pyclustering.nnet.som.type_init Class Reference

Enumeration of initialization types for SOM.

Inheritance diagram for pyclustering.nnet.som.type_init:

Static Public Attributes

- int random = 0

  Weights are randomly distributed using Gaussian distribution (0, 1).
- int random_centroid = 1

  Weights are randomly distributed using Gaussian distribution (input data centroid, 1).
- int random_surface = 2

  Weights are randomly distrbiuted using Gaussian distribution (input data centroid, surface of input data).
- int uniform_grid = 3

  Weights are distributed as a uniform grid that covers whole surface of the input data.

5.56.1 Detailed Description

Enumeration of initialization types for SOM.

See also

som

5.56.2 Member Data Documentation

5.56.2.1 int pyclustering.nnet.som.type_init.random = 0 [static]

Weights are randomly distributed using Gaussian distribution (0, 1).

5.56.2.2 int pyclustering.nnet.som.type_init.random_centroid = 1 [static]

Weights are randomly distributed using Gaussian distribution (input data centroid, 1).

5.56.2.3 int pyclustering.nnet.som.type_init.random_surface = 2 [static]

Weights are randomly distrbiuted using Gaussian distribution (input data centroid, surface of input data).

5.56.2.4 int pyclustering.nnet.som.type_init.uniform_grid = 3 [static]

Weights are distributed as a uniform grid that covers whole surface of the input data.

The documentation for this class was generated from the following file:

- pyclustering/nnet/som.py

5.57 pyclustering.cluster.agglomerative.type_link Class Reference

Enumerator of types of link between clusters.

Inheritance diagram for pyclustering.cluster.agglomerative.type_link:

Static Public Attributes

- int SINGLE_LINK = 0
  Nearest objects in clusters is considered as a link.
- int COMPLETE_LINK = 1
  Farthest objects in clusters is considered as a link.
- int AVERAGE_LINK = 2
  Average distance between objects in clusters is considered as a link.
- int CENTROID_LINK = 3
  Distance between centers of clusters is considered as a link.

5.57.1 Detailed Description

Enumerator of types of link between clusters.

5.57.2 Member Data Documentation

5.57.2.1 int pyclustering.cluster.agglomerative.type_link.AVERAGE_LINK = 2 [static]

Average distance between objects in clusters is considered as a link.

5.57.2.2 int pyclustering.cluster.agglomerative.type_link.CENTROID_LINK = 3 [static]

Distance between centers of clusters is considered as a link.

5.57.2.3 int pyclustering.cluster.agglomerative.type_link.COMPLETE_LINK = 1 [static]

Farthest objects in clusters is considered as a link.

5.57.2.4 int pyclustering.cluster.agglomerative.type_link.SINGLE_LINK = 0 [static]

Nearest objects in clusters is considered as a link.

The documentation for this class was generated from the following file:

- pyclustering/cluster/agglomerative.py

5.58 pyclustering.cluster.xmeans.xmeans Class Reference

Class represents clustering algorithm X-Means.

Public Member Functions

- def __init__

  Constructor of clustering algorithm X-Means.
- def process (self)

  Performs cluster analysis in line with rules of X-Means algorithm.
- def get_clusters (self)

  Returns list of allocated clusters, each cluster contains indexes of objects in list of data.
- def get_centers (self)

  Returns list of centers for allocated clusters.

5.58.1 Detailed Description

Class represents clustering algorithm X-Means.

Example:

```
# sample for cluster analysis (represented by list)
sample = read_sample(path_to_sample);

# create object of X-Means algorithm that uses CCORE for processing
xmeans_instance = xmeans(sample, [ [0.0, 0.5] ], ccore = True);

# run cluster analysis
xmeans_instance.process();

# obtain results of clustering
clusters = xmeans_instance.get_clusters();

# display allocated clusters
draw_clusters(sample, clusters);
```

5.58.2 Constructor & Destructor Documentation

5.58.2.1 def pyclustering.cluster.xmeans.xmeans.__init__ (*self*, *data*, *initial_centers*, *kmax* = 20, *tolerance* = 0.025, *criterion* = splitting_type.BAYESIAN_INFORMATION_CRITERION, *ccore* = False)

Constructor of clustering algorithm X-Means.

Parameters

in	*data*	(list): Input data that is presented as list of points (objects), each point should be represented by list or tuple.
in	*initial_centers*	(list): Initial coordinates of centers of clusters that are represented by list↩ : [center1, center2, ...].
in	*kmax*	(uint): Maximum number of clusters that can be allocated.
in	*tolerance*	(double): Stop condition for each iteration: if maximum value of change of centers of clusters is less than tolerance than algorithm will stop processing.
in	*criterion*	(splitting_type): Type of splitting creation.
in	*ccore*	(bool): Defines should be CCORE (C++ pyclustering library) used instead of Python code or not.

5.58.3 Member Function Documentation

5.58.3.1 def pyclustering.cluster.xmeans.xmeans.get_centers (*self*)

Returns list of centers for allocated clusters.

Returns

(list) List of centers for allocated clusters.

See also

process()
get_clusters()

5.58.3.2 def pyclustering.cluster.xmeans.xmeans.get_clusters (*self*)

Returns list of allocated clusters, each cluster contains indexes of objects in list of data.

Returns

(list) List of allocated clusters.

See also

process()
get_centers()

5.58.3.3 def pyclustering.cluster.xmeans.xmeans.process (*self*)

Performs cluster analysis in line with rules of X-Means algorithm.

Remarks

Results of clustering can be obtained using corresponding gets methods.

See also

get_clusters()
get_centers()

The documentation for this class was generated from the following file:

- pyclustering/cluster/xmeans.py