si4ul

Release 0.1.0

omori.y

Feb 22, 2022

CONTENTS:

CHAPTER

ONE

SI4UL.CHANGEPOINT_SI MODULE

si4ul.changepoint_si.**optseg_si**(*x*, *sigma=1*, *beta=1.5*)

computing valid p-values for optimal changepoint by Selective Inference using Dynamic Programming.

Parameters

- **x** (*array-like of shape (point-num)*) – time series data.
- **sigma** (*float*) – standard deviation in distribution of time series data.
- **beta** (*float*) – regularization factor.

Returns changepoint list and p-values of each changepoint.

Return type (array-like of shape (changepoint-num), array-like of shape (changepoint-num))

Examples

```
>>> seg, p_list = changepoint_si.optseg_si(x)
>>> print(seg)
[25, 40, 59, 80, 100, 125, 140, 159, 180]
>>> print(p_list)
[6.439051311970013e-06, 5.451462826100586e-07, 3.589542368255502e-07, 7.
↪74132481356728e-05, 0.11719543493338598, 6.4387982662212394e-06, 1.
↪1776732917248327e-06, 3.589542368255725e-07, 1.8180641069769048e-09]
```

si4ul.changepoint_si.**optseg_si_oc**(*x*, *sigma=1*, *beta=1.5*)

computing valid p-values for optimal changepoint by over-comditioning Selective Inference.

Parameters

- **x** (*array-like of shape (point-num)*) – time series data.
- **sigma** (*float*) – standard deviation in distribution of time series data.
- **beta** (*float*) – regularization factor.

Returns changepoint list and p-values of each changepoint.

Return type (array-like of shape (changepoint-num), array-like of shape (changepoint-num))

Examples

```
>>> seg, p_list = changepoint_si.optseg_si(x)
>>> print(seg)
[25, 40, 59, 75, 80, 100, 125, 140, 159, 175, 180]
>>> print(p_list)
[0.013528694810211626, 0.05567852225768205, 0.329173206633249, 0.360141111520463954,␣
↪0.4101971284236369, 0.6981932657047393, 0.16996861755494708, 0.3719537702838958,␣
↪0.3592986205789473, 0.360141111520464676, 0.1196678816881915]
```

si4ul.changepoint_si.**plot_changepoint_detection**(*x*, *sg_results*, *p_value_list*, *alpha*, *underlying=None*, *segment_size=0*, *title='OptSeg-SI'*)

the result of optimal changepoint after SI.

Parameters

- **x** (*array-like of shape (point-num)*) – time series data.
- **sg_results** (*array-like of shape (changepoint-num)*) – changepoint list.
- **p_value_list** (*array-like of shape (changepoint-num)*) – p-values of each changepoint.
- **alpha** (*float*) – significance level.
- **underlying** (*array-like of shape (true changepoint-num)*) – mean values of true segment.
- **segment_size** (*float*) – true segment size.
- **title** (*string*) – plot title.

CHAPTER

TWO

SI4UL.MULTISEQ_CP_SI MODULE

si4ul.multiseq_cp_si.**multiseq_cp_dc_si**(*X*, *K*, *L*, *Xi=1*, *Sigma=1*, *test='hom'*, *width=0*, *phi=0.5*)
computing valid p-values for changepoint with Double-CUSUM by Selective Inference.

Parameters

- **X** (*ndarray of shape (component, location)*) – multi-dimensional sequence.
- **K** (*int*) – the number of change locations.
- **L** (*int*) – minimum segment length.
- **Xi** (*float or ndarray, optional*) – component's variance. Defaults to 1.
- **Sigma** (*float or ndarray, optional*) – location's variance. Defaults to 1.
- **test** (*str, optional*) – Set 'hom' for homotopy search, 'oc' for over-conditioning, 'naive' for naive test. Defaults to 'homotopy'.
- **width** (*int, optional*) – same cp width. Defaults to 0
- **phi** (*float, optional*) – Double-CUSUM's parameter. Defaults to 0.5

Returns change-location, change-component and p-values of each changepoint.

Return type (array-like of shape (changepoint-num))

Examples

```
>>> result = multiseq_cp_dc_si(X, 1, 5)
>>> print(result)
[[10, 0, 1.1237052306791954e-12], [10, 1, 2.050601922239704e-09]]
```

si4ul.multiseq_cp_si.**multiseq_cp_scan_si**(*X*, *K*, *L*, *Xi=1*, *Sigma=1*, *test='hom'*, *width=0*)
computing valid p-values for changepoint with scan statistic by Selective Inference.

