Moduler Documentation

Release 1.0

Jose Sergio Hleap

July 24, 2015

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

Graph based Modularity. This script will evaluate the data for modules. Such modules are defined as correlating variables, so the clustering is performed in the correlation space. It has an optional statistical significance test for the clustering and power analysis of the result, as well as a bootstrap analysis. See options for more details.

E-mail: jshleap@dal.ca

In this version the the R dependencies have been extracted, and with them the RV coefficient test.

1.1 Requirements:

1. numpy module
2. scipy module
3. statsmodels module
4. matplotlib
5. scikit-learn
6. PDBnet module: This is an open source module and can be found in :download: *LabBlouin-Tools<https://github.com/LabBlouin/LabBlouinTools>*

   To install python modules 1-5 in UBUNTU: sudo apt-get install python-<module> OR sudo easy_install -U <module> OR sudo pip install -U <module>

   For PDBnet, the whole directory should be downloded into a labblouin folder which should be in your pythonpath

CHAPTER

TWO

TUTORIAL

Aqui va el turorial que no he realizado todavia

CHAPTER

THREE

DOCUMENTATION

class `Moduler.`**`GMdata`**(*prefix*, *dimension=3*, *t='gm'*, *asdistance=False*, *contacts=False*)

GM object that populates GM data

Parameters

- **`prefix`** (*string*) – Prefix of your GM file and accompanying files
- **`dimension`** (*integer*) – The number of cartesian dimensions to be analyzed. By default is set to 3.
- **`t`** (*string*) – Type of input. Either gm or csv, both semicolon delimited files, but the former includes names of the observations in the first field. By default is gm
- **`asdistance`** (*boolean*) – whether or not to tranform the data into a distance matrix

`Correlated`(*GMstatsInstance*)

Include a GMstats instance

Parameters **`GMstatsInstance`** (*:class GMstats*) – An instance of the class GMstats

`Load_contacts`()

Load the contacts from file, from PDBstructure or None

`Read_GM`()

Load data from a gm (coordinates file) file. The file is a semicolon separated file with row names in the first field.

`bootstrap_replicate`()

Create a bootstrap replicate of the data

class `Moduler.`**`GMstats`**(*prefix*, *matrix*, *dimensions*, *sample_size=None*)

Include all stats related things with a GM file

Parameters

- **`prefix`** (*string*) – Prefix of your GM file and accompanying files
- **`matrix`** (:class *numpy.array*) – A numpy nd array with the coordinates or info to be analysed. It contains the dimensions as first element, rows and columns follow.
- **`dimensions`** (*integer*) – The number of cartesian dimensions to be analyzed. By default is set to 3.
- **`sample_size`** (*integer*) – Number of observations

`Clus_Power`(*membership*)

Make a power analysis of the clusters by igraph and outputs a table with the proportion of elements above the n required according to significance, power and correlation. it can test different clusters that are not in the class

Parameters **`m`** (*list or string*) – a membership vector in list form or the name of a file.

`Compute_correlations`(*method='fisher'*, *absolutecov=False*, *confval=0.95*, *threshold=0.0*, *usecov=False*, *writemat=False*, *additive=True*, *power=0.8*)

Parameters

- **`method`** (*string*) – Which method to use for correlation. By default it uses the fisher transformation. Available options are pearson, spearman and fisher.
- **`confval`** (*float*) – Define the confidence level (1-alpha) for the correlation test. By default is 0.95. Can take any float value between 0 and 1.
- **`absolutecov`** (*boolean*) – Whether or not to use absolute values of the correlation matrix
- **`power`** (*float*) – Perform a statistical power analysis with this value as the desired Power (1-type II error). By default 0.8 is used. Can be any float from 0 to 1
- **`usecov`** (*boolean*) – Use covariance instead of correlation.
- **`writemat`** – Write correlation/covariance matrices to file. By default is false. It can take a False, for the aggreatated matrix or cor for the correlation matrix.
- **`additive`** (*boolean*) – Use the mean of the correlation in each dimension instead of the euclidean distance to aglomerate the dimensions. The default behaviour is additive.

