DeBaCl Documentation

Release 0.2.0

Brian P. Kent

July 29, 2013

CHAPTER

ONE

DENSITY-BASED CLUSTERING

DeBaCl is a Python library for estimation of density level set trees and nonparametric density-based clustering. Level set trees are based on the statistically-principled definition of clusters as modes of a probability density function. They are particularly useful for analyzing structure in complex datasets that exhibit multi-scale clustering behavior. DeBaCl is intended to promote the practical use of level set trees through improvements in computational efficiency, flexible algorithms, and an emphasis on modularity and user customizability.

The tutorial for DeBaCl is an IPython Notebook. It is viewable on nbviewer, or as a PDF at docs/debacl_tutorial.pdf.

The PDF user manual contains documentation for each function. It can be found in the GitHub repository at docs/debacl_manual.pdf. A paper describing the statistical background of level set trees and level set tree clustering is located in the repository, in the docs/ folder as well.

CHAPTER

TWO

GEOMETRIC TREE

Main functions and classes for the DEnsity-BAsed CLustering (DeBaCl) toolbox. Includes functions to construct and modify level set trees produced by standard geometric clustering on each level. Also defines tools for interactive data analysis and clustering with level set trees.

class `geom_tree.`**`ClusterGUI`**(*tree, X, form, output=['scatter'], size=30, f=None, fhat=None, **kwargs*)

Allow the user to interactively select level or mass values in a level set tree and to see the clusters at that level.

tree : LevelSetTree

X [2D numpy array] Original data matrix. Rows are observations.

form [string] Type of level set tree plot. Must be 'lambda' or 'alpha' for this GUI tool. See GeomTree.plot for more detail.

output [list of strings, optional] If output includes 'scatter' and the data has 3 or fewer dimensions, selecting a tree node produces a scatter plot showing the members of the selected node.

size [int, optional] If 'output' includes 'scatter', the size of the points in the scatterplot.

f [2D numpy array, optional] Any function. Arguments in the first column and values in the second. Plotted independently of the data as a blue curve, so does not need to have the same number of rows as values in 'x'. Typically this is the generating probability density function for a 1D simulation.

fhat [list of floats, optional] Density estimate values for the data in 'pts'. Plotted as a black curve, with points colored according to the selected component.

keyword arguments are passed to the GeomTree.plot method.

`getClusters`()

Return the cluster memberships for the currently selected level or mass value.

`handle_click`(*event*)

Deals with a user click on the interactive plot.

`show`()

Show the interactive plot canvas.

class `geom_tree.`**`ComponentGUI`**(*tree, X, form, f=None, fhat=None, output=['scatter'], size=30, **kwargs*)

Allow the user to interactively select level set tree nodes for tree coloring, subsetting, and scatter plots.

tree : GeomTree

X [2D numpy array] Original data matrix. Rows are observations. Must have 3 or fewer columns if the 'output' list includes 'scatter'.

output [list of strings, optional] If the list includes 'tree', selecting a LST node will plot the subtree with the selected node as the root. If output includes 'scatter' and the data has 3 or fewer dimensions, selecting a tree node produces a scatter plot showing the members of the selected node.

form [string] Type of level set tree plot. Must be 'lambda' or 'alpha' for this GUI tool. See GeomTree.plot for more detail.

f [2D numpy array, optional] Any function. Arguments in the first column and values in the second. Plotted independently of the data as a blue curve, so does not need to have the same number of rows as values in 'x'. Typically this is the generating probability density function for a 1D simulation.

fhat [list of floats, optional] Density estimate values for the data in 'pts'. Plotted as a black curve, with points colored according to the selected component.

keyword arguments are passed to the GeomTree.plot method.

`getComponent`()
Return the members of the currently selected level set tree node.

`getIndex`()
Return the index of the currently selected tree node.

`getSubtree`()
Return the subtree with the currently selected node as root.

`handle_pick`(*event*)
Return the members of the component that is picked out.

`show`()
Show the instantiated GUI window (i.e. the interactive LevelSetTree plot).

class `geom_tree.`**`ConnectedComponent`**(*idnum*, *parent*, *children*, *start_level*, *end_level*, *start_mass*, *end_mass*, *members*)
Defines a connected component for level set tree construction. A level set tree is really just a set of Connected-Components.