Parameters

- **X** (*ndarray of shape (component, location)*) – multi-dimensional sequence.
- **K** (*int*) – the number of change locations.
- **L** (*int*) – minimum segment length.
- **Xi** (*float or ndarray, optional*) – component's variance. Defaults to 1.
- **Sigma** (*float or ndarray, optional*) – location's variance. Defaults to 1.
- **test** (*str, optional*) – Set 'hom' for homotopy search, 'oc' for over-conditioning, 'naive' for naive test. Defaults to 'homotopy'.

- **width** (*int, optional*) – same cp width. Defaults to 0

Returns change-location, change-component and p-values of each changepoint.

Return type (array-like of shape (changepoint-num))

Examples

```
>>> result = multiseq_cp_scan_si(X, 1, 5)
>>> print(result)
[[10, 0, 1.1237052306791954e-12], [10, 1, 2.050601922239704e-09]]
```

si4ul.multiseq_cp_si.**plot_multiseq**(*X*, *true_cp=None*, *true_mean=None*)
Plot of multi-dimensional sequence.

Parameters

- **X** (*ndarray of shape (component, location)*) – multi-dimensional sequence.
- **true_cp** (*array-like of shape (components, true-cp-num+2)*) – the true change point position
- **true_mean** (*array-like of shape (components, true-cp-num+1)*) – true population mean

si4ul.multiseq_cp_si.**plot_multiseq_si**(*X*, *result*, *alpha=0.05*, *true_cp=None*, *true_mean=None*)
the result of multiseq changepoint after SI.

Parameters

- **X** (*ndarray of shape (component, location)*) – multi-dimensional sequence.
- **result** (*array-like of shape (changepoint-num)*) – change-location, change-component and p-values list
- **alpha** (*float*) – significance level. Defaults to 0.05
- **true_cp** (*array-like of shape (components, true-cp-num+2)*) – the true change point position
- **true_mean** (*array-like of shape (components, true-cp-num+1)*) – true population mean

CHAPTER

THREE

SI4UL.KMEANS_SI MODULE

si4ul.kmeans_si.**all_clusters_combination_test**(*obs_model*, *test_gene=None*, *sigma=1.0*)

post clustering inference for all cluster combinations. If test_gene is set, test is PCI_gene. otherwise, test is PCI_cluster.

Parameters

- **obs_model** (KMeans) – reffer to document of KMeans.
- **test_gene** (*int*) – feature to compare.
- **sigma** (*float*) – standard deviation in distribution.

Returns matrix of test statistics, matrix of homotopy PCI p-value and matrix of naive p-value.

Return type (array-like of shape(3, cluster_num, cluster_num))

Examples

```
>>> print(kmeans_si.all_clusters_combination_test(obs_model))
pci_cluster
(array([[nan, 5.24520442, 4.53785448], [nan, nan, 5.38552569], [nan, nan, nan]]),
array([[nan, 0.01863257, 0.00028306], [nan, nan, 0.00359755], [nan, nan, nan]]),
array([[nan, 1.06122705e-06, 3.37658148e-05], [nan, nan, 5.03368420e-07], [nan, nan,
↪ nan]]))
```

si4ul.kmeans_si.**kmeans**(*X*, *n_clusters*)

k-means clustering algorithm.

Parameters

- **X** (*array-like of shape (n, d)*) – data matrix.
- **n_clusters** (*int*) – number of cluster.

Returns reffer to document of KMeans.

Return type *KMeans*

Examples

```
>>> print(kmeans_si.kmeans(X, K))
<si4ul.si.kmeans_si.KMeans at 0x102aef700>
```

si4ul.kmeans_si.**pci_cluster**(*obs_model*, *comparison_clusters*, *sigma=1.0*, *max_iter=1000*, *random_seed=0*, *z_max=20*)

post clustering inference for test between clusters.

Parameters

- **obs_model** (KMeans) – reffer to document of KMeans.
- **comparison_clusters** (*array-like of shape(2)*) – set of clusters to compare.
- **sigma** (*float*) – standard deviation in distribution.
- **max_iter** (*int*) – upper limit count of iteration in k-means algorithm.
- **random_seed** (*int*) – seed of random for determine initial cluster.
- **z_max** (*float*) – upper limit of parameter z on test statisitcs vector.

Returns test statistics, homotopy PCI p-value and naive p-value.

