SimpSOM Documentation

Release 1.3.0

Federico Comitani

Nov 20, 2017

CONTENTS:

CHAPTER

ONE

INTRODUCTION

SimpSOM is a lightweight implementation of Kohonen Self-Organising Maps (SOM) for Python 2.7, useful for unsupervised learning, clustering and dimensionality reduction.

The package is now available on PyPI, to retrieve it just type `pip install SimpSOM` or download it from here and install with `python setup.py install`.

It allows to build and train SOM on your dataset, save/load the trained network weights, and display or print graphs of the network with selected features. The function `run_colorsExample()` will run a toy model, where a number of colors will be mapped from the 3D RGB space to the 2D network map and clustered according to their similarity in the origin space.

1.1 Dependencies

- Numpy 1.11.0 (older versions may work);
- Matplotlib 1.5.1 (older versions may work);
- Sklearn 0.15 (older versions may work);

1.2 Example of Usage

Here is a quick example on how to use the library with a `raw_data` dataset:

```
#Import the library
import SimpSOM as sps

#Build a network 20x20 with a weights format taken from the raw_data and activate␣
↪Periodic Boundary Conditions.
net = sps.somNet(20, 20, raw_data, PBC=True)

#Train the network for 10000 epochs and with initial learning rate of 0.1.
net.train(0.01, 10000)

#Save the weights to file
net.save('filename_weights')

#Print a map of the network nodes and colour them according to the first feature␣
↪(column number 0) of the dataset
#and then according to the distance between each node and its neighbours.
net.nodes_graph(colnum=0)
net.diff_graph()
```

```
#Project the datapoints on the new 2D network map.
net.project(raw_data, labels=labels)

#Cluster the datapoints according to the Quality Threshold algorithm.
net.cluster(raw_data, type='qthresh')
```

CHAPTER

TWO

SIMPSOM

2.1 SimpSOM package

2.1.1 Submodules

2.1.2 SimpSOM.densityPeak module

Density Peak Clustering

A Rodriguez, A Laio, Clustering by fast search and find of density peaks SCIENCE, 1492, vol 322 (2014)

6. Comitani @2017

class SimpSOM.densityPeak.**collection**(*coorArray*, *typeFunc='gaussian'*, *percent=0.02*, *PBC=False*, *netHeight=0*, *netWidth=0*)

Class for a collection of point objects.

cluster_assign()

Assign a cluster to each point according according to its nearest neighbour with higher density.

core_assign()

Assign points as belonging to the core or the halo of a cluster.

decision_graph(*show=False*, *printout=True*)

Calculate the decision graph, delta vs rho for the points belonging to the collection and find the cluster centers.

Args: show (bool, optional): Choose to display the plot. printout (bool, optional): Choose to save the plot to a file.

get_clusterList()

Returns the indeces of the clustered points as a list.

Returns: clusters (list, int): a list of lists containing the points indices belonging to each cluster

refd = None

Make sure rhos are set before setting deltas

set_deltas()

Calculate the distance from higher density points for each point in the dataset.

set_dists()

Calculate the distance matrix for all points.

set_rhos(*typeFunc='step'*)

Calculate the density for each point in the dataset.

Args: typeFunc (str): step function type (step, gaussian or logistic)

SimpSOM.densityPeak.**densityPeak**(*sample*, *show=False*, *printout=False*, *percent=0.02*, *PBC=False*, *netHeight=0*, *netWidth=0*)
Run the complete clustering algorithm in one go and returns the clustered indeces as a list.

Args: sample (array): The input dataset show (bool, optional): Choose to display the decision graph. printout (bool, optional): Choose to save the decision graph to a file.

Returns: clusters (list, int): a list of lists containing the points indices belonging to each cluster

SimpSOM.densityPeak.**dist**(*p1*, *p2*, *metric='euclid'*, *PBC=False*, *netHeight=0*, *netWidth=0*)
Calculate the distance between two point objects in a N dimensional space according to a given metric.

Args: p1 (point): First point object for the distance. p2 (point): Second point object for the distance. metric (string): Metric to use. For now only euclidean distance is implemented. PBC (bool, optional): Activate/deactivate Periodic Boundary Conditions. netHeight (int, optional): Number of nodes along the first dimension, required for PBC. netWidth (int, optional): Numer of nodes along the second dimension, required for PBC.

Returns: (float): The distance between the two points.

SimpSOM.densityPeak.**gaussian**(*p1*, *p2*, *sigma*, *PBC=False*, *netHeight=0*, *netWidth=0*)
Gaussian function of the distance between two points scaled with sigma.