`copy`()
Creates and returns a copy of a ConnetedComponent object.

component : ConnectedComponent

class `geom_tree.`**`GeomTree`**(*bg_sets*, *levels*)
Defines methods and attributes for a level set tree, i.e. a collection of connected components organized hierarchically.

bg_sets [list of lists] The observations removed as background points at each successively higher density level.

levels [array_like] The probability density level associated with each element in 'bg_sets'.

`allModeCluster`()
Set every leaf node as a foreground cluster.

labels [2-dimensional numpy array] Each row corresponds to an observation. The first column indicates the index of the observation in the original data matrix, and the second column is the integer cluster label (starting at 0). Note that the set of observations in this "foreground" set is typically smaller than the original dataset.

leaves [list] Indices of tree nodes corresponding to foreground clusters. This is the same as 'nodes' for other clustering functions, but here they are also the leaves of the tree.

`collapseLeaves`(*active_nodes*)
Removes descendent nodes for the branches in 'active_nodes'.

active_nodes [array-like] List of nodes to use as the leaves in the collapsed tree.

`constructBranchMap`(*ix*, *interval*, *scale*, *width*, *sort*)
Map level set tree nodes to locations in a plot canvas. Finds the plot coordinates of vertical line segments corresponding to LST nodes and horizontal line segments corresponding to node splits. Also provides

indices of vertical segments and splits for downstream use with interactive plot picker tools. This function is not meant to be called by the user; it is a helper function for the LevelSetTree.plot() method. This function is recursive: it calls itself to map the coordinates of children of the current node 'ix'.

ix [int] The tree node to map.

interval: length 2 tuple of floats Horizontal space allocated to node 'ix'.

scale : {'lambda', 'alpha'}, optional

width [{'uniform', 'mass'}, optional] Determines how much horzontal space each level set tree node is given. See LevelSetTree.plot() for more information.

sort [bool] If True, sort sibling nodes from most to least points and draw left to right. Also sorts root nodes in the same way.

segments [dict] A dictionary with values that contain the coordinates of vertical line segment endpoints. This is only useful to the interactive analysis tools.

segmap [list] Indicates the order of the vertical line segments as returned by the recursive coordinate mapping function, so they can be picked by the user in the interactive tools.

splits [dict] Dictionary values contain the coordinates of horizontal line segments (i.e. node splits).

splitmap [list] Indicates the order of horizontal line segments returned by recursive coordinate mapping function, for use with interactive tools.

`constructMassMap`(*ix*, *start_pile*, *interval*, *width_mode*)

Map level set tree nodes to locations in a plot canvas. Finds the plot coordinates of vertical line segments corresponding to LST nodes and horizontal line segments corresponding to node splits. Also provides indices of vertical segments and splits for downstream use with interactive plot picker tools. This function is not meant to be called by the user; it is a helper function for the LevelSetTree.plot() method. This function is recursive: it calls itself to map the coordinates of children of the current node 'ix'. Differs from 'constructBranchMap' by setting the height of each vertical segment to be proportional to the number of points in the corresponding LST node.

ix [int] The tree node to map.

start_pile: float The height of the branch on the plot at it's start (i.e. lower terminus).

interval: length 2 tuple of floats Horizontal space allocated to node 'ix'.

width_mode [{'uniform', 'mass'}, optional] Determines how much horzontal space each level set tree node is given. See LevelSetTree.plot() for more information.

segments [dict] A dictionary with values that contain the coordinates of vertical line segment endpoints. This is only useful to the interactive analysis tools.

segmap [list] Indicates the order of the vertical line segments as returned by the recursive coordinate mapping function, so they can be picked by the user in the interactive tools.

splits [dict] Dictionary values contain the coordinates of horizontal line segments (i.e. node splits).

splitmap [list] Indicates the order of horizontal line segments returned by recursive coordinate mapping function, for use with interactive tools.

`findKCut`(*k*)

Find the lowest level cut that has k connected components. If there are no levels that have k components, then find the lowest level that has at least k components. If no levels have > k components, find the lowest level that has the maximum number of components.

k [int] Desired number of clusters/nodes/components.

cut [float] Lowest density level where there are k nodes.

`firstKCluster`(*k*)

Returns foreground cluster labels for the 'k' modes with the lowest start levels. In principle, this is the 'k' leaf nodes with the smallest indices, but this function double checks by finding and ordering all leaf start values and ordering.

k [integer] The desired number of clusters.

labels [2-dimensional numpy array] Each row corresponds to an observation. The first column indicates the index of the observation in the original data matrix, and the second column is the integer cluster label (starting at 0). Note that the set of observations in this "foreground" set is typically smaller than the original dataset.

nodes [list] Indices of tree nodes corresponding to foreground clusters.