Return type (float, float, float)

Examples

```
>>> print(kmeans_si.pci_cluster(obs_model, comparison_clusters))
(5.245204424402314, 0.018632573868904267, 1.0612270479959528e-06)
```

si4ul.kmeans_si.**pci_gene**(*obs_model*, *comparison_clusters*, *test_gene*, *sigma=1.0*, *max_iter=1000*, *random_seed=0*, *z_max=20*)

post clustering inference for test between clusters about a feature.

Parameters

- **obs_model** (KMeans) – reffer to document of KMeans.
- **comparison_clusters** (*array-like of shape(2)*) – set of clusters to compare.
- **sigma** (*float*) – standard deviation in distribution.
- **test_gene** (*int*) – feature to compare.
- **max_iter** (*int*) – upper limit count of iteration in k-means algorithm.
- **random_seed** (*int*) – seed of random for determine initial cluster.
- **z_max** (*float*) – upper limit of parameter z on test statisitcs vector.

Returns test statistics, homotopy PCI p-value and naive p-value.

Return type (float, float, float)

Examples

```
>>> print(kmeans_si.pci_gene(obs_model, comparison_clusters, gene_id))
(0.26352301450242993, 0.4212956294190716, 0.20005529456703786)
```

si4ul.kmeans_si.**plot_histogram**(*obs_model*, *comparison_clusters*, *test_gene*, *is_plot_norm=False*)
plot histogram of distribution per cluster using test.

Parameters

- **obs_model** (KMeans) – reffer to document of KMeans.
- **comparison_clusters** (*array-like of shape(2)*) – set of clusters to compare.
- **test_gene** (*int*) – feature to plot.
- **is_plot_norm** (*bool*) – whether plot normal distribution in background.

si4ul.kmeans_si.**plot_p_matrix**(*matrix*, *digit=3*, *alpha=0.05*)
plot matrix of p-value that is calculated by each cluster combinations.

Parameters

- **matrix** (*array-like of shape(cluster_num, cluster_num)*) – matrix of p-value.
- **digit** (*int*) – digit number to display.
- **alpha** (*float*) – significant level.

si4ul.kmeans_si.**plot_scatter**(*obs_model*, *comparison_clusters*, *show_dims*)
plot scatter data in inputted 2-dims per cluster using test.

Parameters

- **obs_model** (KMeans) – reffer to document of KMeans.
- **comparison_clusters** (*array-like of shape(2)*) – set of clusters to compare.
- **show_dims** (*array-like of shape(2)*) – set of dims to show.

si4ul.kmeans_si.**plot_statistics_matrix**(*matrix*, *digit=3*)
plot matrix of statistics that is calculated by each cluster combinations.

Parameters

- **matrix** (*array-like of shape(cluster_num, cluster_num)*) – matrix of test statistics.
- **digit** (*int*) – digit number to display.

si4ul.kmeans_si.**plot_violin**(*obs_model*, *test_gene*)
plot violin per cluster using test and other.

Parameters

- **obs_model** (KMeans) – reffer to document of KMeans.
- **test_gene** (*int*) – feature to plot.

3.1 si4ul.si.kmeans_si module

class si4ul.si.kmeans_si.**KMeans**(*X*, *n_clusters*, *max_iter=1000*, *random_seed=0*)

Bases: object

this class returns the results of k-means clustering. we can get from kmeans_si.kmeans(). other kmeans_si APIs need this object to input.

cluster_centers_
cluster center matrix.

Type array-like of shape(n_clusters, d)

count
count of iteration in k-means algorithm.

Type int

labels_
label vector that each data join.

Type array-like of shape(n)

label_num_list
list of the number of data contained in the cluster.

Type array-like of shape(n_clusters)

max_iter
upper limit count of iteration in k-means algorithm.

Type int

n_clusters
number of cluster.

Type int

random_seed
seed of random for determine initial cluster.

Type int

X
data matrix.

Type array-like of shape(n, d)

Examples

```
>>> kMeans.X
array([[ 1.32473015,  0.12584954], [ 0.17229311,  2.01200624], [ 1.47662313, -1.
↪28557418], [ 0.1302503 , -0.43927367], [-1.41546232,  0.13654847], [-1.05260842, ␣
↪1.20597647], [-0.14717876, -0.15950429], [-0.61262757,  0.05772617], [ 0.92854842,
↪ -0.49440228], [-0.80456805, -1.15935248]])
>>> kMeans.n_clusters
3
>>> kMeans.cluster_centers_
array([[ 1.47662313, -1.28557418], [ 0.55908753, -0.24183267], [-0.74259465,  0.
↪45058097]])
```

(continues on next page)

(continued from previous page)

```
>>> kMeans.labels_
array([1, 2, 0, 1, 2, 2, 1, 2, 1, 2])
>>> kMeans.label_num_list
[1, 4, 5]
>>> kMeans.count
2
```

CHAPTER

FOUR

SI4UL.SEGMENTATION_SI MODULE

si4ul.segmentation_si.**plot_histogram**(*local_white*)
plot histogram of pixel values in input image.

Parameters **local_white** (LocalWhite) – segmetation object.

si4ul.segmentation_si.**plot_histogram_region**(*local_white*, *display_statistics=False*)
plot histogram of pixel values per region.

Parameters **local_white** (LocalWhite) – segmetation object.

si4ul.segmentation_si.**psegi_thresholding**(*local_white*, *sigma=10*)
post segmentation inference for local white segmentation.

Parameters

- **local_white** (LocalWhite) – segmetation object.
- **sigma** (*float*) – standard deviation in distribution of input image.

Returns test statisitics, PSegI p-value, naive p-value.

Return type (float, float, float)

Examples

```
>>> print(segmentation_si.psegi_thresholding(local_white))
(31.08397312484288, 0.08085241645874175, 0.0)
```

si4ul.segmentation_si.**thresholding**(*image_path*, *result_dir_path*, *window_size=50*, *bias=1.1*, *is_blur=True*, *ksize=(11, 11)*, *sigma_x=0*, *sigma_y=0*, *is_output_regions=False*)
local white segmentation algorithm. If bias is bigger than 1, algorithm can detect black object. If bias is smaller than 1, algorithm can detect white object.

Parameters

- **image_path** (*string*) – image path to segmentation.
- **result_dir_path** (*string*) – directory to store the segmentation results.
- **window_size** (*int*) – length of the side of the square in the nearest pixel range.
- **bias** (*float*) – bias used in threshold determination. If it's bigger than 1, algorithm can detect black object. If it's smaller than 1, algorithm can detect white object.
- **isBlur** (*bool*) – whether to use Gaussian smoothing for image preprocessing.
- **ksize** (*Tuple(int, int)*) – the side of the convolution window for Gaussian smoothing.

- **sigma_x** (*float*) – standard deviation of x-coordinate for Gaussian smoothing.
- **sigma_y** (*float*) – standard deviation of y-coordinate for Gaussian smoothing.
- **is_output_regions** (*bool*) – wheter to make images of each regions.

Returns reffer to document of LocalWhite.

Return type *LocalWhite*

Examples

```
>>> print(segmentation_si.thresholding('./image.jpg', './result/image_18'))
<si4ul.si.segmentation_si.LocalWhite at 0x10e33efa0>
```

4.1 si4ul.si.segmentation_si module

class si4ul.si.segmentation_si.**LocalWhite**(*image_path*, *result_dir_path*, *window_size*, *bias*, *is_blur*, *ksize*, *sigma_x*, *sigma_y*)

Bases: object

this class returns the results of local white segmentation. we can get from segmentation_si.thresholding(). other segmentation_si APIs need this object to input.

bias
bias used in threshold determination. If it's bigger than 1, algorithm can detect black object. If it's smaller than 1, algorithm can detect white object.

Type float

image
input image of segmentation.

Type array-like of shape (image_height, image_width)

image_gaussian
image after Gaussian smoothing.

Type array-like of shape (image_height, image_width)

image_height
height of image.

Type int

image_original
image matrix read in from the input image.

Type array-like of shape (image_height, image_width)

image_path
image path to segmentation.

Type string

image_width
width of image.

Type int

result_dir_path
directory to store the segmentation results.

Type string

window_size
length of the side of the square in the nearest pixel range.

Type int

Examples

```
>>> localWhite.image
array([[147, 147, 147, ..., 155, 157, 157],
    [147, 147, 148, ..., 154, 156, 156],
    [148, 148, 148, ..., 153, 154, 154],
    ...,
    [159, 158, 157, ..., 152, 151, 150],
    [159, 159, 157, ..., 152, 150, 150],
    [159, 159, 157, ..., 152, 150, 149]], dtype=uint8)
>>> localWhite.image_height, localWhite.image_width
(88, 73)
```

PYTHON MODULE INDEX

S

INDEX

S

T

W

X