`F_transf`(*r*)
Compute the Fisher transformation of correlation

`GetAglomeratedThreshold`()
Get threshold for the n dimensions

`LDA`(*membership*, *which='lms'*, *ellipses=2.5*)
Perform a Linear discriminant analysis of the transposed data. Membership must be an array of integers of the same lenght of the number of observations in the data.

Parameters

- **`membership`** (*list*) – a list corresponding to memberships of the entries in data
- **`which`** (*string*) – Either 'gm' or 'lms'. To perform the LDA in the full matrix or only in the correlation matrix.
- **`ellipses`** (*float*) – A value representing the estandard deviations for confidence ellipses. By default is set to 2.5 (95% confidence ellipses)

`Power_r`(*corr*)
Compute the power of the correlation using the Z' trasnformation of correlation coefficient: Z'=arctang(r)+r/(2*(n-1)) (see Cohen (1988) p.546). It will return the required n fo the power and significance chosen.

Parameters **`corr`** (*float*) – Correlation value

`SigCorrOneMatrix`(*sliced*)
Performs the significance of correlation test according to the method passed

Parameters **`sliced`** (:class *numpy.array*) – an array with single dimensional data

`Sigcorr`()
Test if the correlation is significantly different than 0 with the method specified

`UseCorr`()
Use pearson correlation without a significant test

`UseCov`()
: Create a variance-covariance matrix

`Z_fisher`()
: Compute the sample - corrected Z_alpha for hypotesis testing of Fisher transformation of correlation

`agglomerare_additive`()
: Agglomerate landmark dimensions using euclidean distance

`agglomerare_mean`()
: Agglomerate landmark dimensions using average of correlation

class `Moduler.`**`GMgraph`**(*prefix*, *Matrix*, *unweighted=False*, *gfilter=[]*, *threshold=0.0*)
: Create a graph based on an input in matrix form and compute modularity

Parameters

- **`Matrix`** (:class *numpy.array* or :class *GMstats*) – an square matrix where the indices represent intended nodes and the values the relationship between them
- **`unweighted`** (*boolean*) – create an unweighted graph as opposed to a weighted (correlation; default) one.
- **`gfilter`** (*list of tuples*) – List of tuples corresponding to desired conection of the graph. Each element in pair tuple must correspond to node indices in the graph. This is a topology constraint.
- **`threshold`** (*float*) – A float corresponding to the threshold to create a conection between nodes. This is very user dependendent and is set to 0 as default.

`Build_igraph`()
: Build a graph, using igraph library. It will return it, and store it as an attribute (self.g)

Returns :class:: 'igraph.Graph'

`Cluster2File`()
: Write cluster to a file and rename the cluster with PDB friendly characters if possible (this is specific use)

`Get_StructProps`(*overall=False*)
: Get the structural properties in the graph

Parameters **`overall`** (*boolean*) – Calculate the centralities on the full graph, as opposed as by modules (Default behaviour).

`Graph_Cluster`(*method='fastgreedy'*, ***kwargs*)
: Clustering by components comstGreed, using igraph library.

Parameters

- **`method`** (*string.*) – method in igraph ofr community detection. It can be fastgreedy, infomap leading_eigenvector_naive,leading_eigenvector,label_propagation, multilevel, optimal_modularity, edge_betweenness, spinglass, walktrap. For details see igraph documentation.
- **`kwargs`** – other arguments passed to the igraph methods

`Identify_Singletons`(*method='fastgreedy'*)
: Given a membership vector identify singletons

`LDAmerge`(*which='lms'*, *ellipses=2.5*, *dimensions=1*)
: Perform an LDA analisis and merge all classes which 95% confidence ellipses collide.

Parameters

- **`which`** (*string*) – Either 'gm' or 'lms'. To perform the LDA in the full matrix or only in the correlation matrix.
- **`ellipses`** (*float*) – A value representing the estandard deviations for confidence ellipses. By default is set to 2.5 (95% confidence ellipses)
- **`dimensions`** (*integer*) – dimensions of the matrix

class `Moduler.`**`SupportClustering`**(*prefix*, *data*, *membership*, *dimensions*, *permutations*, *confval=0.95*, *threshold=0.0*)

This class provides ways to provide statistical support for a given clustering scheme. It is based in testing if the correlation between groups is significatly different than between groups.