Args: p1 (point): First point object for the distance. p2 (point): Second point object for the distance. sigma (float): The scaling factor for the distance. PBC (bool, optional): Activate/deactivate Periodic Boundary Conditions. netHeight (int, optional): Number of nodes along the first dimension, required for PBC. netWidth (int, optional): Numer of nodes along the second dimension, required for PBC.

Returns: (float): value of the gaussian function.

class SimpSOM.densityPeak.**pt**(*coordinates*)
Class for the points to cluster.

set_delta(*coll*)
Calculate the distance of the point from higher density points and set the nearest neighbour. [Deprecated]

Args: coll (collection): collection containing all the points of the dataset used to calculate the distance.

set_dist(*coll*)
Calculate the distances from all other points in a collection. [Deprecated]

Args: coll (collection): collection containing all the points of the dataset used to calculate the distances.

set_rho(*coll*, *typeFunc='step'*)
Calculate the density of the single point for a given dataset. [Deprecated]

Args: coll (collection): collection containing all the points of the dataset used to calculate the density. typeFunc (str): step function type (step, gaussian kernel or logistic).

SimpSOM.densityPeak.**sigmoid**(*p1*, *p2*, *sigma*, *PBC=False*, *netHeight=0*, *netWidth=0*)
Logistic function of the distance between two points scaled with sigma.

Args: p1 (point): First point object for the distance. p2 (point): Second point object for the distance. sigma (float): The scaling factor for the distance. PBC (bool, optional): Activate/deactivate Periodic Boundary Conditions. netHeight (int, optional): Number of nodes along the first dimension, required for PBC. netWidth (int, optional): Numer of nodes along the second dimension, required for PBC.

Returns: (float): value of the logistic function.

SimpSOM.densityPeak.**step**(*p1*, *p2*, *cutoff*, *PBC=False*, *netHeight=0*, *netWidth=0*)
Step function activated when the distance of two points is less than the cutoff.

Args: p1 (point): First point object for the distance. p2 (point): Second point object for the distance. cutoff (float): The cutoff to define the proximity of the points. PBC (bool, optional): Activate/deactivate Periodic

Boundary Conditions. netHeight (int, optional): Number of nodes along the first dimension, required for PBC. netWidth (int, optional): Numer of nodes along the second dimension, required for PBC.

Returns: (int): 1 if the points are closer than the cutoff, 0 otherwise.

SimpSOM.densityPeak.**test**()
Run the complete clustering algorithm on a test case and print the clustered points graph.

2.1.3 SimpSOM.hexagons module

Hexagonal tiling library

6. Comitani @2017

SimpSOM.hexagons.**coorToHex**(*x*, *y*)
Convert Cartesian coordinates to hexagonal tiling coordinates.

Args: x (float): position along the x-axis of Cartesian coordinates. y (float): position along the y-axis of Cartesian coordinates.

Returns: array: a 2d array containing the coordinates in the new space.

SimpSOM.hexagons.**plot_hex**(*fig*, *centers*, *weights*)
Plot an hexagonal grid based on the nodes positions and color the tiles according to their weights.

Args: fig (matplotlib figure object): the figure on which the hexagonal grid will be plotted. centers (list, float): array containing couples of coordinates for each cell

to be plotted in the Hexagonal tiling space.

weights (list, float): array contaning informations on the weigths of each cell, to be plotted as colors.

Returns: ax (matplotlib axis object): the axis on which the hexagonal grid has been plotted.

2.1.4 SimpSOM.qualityThreshold module

Quality Threshold Clustering

L. J. Heyer, S. Kruglyak and S. Yooseph, Exploring Expression Data: Identification and Analysis of Coex-pressed Genes Genome Research, Vol. 9, No. 11, 1999, pp. 1106-1115.

6. Comitani @2017

SimpSOM.qualityThreshold.**qualityThreshold**(*sample*, *cutoff=5*, *PBC=False*, *netHeight=0*, *netWidth=0*)
Run the complete clustering algorithm in one go and returns the clustered indeces as a list.

Args: sample (array): The input dataset cutoff (float, optional): The clustering cutoff distance. PBC (bool, optional): Activate/deactivate Periodic Boundary Conditions. netHeight (int, optional): Number of nodes along the first dimension, required for PBC. netWidth (int, optional): Numer of nodes along the second dimension, required for PBC.

Returns: clusters (list, int): a list of lists containing the points indices belonging to each cluster

SimpSOM.qualityThreshold.**test**()

2.1.5 Module contents

SimpSOM (Simple Self-Organizing Maps) v1.3.0 F. Comitani @2017

A lightweight python library for Kohonen Self-Organising Maps (SOM).

SimpSOM.**run_colorsExample**()
Example of usage of SimpSOM: a number of vectors of length three (corresponding to the RGB values of a color) are used to briefly train a small network. Different example graphs are then printed from the trained network.

class SimpSOM.**somNet**(*netHeight*, *netWidth*, *data*, *loadFile=None*, *PCI=0*, *PBC=0*)
Kohonen SOM Network class.

PCI = None
Switch to activate periodic boundary conditions.

cluster(*array*, *type='qthresh'*, *cutoff=5*, *quant=0.2*, *percent=0.02*, *numcl=8*, *savefile=True*, *filetype='dat'*, *show=False*, *printout=True*)

Clusters the data in a given array according to the SOM trained map. The clusters can also be plotted.

Args: array (np.array): An array containing datapoints to be clustered. type (str, optional): The type of clustering to be applied, so far only quality threshold (qthresh)

algorithm is directly implemented, other algorithms require sklearn.

cutoff (float, optional): Cutoff for the quality threshold algorithm. This also doubles as maximum distance of two points to be considered in the same cluster with DBSCAN.

percent (float, optional): The percentile that defines the reference distance in density peak clustering (dpeak). numcl (int, optional): The number of clusters for K-Means clustering quant (float, optional): Quantile used to calculate the bandwidth of the mean shift algorithm. savefile (bool, optional): Choose to save the resulting clusters in a text file. filetype (string, optional): Format of the file where the clusters will be saved (csv or dat) show (bool, optional): Choose to display the plot. printout (bool, optional): Choose to save the plot to a file.

Returns: (list of int): A nested list containing the clusters with indexes of the input array points.

colorEx = None
Switch to activate periodic PCA weights initialisation.

data = None
Load the weights from file, generate them randomly or from PCA.

diff_graph(*show=False*, *printout=True*)
Plot a 2D map with nodes and weights difference among neighbouring nodes.

Args: show (bool, optional): Choose to display the plot. printout (bool, optional): Choose to save the plot to a file.

find_bmu(*vec*)
Find the best matching unit (BMU) for a given vector.

Args: vec (np.array): The vector to match.

Returns: bmu (somNode): The best matching unit node.

nodes_graph(*colnum=0*, *show=False*, *printout=True*)
Plot a 2D map with hexagonal nodes and weights values

Args: colnum (int): The index of the weight that will be shown as colormap. show (bool, optional): Choose to display the plot. printout (bool, optional): Choose to save the plot to a file.

project (*array*, *colnum=-1*, *labels=[]*, *show=False*, *printout=True*)

Project the datapoints of a given array to the 2D space of the SOM by calculating the bmus. If requested plot a 2D map with as implemented in nodes_graph and adds circles to the bmu of each datapoint in a given array.

Args: array (np.array): An array containing datapoints to be mapped. colnum (int): The index of the weight that will be shown as colormap.

If not chosen, the difference map will be used instead.

show (bool, optional): Choose to display the plot. printout (bool, optional): Choose to save the plot to a file.

Returns: (list): bmu x,y position for each input array datapoint.

save (*fileName='somNet_trained'*)
Saves the network dimensions, the pbc and nodes weights to a file.

Args: fileName (str, optional): Name of file where the data will be saved.

train (*startLearnRate=0.01*, *epochs=-1*)
Train the SOM.

Args: startLearnRate (float): Initial learning rate. epochs (int): Number of training iterations. If not selected (or -1)

automatically set epochs as 10 times the number of datapoints

update_lrate (*iter*)
Update the learning rate.

Args: iter (int): Iteration number.

update_sigma (*iter*)
Update the gaussian sigma.

Args: iter (int): Iteration number.

class SimpSOM.**somNode** (*x*, *y*, *numWeights*, *netHeight*, *netWidth*, *PBC*, *minVal=[]*, *maxVal=[]*, *pcaVec=[]*, *weiArray=[]*)
Single Kohonen SOM Node class.

get_distance (*vec*)
Calculate the distance between the weights vector of the node and a given vector.

Args: vec (np.array): The vector from which the distance is calculated.

Returns: (float): The distance between the two weight vectors.

get_nodeDistance (*node*)
Calculate the distance within the network between the node and another node.

Args: node (somNode): The node from which the distance is calculated.

Returns: (float): The distance between the two nodes.

update_weights (*inputVec*, *sigma*, *lrate*, *bmu*)
Update the node Weights.

Args: inputVec (np.array): A weights vector whose distance drives the direction of the update. sigma (float): The updated gaussian sigma. lrate (float): The updated learning rate. bmu (somNode): The best matching unit.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S