`firstKLevelCluster`(*k*)

Use the first K clusters to appear in the level set tree as foreground clusters. In general, K-1 clusters will appear at a lower level than the K'th cluster; this function returns all members from all K clusters (rather than only the members in the upper level set where the K'th cluster appears). There are not always K clusters available in a level set tree - see LevelSetTree.findKCut for details on default behavior in this case.

k [int] Desired number of clusters.

labels [2-dimensional numpy array] Each row corresponds to an observation. The first column indicates the index of the observation in the original data matrix, and the second column is the integer cluster label (starting at 0). Note that the set of observations in this "foreground" set is typically smaller than the original dataset.

nodes [list] Indices of tree nodes corresponding to foreground clusters.

`getClusterLabels`(*method='all-mode'*, ***kwargs*)

Generic function for retrieving custer labels from the level set tree. Dispatches a specific cluster labeling function.

method [{'all-mode', 'first-k', 'upper-set', 'k-level'}, optional] Method for obtaining cluster labels from the tree. 'all-mode' treats each leaf of the tree as a separate cluter. 'first-k' finds the first K non-overlapping clusters from the roots of the tree. 'upper-set' returns labels by cutting the tree at a specified density (lambda) or mass (alpha) level. 'k-level' returns labels at the lowest density level that has k nodes.

k [integer] If method is 'first-k' or 'k-level', this is the desired number of clusters.

threshold [float] If method is 'upper-set', this is the threshold at which to cut the tree.

scale [{'lambda', 'alpha'}] If method is 'upper-set', this is vertical scale which 'threshold' refers to. 'lambda' corresponds to a density level, 'alpha' corresponds to a mass level.

labels [2-dimensional numpy array] Each row corresponds to an observation. The first column indicates the index of the observation in the original data matrix, and the second column is the integer cluster label (starting at 0). Note that the set of observations in this "foreground" set is typically smaller than the original dataset.

nodes [list] Indices of tree nodes corresponding to foreground clusters.

`makeSubtree`(*ix*)

Return the subtree with node 'ix' as the root, and all ancestors of 'ix'.

ix [int] Node to use at the root of the new tree.

T [LevelSetTree] A completely indpendent level set tree, with 'ix' as the root node.

massToLevel(*alpha*)

Convert the specified mass value into a level location.

alpha [float] Float in the interval [0.0, 1.0], with the desired fraction of all points.

cut_level: float Density level corresponding to the 'alpha' fraction of background points.

mergeBySize(*threshold*)

Prune splits from a tree based on size of child nodes. Merge members of child nodes rather than removing them.

threshold [numeric] Tree branches with fewer members than this will be merged into larger siblings or parents.

Modifies a level set tree in-place.

plot(*form*, *width='uniform'*, *sort=True*, *gap=0.05*, *color_nodes=None*)

Create a level set tree plot in Matplotlib.

form [{'lambda', 'alpha', 'kappa', 'old'}] Determines main form of the plot. 'lambda' is the traditional plot where the vertical scale is density levels, but plot improvements such as mass sorting of the nodes and colored nodes are allowed and the secondary 'alpha' scale is visible (but not controlling). The 'old' form uses density levels for vertical scale but does not allow plot tweaks and does not show the secondary 'alpha' scale. The 'alpha' setting makes the uppper level set mass the primary vertical scale, leaving the 'lambda' scale in place for reference. 'kappa' makes node mass the vertical scale, so that each node's vertical height is proportional to its mass excluding the mass of the node's children.

width [{'uniform', 'mass'}, optional] Determines how much horzontal space each level set tree node is given. The default of "uniform" gives each child node an equal fraction of the parent node's horizontal space. If set to 'mass', then horizontal space is allocated proportional to the mass (i.e. fraction of points) of a node relative to its siblings.

sort [bool, optional] If True, sort sibling nodes from most to least points and draw left to right. Also sorts root nodes in the same way.

gap [float, optional] Fraction of vertical space to leave at the bottom. Default is 5%, and 0% also works well. Higher values are used for interactive tools to make room for buttons and messages.

color_nodes [list, optional] Each entry should be a valid index in the level set tree that will be colored uniquely.

fig [matplotlib figure] Use fig.show() to view, fig.savefig() to save, etc.

segments [dict] A dictionary with values that contain the coordinates of vertical line segment endpoints. This is only useful to the interactive analysis tools.

segmap [list] Indicates the order of the vertical line segments as returned by the recursive coordinate mapping function, so they can be picked by the user in the interactive tools.

splits [dict] Dictionary values contain the coordinates of horizontal line segments (i.e. node splits).

splitmap [list] Indicates the order of horizontal line segments returned by recursive coordinate mapping function, for use with interactive tools.

prune(*method='size-merge'*, ***kwargs*)

Prune the tree. A dispatch function to other methods.

method : {'size-merge'}

gamma [integer] Nodes smaller than this will be merged (for 'size-merge') or cut (for 'size-cut')

Modifies the tree in-place.

`save` (*fname*)

Save a level set tree object to file.

Saves a level set tree as a MATLAB struct using the scipy.io module. Ignore the warning about using oned_as default value ('column').

fname [string] File to save the tree to. The .mat extension is not necessary.

`upperSetCluster` (*threshold*, *scale='alpha'*)

Set foreground clusters by finding connected components at an upper level set or upper mass set.

threshold [float] The level or mass value that defines the foreground set of points, depending on 'scale'.

scale [{'alpha', 'lambda'}] Determines if the 'cut' threshold is a density level value or a mass value (i.e. fraction of data in the background set)

labels [2-dimensional numpy array] Each row corresponds to an observation. The first column indicates the index of the observation in the original data matrix, and the second column is the integer cluster label (starting at 0). Note that the set of observations in this "foreground" set is typically smaller than the original dataset.

nodes [list] Indices of tree nodes corresponding to foreground clusters.

`geom_tree.constructTree` (*W*, *levels*, *bg_sets*, *mode='general'*, *verbose=False*)

Construct a level set tree. A level set tree is constructed by identifying connected components of observations at successively higher levels of a probability density estimate.

W [2D array] An adjacency matrix for a similarity graph on the data.

levels: array Defines the density levels where connected components will be computed. Typically this includes all unique values of a function evaluated on the data points.

bg_sets: list of lists Specify which points to remove as background at each density level in 'levels'.

mode: {'general', 'density'}, optional Establish if the values in 'levels' come from a probability density or pseudo-density estimate, in which case there is a natural floor at 0. The default is to model an arbitrary function, which requires a run-time choice of floor value.

verbose: {False, True}, optional If set to True, then prints to the screen a progress indicator every 100 levels.

T [levelSetTree] See debacl.levelSetTree for class and method definitions.

`geom_tree.geomTree` (*X*, *k*, *gamma*, *n_grid=None*, *verbose=True*)

Construct a level set tree, from soup to nuts. This function assumes a k-nearest neighbor similarity graph and k-nearest neighbor density estimate, which is less flexible than building the similarity graphy, estimating the (pseudo-) density, constructing the tree, and pruning with separate functions.

X [2-dimensional numpy array] The data matrix. Rows are observations.

k [integer] Number of observations to consider as neighbors to a given point, in both the k-nearest neighbor similarity graph and k-nearest neighbor density estimate.

n_grid [integer] Number of cells in the density level mesh. If None, level sets are computed and decomposed as each observation is removed from the upper level set.

gamma [integer] Size threshold for pruning small leaf nodes of the tree. Uses the sizeMerge function to prune.

verbose [boolean, optional] Prints progress updates to the screen if True, the default.

T [LevelSetTree object] The pruned level set tree estimated from a k-nearest neighbor similarity graph and density estimate.

`geom_tree.`**`loadTree`**(*fname*)

Load a saved tree from file.

fname [string] Filename to load. The .mat extension is not necessary.

T [LevelSetTree] The loaded and reconstituted level set tree object.

CHAPTER THREE

CHAUDHURI-DASGUPTA TREE

Main functions and classes for construction and use of Chaudhuri-Dasgupta level set trees. A companion to debacl.py, which has a more developed set of tools for working with generic level set trees.

class `cd_tree.`**`CDTree`**
Defines methods and attributes for a Chaudhuri-Dasgupta level set tree.

`allModeCluster``()`
Set every leaf node as a foreground cluster.

labels [2-dimensional numpy array] Each row corresponds to an observation. The first column indicates the index of the observation in the original data matrix, and the second column is the integer cluster label (starting at 0). Note that the set of observations in this "foreground" set is typically smaller than the original dataset.

leaves [list] Indices of tree nodes corresponding to foreground clusters. This is the same as 'nodes' for other clustering functions, but here they are also the leaves of the tree.

