multivariate$_{i}nferenceDocumentation$

Release 0.1.0

Eshin Jolly

Sep 19, 2018

Contents

Package to test and compare different multivariate inference methods.

CHAPTER 1

Installation

```
pip install git+https://github.com/ejolly/multivariate_inference
```

1.1 Usage

To use multivariate_inference in a project:

```
import multivariate_inference
```

1.2 API Reference

This reference provides detailed documentation for all the features in multivariate_inference

1.2.1 `multivariate_inference.helpers`: General Helper Functions

Helper function definitions.

`multivariate_inference.helpers.`**`upper`**(*mat*)
Return upper triangle of matrix

`multivariate_inference.helpers.`**`isPSD`**(*mat*, *tol=1e-08*)
Check if matrix is positive-semi-definite by virtue of all its eigenvalues being >= 0. The cholesky decomposition does not work for edge cases because np.linalg.cholesky fails on matrices with exactly 0 valued eigenvalues, whereas in Matlab this is not true, so that method appropriate. Ref: https://goo.gl/qKWWzJ

`multivariate_inference.helpers.`**`nearestPSD`**(*A*, *nit=100*)
Higham (2000) algorithm to find the nearest positive semi-definite matrix that minimizes the Frobenius distance/norm. Sstatsmodels using something very similar in corr_nearest(), but with spectral SGD to search for a local minima. Reference: https://goo.gl/Eut7UU

Parameters **nit** (*int*) – number of iterations to run algorithm; more iterations improves accuracy but increases computation time.

multivariate_inference.helpers.**easy_multivariate_normal**(*num_obs*, *num_features*, *corrs*, *mu=0.0*, *sigma=1.0*, *seed=None*, *forcePSD=True*, *return_new_corrs=False*, *nit=100*)

Function to more easily generate multivariate normal samples provided a correlation matrix or list of correlations (upper triangle of correlation matrix) instead of a covariance matrix. Defaults to returning approximately standard normal (mu = 0; sigma = 1) variates. Unlike numpy, if the desired correlation matrix is not positive-semi-definite, will by default issue a warning and find the nearest PSD correlation matrix and generate data with this matrix. This new matrix can optionally be returned used the return_new_corrs argument.

Parameters

- **num_obs** (*int*) – number of observations/samples to generate (rows)
- **corrs** (*ndarray/list/float*) – num_features x num_features 2d array, flattend numpy array of length (num_features * (num_features-1)) / 2, or scalar for same correlation on all off-diagonals
- **num_features** (*int*) – number of features/variables/dimensions to generate (columns)
- **mu** (*float/list*) – mean of each feature across observations; default 0.0
- **sigma** (*float/list*) – sd of each feature across observations; default 1.0
- **forcePD** (*bool*) – whether to find and use a new correlation matrix if the requested one is not positive semi-definite; default False
- **return_new_corrs** (*bool*) – return the nearest correlation matrix that is positive semi-definite used to generate data; default False
- **nit** (*int*) – number of iterations to search for the nearest positive-semi-definite correlation matrix is the requested correlation matrix is not PSD; default 100

Returns correlated data as num_obs x num_features array

Return type ndarray

multivariate_inference.helpers.**kde_pvalue**(*permutation_distribution*, *test_statistic*, *tails=2*, *kde_grid_size=200*)

Use a KDE to smooth a permutation distribution and use a interpolation to compute p-values a la: https://users.aalto.fi/~eglerean/bramila_mantel.m

Parameters

- **permutation_distribution** (*ndarry*) – array of permuted test statistics
- **test_statistic** (*float*) – true value of computed test statistic
- **tails** (*int*) – two-tailed or one-tailed p-value; default two-tailed
- **kde_grid_size** (*int*) – size of the kde grid to generate; default 200 if len(permutation_distribution) <= 5000 otherwise multiples of 200 correponding to how many extra permutations were performed in multiples of 5000

multivariate_inference.helpers.**create_heterogeneous_simulation**(*r_within_1*, *r_within_2*, *r_between_1*, *r_between_2*, *n_variables*)

Create a heterogeneous multivariate covariance matrix based on: Omelka, M. and Hudecova, S. (2013) A comparison of the Mantel test with a generalised distance covariance test. Environmetrics, Vol. 24, 449–460. DOI: 10.1002/env.2238.

1.2.2 multivariate_inference.dependence: Multivariate Dependence Measures

Dependence measures functions.

multivariate_inference.dependence.**double_center**(*mat*)
Double center a 2d array.

Parameters **mat** (*ndarray*) – 2d numpy array

Returns double-centered version of input

Return type mat (ndarray)

multivariate_inference.dependence.**u_center**(*mat*)
U-center a 2d array. U-centering is a bias-corrected form of double-centering

Parameters **mat** (*ndarray*) – 2d numpy array

Returns u-centered version of input

Return type mat (narray)

multivariate_inference.dependence.**distance_correlation**(*x*, *y*, *bias_corrected=True*, *return_all_stats=False*)

Compute the distance correlation betwen 2 arrays. Distance correlation involves computing the normalized covariance of two centered euclidean distance matrices. Each distance matrix is the euclidean distance between rows (if x or y are 2d) or scalars (if x or y are 1d). Each matrix is centered using u-centering, a bias-corrected form of double-centering. This permits inference of the normalized covariance between each distance matrix using a one-tailed directional t-test. (Szekely & Rizzo, 2013). While distance correlation is normally bounded between 0 and 1, u-centering can produce negative estimates, which are never significant. Therefore these estimates are windsorized to 0, ala Geerligs, Cam-CAN, Henson, 2016.

Parameters

- **x** (*ndarray*) – 1d or 2d numpy array of observations by features
- **y** (*ndarry*) – 1d or 2d numpy array of observations by features
- **bias_corrected** (*bool*) – if false use double-centering but no inference test is performed, if true use u-centering and perform inference; default True
- **return_all_stats** (*bool*) – if true return distance covariance and variances of each array as well; default False

Returns dictionary of results (correlation, t, p, and df.) Optionally, covariance, x variance, and y variance

Return type results (dict)

multivariate_inference.dependence.**procrustes_similarity**(*mat1*, *mat2*, *n_permute=5000*, *tail=1*, *n_jobs=-1*, *random_state=None*)
Use procrustes super-position to perform a similarity test between 2 matrices. Matrices need to match in size on their first dimension only, as the smaller matrix on the second dimension will be padded with zeros. After

aligning two matrices using the procrustes transformation, use the computed disparity between them (sum of squared error of elements) as a similarity metric. Shuffle the rows of one of the matrices and recompute the disparity to perform inference (Peres-Neto & Jackson, 2001). Note: by default this function reverses disparity to treat it like a *similarity* measure like correlation, rather than a distance measure like correlation distance, i.e. smaller values mean less similar, larger values mean more similar.

Parameters

- **mat1** (*ndarray*) – 2d numpy array; must have same number of rows as mat2
- **mat2** (*ndarray*) – 1d or 2d numpy array; must have same number of rows as mat1
- **n_permute** (*int*) – number of permutation iterations to perform
- **tail** (*int*) – either 1 for one-tailed or 2 for two-tailed test; default 2
- **n_jobs** (*int*) – The number of CPUs to use to do permutation; default -1 (all)

Returns similarity between matrices bounded between 0 and 1 pval (float): permuted p-value

Return type similarity (float)

CHAPTER 2

Usage

CHAPTER 3

API

Python Module Index

m

Index