Parameters

- **`prefix`** (*string*) – a prefix for the output.
- **`data`** (:class *numpy.ndarray*) – a 2D numpy array with the data from which the clustering was inferred.
- **`membership`** (*list*) – a list equal to the second dimension of data (i.e. data.shape[1]), corresponding to the clustering scheme being tested.
- **`dimensions`** (*integer*) – Number of dimensions in your data matrix. If your matrix is correlation or related, dimesions should be one.
- **`permutations`** (*integer*) – Number of permutations to perform the permutation t-test.
- **`confval`** (*float*) – confidence value for the test (1 - alpha).
- **`threshold`** (*float or None*) – Value to filter out values of the correlation. If set to none, no threshold filtering will be done.

`AreModneighbours`(*A*, *indA*, *indB*)

loop over the adjacency matrix to find if indA and indB are in the neiborhood

Parameters

- **`A`** (*list*) – a list of tupples of related entries
- **`indA`** (*list*) – indices of grup A
- **`indB`** (*list*) – indices of grup B

Returns boolean of whether or not indA and indB are neighbours

`BipartitionAgree`(*a*, *b*)

Return whether 2 strings of bipartitions agree or conflict. The strings must consist of 1 and 0 only

Params a,b binary bipartition strings to be compared for agreements.

`DealDimensions`()

If the data has more than one dimension (in the cartesian sense or the origin of the data), split it, compute correlation of each dimension and then aggregate it using euclidean distance of the coefficients of determination. It assumes that the dimensions are interleaved. This correlation does not have a significance testing, use caution. It is reccomended to use GMstats class before using this class.

`FDR_correction`()

Compute the False Discovery Rate correction for the critical value

`FDRc_sigtest`()

Perform the logical significance test using FDR corrected critical value. Returns a binary dictionary of the comparisons

VectorToEdgeList(*v*)

Convert a membership vector to a list of edges. The membership vector must be a list or space separated string.

Parameters **v** (*list*) – a membership vector in list form

WriteBoot()

Write Bootstrap results to screen and file

bipartition_agreement(*prefix*)

Calculate the local bipartition agreement scores

bootstrap(*boot=0*, *contacts=[]*, *unweighted=False*, *graphmethod='fastgreedy'*, *lda=False*, *iterative=True*, ***kwargs*)

Execute the bootstrap inferfence. This boostrap resample obsevations (rows) in the data

Parameters

- **boot** (*integer*) – Number of bootstrap replicates to be performed
- **contacts** (*list*) – filter out non-contact interactions. Contacts passed as list of tuples
- **unweighted** (*boolean*) – create an unweighted graph as opposed to a weighted (correlation; default) one.
- **graphmethod** – method in igraph ofr community detection. It can be fastgreedy, infomap leading_eigenvector_naive,leading_eigenvector,label_propagation, multilevel, optimal_modularity, edge_betweenness, spinglass, walktrap. For details see igraph documentation.
- **lda** (*boolean*) – Use LDA to premerge the community detection clusters
- **kwargs** – parameter and arguments for GMstats class in the method Compute_correlations

filter_singletons(*D*)

Giving a dictionary with cluster indices, return whichones are singletons

Parameters **D** (*dictionary*) – a dictionary with cluster indices

Returns a dictionary with the indices of unconnected nodes

get_cluster_indices()

Returns a dictionary with cluster indices

if_bootfile(*boot=100*)

if another bootsrap intance has been called and crashed, this function will finished the remaining and / or compute the agreement

Parameters **boot** (*integer*) – Number of bootstrap replicates to be performed

iterative_permt(*filterin*)

Perform a permutation test iteratively, until a stable membership vector is reached. In each iteration a permutation test is performed between the clusters, and a merge event will happen if no evidence of difference is found.