`constructBranchMap``(`*ix*, *interval*, *width*`)`
Map level set tree nodes to locations in a plot canvas. Finds the plot coordinates of vertical line segments corresponding to LST nodes and horizontal line segments corresponding to node splits. Also provides indices of vertical segments and splits for downstream use with interactive plot picker tools. This function is not meant to be called by the user; it is a helper function for the LevelSetTree.plot() method. This function is recursive: it calls itself to map the coordinates of children of the current node 'ix'.

ix [int] The tree node to map.

interval: length 2 tuple of floats Horizontal space allocated to node 'ix'.

width [{'uniform', 'mass'}, optional] Determines how much horzontal space each level set tree node is given. See LevelSetTree.plot() for more information.

segments [dict] A dictionary with values that contain the coordinates of vertical line segment endpoints. This is only useful to the interactive analysis tools.

segmap [list] Indicates the order of the vertical line segments as returned by the recursive coordinate mapping function, so they can be picked by the user in the interactive tools.

splits [dict] Dictionary values contain the coordinates of horizontal line segments (i.e. node splits).

splitmap [list] Indicates the order of horizontal line segments returned by recursive coordinate mapping function, for use with interactive tools.

`firstKCluster``(`*k*`)`
Returns foreground cluster labels for the 'k' modes with the lowest start levels. In principle, this is the 'k' leaf nodes with the smallest indices, but this function double checks by finding and ordering all leaf start values and ordering.

k [integer] The desired number of clusters.

labels [2-dimensional numpy array] Each row corresponds to an observation. The first column indicates the index of the observation in the original data matrix, and the second column is the integer cluster label (starting at 0). Note that the set of observations in this "foreground" set is typically smaller than the original dataset.

nodes [list] Indices of tree nodes corresponding to foreground clusters.

`getClusterLabels`(*method='all-mode'*, ***kwargs*)

Umbrella function for retrieving custer labels from the level set tree.

method [{'all-mode', 'first-k', 'upper-set', 'k-level'}, optional] Method for obtaining cluster labels from the tree. 'all-mode' treats each leaf of the tree as a separate cluter. 'first-k' finds the first K non-overlapping clusters from the roots of the tree. 'upper-set' returns labels by cutting the tree at a specified density (lambda) or mass (alpha) level. 'k-level' returns labels at the lowest density level that has k nodes.

k [integer] If method is 'first-k' or 'k-level', this is the desired number of clusters.

threshold [float] If method is 'upper-set', this is the threshold at which to cut the tree.

labels [2-dimensional numpy array] Each row corresponds to an observation. The first column indicates the index of the observation in the original data matrix, and the second column is the integer cluster label (starting at 0). Note that the set of observations in this "foreground" set is typically smaller than the original dataset.

nodes [list] Indices of tree nodes corresponding to foreground clusters.

`makeSubtree`(*ix*)

Return the subtree with node 'ix' as the root, and all ancestors of 'ix'.

ix [int] Node to use at the root of the new tree.

T [LevelSetTree] A completely indpendent level set tree, with 'ix' as the root node.

`mergeBySize`(*threshold*)

Prune splits from a tree based on size of child nodes. Merge members of child nodes rather than removing them.

threshold [numeric] Tree branches with fewer members than this will be merged into larger siblings or parents.

Modifies a level set tree in-place.

`plot`(*width='uniform'*, *gap=0.05*)

Create a static plot of a Chaudhuri-Dasgupta level set tree.

width [{'uniform', 'mass'}, optional] Determines how much horzontal space each level set tree node is given. The default of "uniform" gives each child node an equal fraction of the parent node's horizontal space. If set to 'mass', then horizontal space is allocated proportional to the mass (i.e. fraction of points) of a node relative to its siblings.

sort [bool] If True, sort sibling nodes from most to least points and draw left to right. Also sorts root nodes in the same way.

gap [float] Fraction of vertical space to leave at the bottom. Default is 5%, and 0% also works well. Higher values are used for interactive tools to make room for buttons and messages.

fig [matplotlib figure] Use fig.show() to view, fig.savefig() to save, etc.

prune (*method='size-merge'*, ***kwargs*)
Prune the tree. A dispatch function to other methods.

method : {'size-merge'}

gamma [integer] Nodes smaller than this will be merged (for 'size-merge') or cut (for 'size-cut')

Modifies the tree in-place.

save (*fname*)
Save a level set tree object to file.

Saves a level set tree as a MATLAB struct using the scipy.io module. Ignore the warning about using oned_as default value ('column').

fname [string] File to save the tree to. The .mat extension is not necessary.

upperSetCluster (*threshold*)
Set foreground clusters by finding connected components at an upper level set. This is slightly different than GeomTree.upperSetCluster in that this method returns all members of tree nodes that cross the desired threshold, rather than the components of the true upper level set.

threshold [float] The radius that defines the foreground set of points.

labels [2-dimensional numpy array] Each row corresponds to an observation. The first column indicates the index of the observation in the original data matrix, and the second column is the integer cluster label (starting at 0). Note that the set of observations in this "foreground" set is typically smaller than the original dataset.

nodes [list] Indices of tree nodes corresponding to foreground clusters.

class `cd_tree.`**ConnectedComponent** (*idnum*, *parent*, *children*, *start_radius*, *end_radius*, *members*)
Defines a connected component for level set tree construction. A level set tree is really just a set of Connected-Components.

copy ()
Creates and returns a copy of a CD_Component object.

component : CD_Component

`cd_tree.`**cdTree** (*X*, *k*, *alpha=1.0*, *start='complete'*, *verbose=False*)
Construct a Chaudhuri-Dasgupta level set tree. A level set tree is constructed by identifying connected components of observations as edges are removed from the geometric graph in descending order of pairwise distance.

X [2D array] Data matrix, with observations as rows.

k [integer] Number of observations to consider as neighbors of each point.

alpha [float] A robustness parameter. Dilates the threshold for including edges in an upper level set similarity graph.

start [{'complete', 'knn'}, optional] Initialization of the similarity graph. 'Complete' starts with a complete similarity graph (as written in the Chaudhuri-Dasgupta paper) and knn starts with a k-nearest neighbor similarity graph.

verbose: {False, True}, optional If set to True, then prints to the screen a progress indicator every 100 levels.

T [levelSetTree] See debacl.levelSetTree for class and method definitions.

`cd_tree.`**loadTree** (*fname*)
Load a saved tree from file.

fname [string] Filename to load. The .mat extension is not necessary.

T [LevelSetTree] The loaded and reconstituted level set tree object.

CHAPTER

FOUR

UTILITIES

General utility functions for the DEnsity-BAsed CLustering (DeBaCl) toolbox.

class `utils.`**`Palette`**(*use='scatter'*)

Define some good RGB sscolors manually to simplify plotting upper level sets and foreground clusters.

use [{'scatter', 'lines', 'neuroimg'}, optional] Application for the palette. Different palettes work better in different settings.

`applyColorset`(*ix*)

Turn a numpy array of group labels (integers) into RGBA colors.

`utils.`**`assignBackgroundPoints`**(*X*, *clusters*, *method=None*, *k=1*)

Assign level set tree background points to existing foreground clusters. This function packages a few very basic classification methods. Any classification method could work for this step of the data segmentation pipeline.

X [2-dimensional numpy array] The original data, with rows as observations.

clusters [2D numpy array] Foreground cluster assignments. Observation index is in the first entry of each row, with cluster label in the second entry. This is exactly what is returned by any of the LevelSetTree clustering methods.

method [{None, 'centers', 'knn', 'zero'}, optional] Which classification technique to use. The default of None sets background points to be a separate cluster. Option 'zero' does the same, but resets the cluster labels to the background points are labeled as '0'. The 'knn' method does a k-nearest neighbor classified, while option 'centers' assigns each background point to the cluster with the closet center (mean) point.

k [int, optional] If 'method' is 'knn', this is the number of neighbors to use for each observation.

labels [2-dimensional numpy array] Follows the same pattern as the 'clusters' parameter: each row is a data point, with the first entry as the observation index and the second entry the integer cluster label. Here though all points should be assigned, so the first column is just 1, ..., n, where n is the number of points.

`utils.`**`clusterHistogram`**(*x*, *cluster*, *fhat=None*, *f=None*, *levels=None*)

Plot a histogram and illustrate the location of selected cluster points.

The primary plot axis is a histogram. Under this plot is a second axis that shows the location of the points in 'cluster', colored according to cluster label. If specified, also plot a density estimate, density function (or any function), and horizontal guidelines. This is the workhorse of the DeBaCl interactive tools for 1D data.

x [1D numpy array of floats] The data.

cluster [2D numpy array] A cluster matrix: rows represent points in 'x', with first entry as the index and second entry as the cluster label. The output of all LevelSetTree clustering methods are in this format.

fhat [list of floats, optional] Density estimate values for the data in 'x'. Plotted as a black curve, with points colored according to 'cluster'.

f [2D numpy array, optional] Any function. Arguments in the first column and values in the second. Plotted independently of the data as a blue curve, so does not need to have the same number of rows as values in 'x'. Typically this is the generating probability density function for a 1D simulation.

levels [list of floats, optional] Each entry in 'levels' causes a horizontal dashed red line to appear at that value.

fig [matplotlib figure] Use fig.show() to show the plot, fig.savefig() to save it, etc.

`utils.`**`constructDensityGrid`**(*density*, *mode='mass'*, *n_grid=None*)

Create the inputs to a level set tree object. Create a list of lists of points to remove at each iteration of a level set or mass tree. Also create a list of the density level at each iteration.

density [1D numpy array] An array with one value for each data point. Typically this is a density estimate, but it can be any function.

mode [{'mass', 'levels'}, optional] Determines if the tree should be built by removing a constant number of points (mass) at each iteration, or on a grid of evenly spaced density levels. If 'n_grid' is set to None, the 'mass' option will remove 1 point at a time and the 'levels' option will iterate through every unique value of the 'density' array.

n_grid [int, optional] The number of tree heights at which to estimate connected components. This is essentially the resolution of a level set tree built for the 'density' array.

bg_sets [list of lists] Defines the points to remove as background at each iteration of level set tree construction.

levels [array-like] The density level at each iteration of level set tree construction.

`utils.`**`drawSample`**(*n*, *k*)

Draw a sample of size k from n items without replacement. Chooses k indices from range(n) without replacement by shuffling range(n) uniformly over all permutations. In numpy 1.7 and beyond, the "choice" function is a better option.

n [int] Total number of objects.

k [int] Sample size.

ix_keep [list of ints] Indices of objects selected in the sample.

`utils.`**`epsilonGraph`**(*x*, *eps=None*, *q=0.05*, *self_edge=False*)

Constructs an epsilon-neighborhood graph adjacency matrix. Constructs a graph where the rows of 'x' are vertices and pairs of vertices are connected by edges if they are within euclidean distance epsilon of each other. Return the adjacency matrix for this graph.

x [2D numpy array] The rows of x are the observations which become graph vertices.

eps [float, optional] The distance threshold for neighbors. If unspecified, defaults to the proportion in 'q'.

q [float, optional] If 'eps' is unspecified, this determines the neighbor threshold distance. 'eps' is set to the 'q' quantile of all (n choose 2) pairwise distances, where n is the number of rows in 'x'.

self_edge [boolean, boolean] Flag to include or exclude (default) self-edges. Equivalent to having 1's (self-edge = True) or 0's (self-edge = False) on the diagonal of the adjacency matrix.

W [2-dimensional numpy array of booleans] The adjacency matrix for the graph.

eps: float The neighbor threshold distance, useful particularly if not initially specified.

`utils.`**`gaussianGraph`**(*x*, *sigma*, *self_edge=False*)

Constructs a complete graph adjacency matrix with a Gaussian similarity kernel. Uses the rows of 'x' as vertices in a graph and connects each pair of vertices with an edge whose weight is the Gaussian kernel of the distance between the two vertices.

x [2D numpy array] Rows of ‘x’ are locations of graph vertices.

sigma [float] The denominator of the Gaussian kernel.

self_edge [boolean, optional] Flag to include or exclude (default) self-edges. Equivalent to having 1’s (self-edge = True) or 0’s (self-edge = False) on the diagonal of the adjacency matrix.

W [2-dimensional numpy array of floats] Adjacency matrix of the Gaussian kernel complete graph on rows of ‘x’. Each entry is a float representing the gaussian similarity between the corresponding rows of ‘x’.

`utils.`**`knnDensity`**(*k_radius*, *n*, *p*, *k*)

Compute the kNN density estimate for a set of points.

k_radius [1-dimensional numpy array of floats] The distance to each points k’th nearest neighbor.

n [int] The number of points.

p [int] The dimension of the data.

k [int] The number of observations considered neighbors of each point.

fhat [1D numpy array of floats] Estimated density for the points corresponding to the entries of ‘k_radius’.

`utils.`**`knnGraph`**(*x*, *k=None*, *q=0.05*, *self_edge=False*)

Compute the symmetric k-NN adjacency matrix for a set of points.

x [numpy array] Data points, with each row as an observation.

k [int, optional] The number of points to consider as neighbors of any given observation. If not specified, use the default value of ‘q’.

q [float, optional] The proportion of points to use as neighbors of a given observation. Defaults to 0.05.

self_edge [boolean, optional] Flag to include or exclude (default) self-edges. Equivalent to having 1’s (self-edge = True) or 0’s (self-edge = False) on the diagonal of the adjacency matrix.

W [2-dimensional numpy array of booleans] A 2D numpy array of shape n x n, where n is the number of rows in ‘x’. The entry at position (i, j) is True if observations i and j are neighbors, False otherwise.

k_radius [list of float] For each row of ‘x’ the distance to its k-1’th nearest neighbor.

`utils.`**`makeColorMatrix`**(*n*, *bg_color*, *bg_alpha*, *ix=None*, *fg_color=[0.8941176470588236, 0.10196078431372549, 0.10980392156862745]*, *fg_alpha=1.0*)

Construct the RGBA color parameter for a matplotlib plot.

This function is intended to allow for a set of “foreground” points to be colored according to integer labels (e.g. according to clustering output), while “background” points are all colored something else (e.g. light gray). It is used primarily in the interactive plot tools for DeBaCl but can also be used directly by a user to build a scatterplot from scratch using more complicated DeBaCl output. Note this function can be used to build an RGBA color matrix for any aspect of a plot, including point face color, edge color, and line color, despite use of the term “points” in the descriptions below.

n [int] Number of data points.

bg_color [list of floats] A list with three entries, specifying a color in RGB format.

bg_alpha [float] Specifies background point opacity.

ix [list of ints, optional] Identifies foreground points by index. Default is None, which does not distinguish between foreground and background points.

fg_color [list of ints or list of floats, optional] Only relevant if 'ix' is specified. If 'fg_color' is a list of integers then each entry in 'fg_color' indicates the color of the corresponding foreground point. If 'fg_color' is a list of 3 floats, then all foreground points will be that RGB color. The default is to color all foreground points red.

fg_alpha [float, optional] Opacity of the foreground points.

rgba [2D numpy array] An 'n' x 4 RGBA array, where each row corresponds to a plot point.

`utils.`**`plotForeground`**(*X*, *clusters*, *title=''*, *xlab='x'*, *ylab='y'*, *zlab='z'*, *fg_alpha=0.75*, *bg_alpha=0.3*, *edge_alpha=1.0*, ***kwargs*)

Draw a scatter plot of 2D or 3D data, colored according to foreground cluster label.

X [2-dimensional numpy array] Data points represented by rows. Must have 2 or 3 columns.

clusters [2-dimensional numpy array] A cluster matrix: rows represent points in 'x', with first entry as the index and second entry as the cluster label. The output of all LevelSetTree clustering methods are in this format.

title [string] Axes title

xlab, ylab, zlab [string] Axes axis labels

fg_alpha [float] Transparency of the foreground (clustered) points. A float between 0 (transparent) and 1 (opaque).

bg_alpha [float] Transparency of the background (unclustered) points. A float between 0 (transparent) and 1 (opaque).

kwargs [keyword parameters] Plot parameters passed through to Matplotlib Axes.scatter function.

fig [matplotlib figure] Use fig.show() to show the plot, fig.savefig() to save it, etc.

ax [matplotlib axes object] Allows more direct plot customization in the client function.

`utils.`**`setPlotParams`**(*axes_titlesize=22*, *axes_labelsize=18*, *xtick_labelsize=14*, *ytick_labelsize=14*, *figsize=(9, 9)*, *n_ticklabel=4*)

A handy function for setting matplotlib parameters without adding trival code to working scripts.

axes_titlesize [integer] Size of the axes title.

axes_labelsize [integer] Size of axes dimension labels.

xtick_labelsize [integer] Size of the ticks on the x-axis.

ytick_labelsize [integer] Size of the ticks on the y-axis.

figure_size [tuple (length 2)] Size of the figure in inches.

PYTHON MODULE INDEX

INDEX