Parameters **filterin** (*list*) – a list of tuples of pairs of variable to be inlcuded (i.e. adjacency list if mem was provided by a graph)

permt(*filterin*, *count=''*)

Perform the permutation test over all pairs of classes in the membership vector provided

Parameters

- **filterin** (*list*) – a list of tuples of pairs of variable to be inlcuded (i.e. adjacency list if mem was provided by a graph)
- **count** (*string or integer*) – Step in which this function has been called. Is for Iterative usage.

subsetTest (*k1*, *k2*)
Given a series of indices, split the data into intra and inter correlation and test for equality

Parameters

- **subsetdata** –
- **k1** (*string or integer*) – Name of one of the clusters being compared
- **k2** (*string or integer*) – Name of the other clusters being compared

twotPermutation (*x1*, *x2*)
This function computes the p-value for the two sample t-test using a permutation test. This is a translation from the DAAG Package function with the same name. :param x1: First array (sample) to be compared :type x1: :class *numpy.ndarray* :param x2: Second array (sample) to be compared :type x2: :class *numpy.ndarray* :param nsim: number of simulations to run ro compute the ovalue :type nsim: integer

write_permt (*newm*, *count=''*)
Write and print the output of the permutation test, the new membership vector, and do some cleanup

3.1 Auxiliary methods:

Moduler.**isFloat** (*string*)
Auxiliary function to test if a string is a float

Parameters **string** (*str*) – The string to be tested

Moduler.**rename_clusters** (*newcl*, *mem*)
Rename clusters using the smallest label in each set. Returns the new membership vector

Parameters

- **newcl** (*list*) – a list with the new membership before renaming
- **mem** (*list*) – original membership vector

Returns A list with the renamed vector

Moduler.**equiclus** (*merge*)
Look for equivalent clusters, and return a list with sets of equivalent clusters

Parameters **merge** (*list*) – list of tuples with the pairs of classes to be merged

Returns cleaned membership vector as a list

Moduler.**ellipse** (*singlegroupx*, *singlegroupy*, *std=2.5*)
Create an ellipse given points with x coordinates in singlegroupx and singlegroupy

Parameters

- **singlegroupx** (:class *numpy.array*) – An array with the x coordinates of data for what the ellipse is to be built.
- **singlegroupy** (:class *numpy.array*) – An array with the y coordinates of data for what the ellipse is to be built.

- **`std`** (*float*) – The standard deviation of the confidence ellipse. By default is set to 2.5 (95% confidence ellipse)

Returns a :class *matplotlib.patches.Ellipse* object

`Moduler.`**`pointsInEllipse`**(*Xs*, *Ys*, *ellipse*)

Tests which set of points are within the boundaries defined by the ellipse. The set of points are defined in two arrays Xs and Ys for the x-coordinates and y-coordinates respectively.

Params Xs x-coordinates of a set of points

Params Ys y-coordinates of a set of points

Parameters **`ellipse`** (:class *matplotlib.patches.Ellipse*) – instance of method ellipse

Returns Tuple of two list, points inside and outside of the ellipse

`Moduler.`**`getEllipses`**(*fit*, *membership*, *std=2.5*)

Populate the ellipses dictionary with as many ellipses as components in membership.

Parameters

- **`fit`** (:class *numpy.array*) – An array normally computed by LDA or other method
- **`std`** (*float*) – The standard deviation of the confidence ellipse. By default is set to 2.5 (95% confidence ellipse)
- **`membership`** (*list*) – A list corresponding to memberships of the entries in data.

Returns A dictionary with Ellipse objects with membership as keys.

`Moduler.`**`EllipseOverlap`**(*ellipse1*, *ellipse2*)

Find collisions between ellipses

Parameters

- **`ellipse1`** (:class *matplotlib.patches.Ellipse*) – Matplotlib ellipse
- **`ellipse2`** (:class *matplotlib.patches.Ellipse*) – Matplotlib ellipse

Returns Boolean of whether or not two ellipses collide

`Moduler.`**`Specificity`**(*ref*, *test*)

Compute specificity from two edgelists (list of edges between nodes/labes)

Parameters

- **`ref`** (*list*) – Reference edgelist
- **`test`** (*list*) – Test edgelist

`Moduler.`**`Sensitivity`**(*ref*, *test*)

Compute sensitivity from two edgelists (list of edges between nodes/labes)

Parameters

- **`ref`** (*list*) – Reference edgelist
- **`test`** (*list*) – Test edgelist

`Moduler.`**`Fscore`**(*sp*, *sn*)

Compute the F-score

Parameters

- **`sp`** (*float*) – Specificity value
- **`sn`** (*float*) – Sensitivity value

`Moduler.`**`clus2list`**(*membership*, *data*)

Giving a membership vector and data, split the data into cluster specific arrays.

Parameters **`membership`** (*list or string*) – a membership vector

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX