pyCFOFiSAX

Version 0.1.0

Lucas Foulon

sept. 19, 2021

Contents:

1 CFOF using *i*SAX trees **3**

2 Forest *i*SAX **5**

3 Tree *i*SAX **7**
3.1 Fonctions Numba pour Tree *i*SAX 10

4 Node *i*SAX **13**
4.1 Nœud Racine 13
4.2 Nœud Interne 14
4.3 Nœud Feuille 14

5 *i*SAX **15**

6 Pour commencer **19**
6.1 Installation 19
6.2 Utilisation 19

7 Réferences **21**

8 Autres **23**

Index des modules Python **25**

Index **27**

Cette documentation présente le fonctionnement des classes Python pour l'approximation des scores CFOF à l'aide d'un ou plusieurs arbres d'indexation *i*SAX.

CHAPITRE 1

CFOF using *i*SAX trees

class pyCFOFiSAX._cfofisax.**CFOFiSAX**

The class for *i*CFOF approximation using *i*SAX index trees. Contains the *i*SAX tree forest.

init_forest_isax(*size_word : int*, *threshold : int*, *data_ts : numpy.ndarray*, *base_cardinality : int = 2*, *number_tree : int = 1*, *indices_partition : Optional[list] = None*, *max_card_alphabet : int = 128*, *boolean_card_max : bool = True*)

Initializes the forest of *i*SAX trees. Requires the parameters of the class *ForestISAX*.

Paramètres

- **size_word** (*int*) – The size of the SAX words
- **threshold** (*int*) – The threshold, maximal size of nodes
- **data_ts** (*numpy.ndarray*) – The sequences to be inserted, to compute the stats of dataset
- **base_cardinality** (*int*) – The smallest cardinality to encode *i*SAX
- **number_tree** (*int*) – The number of *i*SAX trees in the forest
- **indices_partition** (*list*) – A list of indices list where, for each tree, specifies the indices of the sequences to be inserted
- **max_card_alphabet** (*int*) – if `boolean_card_max == True`, the maximum cardinality of *i*SAX encoding in each tree
- **boolean_card_max** (*bool*) – if `== True`, defines maximum cardinality for *i*SAX sequences encoding in each of the trees

score_icfof(*query : numpy.array*, *ntss : numpy.ndarray*, *rho=[0.001, 0.005, 0.01, 0.05, 0.1]*, *each_tree_score : bool = False*, *fast_method : bool = True*)

Compute the *i*CFOF approximations. Call one of the two functions according to the parameter `fast_method` :

— if `True` (default) : `vranglist_by_idtree_faster()`

— if `False` : `vranglist_by_idtree()`

Then sort the vrang list to get CFOF scores approximations based on `rho` parameter values.

Paramètres

- **query** (*numpy.array*) – The sequence to be evaluated
- **ntss** (*numpy.ndarray*) – Reference sequences
- **rho** (*list*) – Rho values for the computation of approximations
- **each_tree_score** (*bool*) – if *True*, returns the scores obtained in each of the trees
- **fast_method** (*bool*) – if *True*, uses the numpy functions for computation, otherwise goes through the tree via a FIFO list of nodes

Renvoie *i*CFOF score approximations

Type renvoyé numpy.ndarray

CHAPITRE 2

Forest *i*SAX

class pyCFOFiSAX._forest_iSAX.**ForestISAX**(*size_word : int*, *threshold : int*, *data_ts : numpy.ndarray*, *base_cardinality : int = 2*, *number_tree : int = 1*, *indices_partition : Optional[list] = None*, *max_card_alphabet : int = 128*, *boolean_card_max : bool = True*)

ForestISAX class containing one or more trees and pretreatment functions on the data contained in these trees

Paramètres

- **size_word** (*int*) – The size of the SAX words
- **threshold** (*int*) – The maximum threshold of nodes
- **data_ts** (*numpy.ndarray*) – The sequences to be inserted to extract the stats
- **base_cardinality** (*int*) – The smallest cardinality for encoding *i*SAX
- **number_tree** (*int*) – The number of TreeISAX trees in the forest
- **indices_partition** (*list*) – a list of index list where, for each tree, specifies the indices of

sequences to be inserted :param int max_card_alphabet : if `boolean_card_max == True`, the maximum cardinality of encoding *i*SAX in each of the trees :param boolean boolean_card_max : if == `True`, Defines a maximum cardinality for encoding *i*SAX Sequences in each of the trees

Variables **length_partition** (*list*) – The length of the SAX words in each tree (== `[size_word]` if ``number_tree

== 1``)

_count_nodes(*id_tree : int*)

The _count_nodes function returns the number of nodes and leaf nodes for a given tree. Uses `count_nodes_by_tree()`.

Paramètres **id_tree** (*int*) – The tree ID to be analyzed

Renvoie the number of internal nodes, the number of leaf nodes

Type renvoyé int, int

_init_trees(*data_ts : numpy.ndarray*, *max_card_alphabet : int*, *boolean_card_max : bool*)

Function that initializes the tree (s) when creating a ForestISAX object

Paramètres

- **data_ts** (*numpy.ndarray*) – The sequences to be inserted to extract the stats
- **max_card_alphabet** (*int*) – if `boolean_card_max == True`, The maximum cardinality of encoding *i*SAX

dans chacun des arbres :param boolean boolean_card_max : if `boolean_card_max == True`, defines maximum cardinality for encoding *i*SAX sequences in each tree

index_data(*new_sequences : numpy.ndarray*)

The Index_Data function allows you to insert a large number of sequences

Paramètres **new_sequences** (*numpy.ndarray*) – The sequences to be inserted

Renvoie The number of sequences (sub sequences) insert into the tree (in the trees)

Type renvoyé numpy.array

list_nodes(*id_tree : int*, *bool_print : bool = False*)

Returns lists of nodes and barycenters of the tree id_tree.Displays statistics on standard output if `bool_print == True` Uses `get_list_nodes_and_barycentre()`.

Paramètres

— **id_tree** (*int*) – The tree ID to be analyzed
— **bool_print** (*boolean*) – Displays the nodes stats on the standard output

Renvoie The list of nodes, the list of internal nodes, the list of barycenters

Type renvoyé list, list, list

number_nodes_visited(*query : numpy.array*, *ntss : numpy.ndarray*)

Count the number of average visited nodes in each tree for calculating the approximation.

Paramètres

— **query** (*numpy.array*) – The sequence to be evaluated
— **ntss** (*numpy.ndarray*) – Reference sequences

Renvoie Returns the number of nodes visited in each tree for the approximation *i*CFOF

Type renvoyé numpy.array

preprocessing_forest_for_icfof(*ntss : numpy.ndarray*, *bool_print : bool = False*, *count_num_node : bool = False*)

Allows us to call, for the `id_tree` to the pre-treatment for the calculation *i*CFOF

Paramètres

— **ntss** – Reference sequences
— **bool_print** (*boolean*) – if True, displays the times of each pre-treatment step
— **count_num_node** (*boolean*) – if True, count the number of nodes

Renvoie if count_num_node, returns the number of nodes contained in each tree

Type renvoyé numpy.array

CHAPITRE 3

Tree *i*SAX

class pyCFOFiSAX._tree_iSAX.**TreeISAX**(*size_word*, *threshold*, *data_ts*, *base_cardinality=2*, *max_card_alphabet=128*, *boolean_card_max=True*)

La classe TreeISAX contenant

— ISAX distribution of sequences to index
— and towards the first root node

Avertissement : In this version, the data_ts are mandatory to define in advance the future breakpoints

Paramètres

— **size_word** (*int*) – The number of Sax discretization for each sequence
— **threshold** (*int*) – The maximum capacity of the nodes of the tree
— **data_ts** (*numpy.ndarray*) – Sequence array to be inserted
— **base_cardinality** (*int*) – The smallest cardinality for encoding iSAX
— **max_card_alphabet** (*int*) – if self.boolean_card_max == True, Max cardinality for encoding iSAX

Variables

— **size_word** (*int*) – Number of letters contained in the SAX words indexed in the tree
— **threshold** (*int*) – Threshold before the separation of a sheet into two leaf nodes

_minmax_nodes()

Returns the breakpoints of the nodes of the tree. Uses _do_bkpt().

Renvoie The min and max breakpoints of the nodes of the tree

Type renvoyé numpy.ndarray

_minmax_obj_vs_node(*ntss_tmp*, *bool_print : bool = False*)

Compute distance min and max between the sequences ntss_tmp and the nodes of the tree.

Paramètres

— **ntss_tmp** (*numpy.ndarray*) – Reference sequences
— **bool_print** (*boolean*) – if True, Displays the times of each preprocessing step

Renvoie Minimum distances between sequences and nodes, Maximum distances between sequences and nodes

Type renvoyé numpy.ndarray, numpy.ndarray

_minmax_obj_vs_nodeleaf()
Computes the min and max distances between the ntss_tmp sequences and the leaf nodes of the tree.

> **Avertissement :** Attention must be executed after *_minmax_obj_vs_node()* and *distrib_nn_for_cdf()*.

count_nodes_by_tree()
The COUNT_NODES_BY_TREE function returns the number of nodes and leaf nodes of the shaft. Uses get_number_internal_and_terminal().

Renvoie the number of internal nodes, the number of leaves nodes

Type renvoyé int, int

distrib_nn_for_cdf(*ntss_tmp*, *bool_print : bool = False*)
Calculates the two indicators, average and standard deviation of the distances, necessary for the use of the CDF of the normal distribution. The computation of these indicators are described in Scoring Message Stream Anomalies in Railway Communication Systems, L.Foulon et al., 2019, ICDMWorkshop.

Paramètres
— **ntss_tmp** (*numpy.ndarray*) – Reference sequences
— **bool_print** (*boolean*) – and True, Displays the nodes stats on the standard output

Renvoie

Type renvoyé list(numpy.ndarray, numpy.array)

get_level_max()
Function to return the max level considering root level = 0

Renvoie The max depth level

Type renvoyé int

get_list_nodes_and_barycentre()
Returns Lists of Nodes and centroids

Renvoie List of nodes, List of Leaf Nodes, List of Leaf centroids

Type renvoyé list, list, list

get_list_nodes_leaf()
Returns List of Leaves Nodes

Renvoie List of leaves nodes

Type renvoyé list

get_nodes_of_level(*level : int*)
Function to return the nodes of a level, considering root level = 0

Paramètres **level** (*int*) – The level of the tree to evaluate

Renvoie The nodes of the ith level of the three

Type renvoyé list

get_number_internal_and_terminal()
Function to return the number of leaf nodes and internal nodes

Renvoie the number of internal nodes, the number of leaves nodes

Type renvoyé int, int

get_size()
Function to return the memory size of the tree, nodes and sequences contained in the tree

Renvoie Total memory size, nodes" memory size, memory size of the sequences

Type renvoyé int, int, int

`get_size_and_width_and_number_types_nodes()`

Feature grouping :
— get_size()
— get_width_of_all_level()
— get_number_internal_and_terminal()

Renvoie Total memory size, memory size of nodes, memory size of the sequences, the number of nodes on each level, the number of internal nodes, the number of sheet nodes, and the number of sequence inserted in the tree

Type renvoyé int, int, int, list, int, int, int

`get_width_of_all_level()`
Function to return the width of all levels in a list, considering root level = 0

Renvoie The number of node on each level of the tree

Type renvoyé list

`get_width_of_level(level : int)`
Return the width of a level, considering root level = 0

Renvoie the number of node on the level of the tree

Type renvoyé int

`insert(new_sequence)`
This insert function convert new sequence in PAA values then call the function insert_paa

Paramètres **new_sequence** (*numpy.array*) – The new sequence to be inserted

`insert_paa(new_paa)`
The insert function that directly calls the function of its root node

Paramètres **new_paa** (*numpy.array*) – The new sequence to be inserted

`number_nodes_visited(sub_query : numpy.array, ntss_tmp : numpy.ndarray)`
Account the number of average visited nodes in the tree for calculating the approximation.

Paramètres
— **sub_query** (*numpy.array*) – The sequence to be evaluated
— **ntss_tmp** (*numpy.ndarray*) – Reference sequences

Renvoie Returns the number of nodes visited in the tree for the approximation *i*CFOF

Type renvoyé numpy.array

`preprocessing_for_icfof(ntss_tmp, bool_print : bool = False, count_num_node : bool = False)`

Allows us to appeal, for the id_tree tree, to the two methods of preprocessing :
— `_minmax_obj_vs_node()`,
— `distrib_nn_for_cdf()`.

Paramètres
— **ntss_tmp** – Reference sequences
— **bool_print** (*boolean*) – if True, Displays the times of each preprocessing step
— **count_num_node** (*boolean*) – if True, count the number of nodes

Renvoie if `count_num_node` True, Returns the number of nodes in the tree

Rtypes int

`vrang_list(sub_query : numpy.array, ntss_tmp : numpy.ndarray)`
Get the vrang list for the `sub_query` sequence in the tree. Necessary for the calculation of the approximation. The same method faster but without the tree course : *vrang_list_faster()*.

Paramètres

— **sub_query** – The sequence to be evaluated
— **ntss_tmp** – Reference sequences (IE. Reference history)

Renvoie The vrang list of``sub_query``

Type renvoyé list(float)

vrang_list_faster(*sub_query : numpy.array*, *ntss_tmp : numpy.ndarray*)
Get the vrang list for the sub_query sequence in the tree. Necessary for the calculation of the approximation. This method is the fast version of the method *vrang_list()*.

Note : This method does not travel the tree, but directly prunes the leaves nodes. Preserved (uncut) leaves will be used by the approximation function.

Paramètres
— **sub_query** – The sequence to be evaluated
— **ntss_tmp** – Reference sequences (IE. Reference history) in PAA format

Renvoie The vrang list of``sub_query``

Type renvoyé list(float)

3.1 Fonctions Numba pour Tree *i*SAX

3.1.1 Pour l'obtention du vrang

pyCFOFiSAX._tree_iSAX.**vrang_seq_ref**(*distance*, *max_array*, *min_array*, *cdf_mean*, *cdf_std*, *num_ts_by_node*, *index_cdf_bin*, *cdf_bins*)
Calculates the vrang from the distance between the sequence to be evaluated and the reference sequence.

Paramètres
— **distance** (*float*) – The distance between the two sequences
— **max_array** (*np_array*) – Max distances between the nodes of the tree and the reference sequence
— **min_array** (*np_array*) – MIN distances between the nodes of the tree and the reference sequence
— **cdf_mean** (*np_array*) – The average distances between the nodes of the tree and the reference sequence
— **cdf_std** (*np_array*) – Dispersion of distances in each leaf node
— **num_ts_by_node** (*np_array*) – The number of sequence in each node sheet
— **index_cdf_bin** (*np_array*) – index of the cdf_bins CDF
— **cdf_bins** (*np_array*) – Normal distribution cdf values centered at the origin and standard deviation

Renvoie le vrang

Type renvoyé int

pyCFOFiSAX._tree_iSAX.**vrang_list_for_all_seq_ref**(*len_seq_list*, *distance*, *max_array*, *min_array*, *cdf_mean*, *cdf_std*, *num_ts_by_node*, *index_cdf_bin*, *cdf_bins*)
Uses the function vrang_seq_ref() For each reference sequence.

Paramètres
— **len_seq_list** (*float*) – The number of reference sequence
— **distance** (*np_array*) – The distance between the two sequences
— **max_array** (*np_ndarray*) – Max distances between the nodes of the tree and the reference sequence

— **min_array** (*np_ndarray*) – MIN distances between the nodes of the tree and the reference sequence
— **cdf_mean** (*np_ndarray*) – The average distances between the nodes of the tree and the reference sequence
— **cdf_std** (*np_array*) – Dispersion of distances in each leaf node
— **num_ts_by_node** (*np_array*) – The number of sequence in each node sheet
— **index_cdf_bin** (*np_array*) – The index of the CDF `cdf_bins`
— **cdf_bins** (*np_array*) – Normal distribution cdf values centered at the origin and standard deviation

Renvoie la liste des vrang

Type renvoyé np_array

3.1.2 Pour compter les nœuds visités

pyCFOFiSAX._tree_iSAX.**nodes_visited_for_seq_ref**(*distance*, *max_array*, *min_array*, *list_parent_node*)

pyCFOFiSAX._tree_iSAX.**nodes_visited_for_all_seq_ref**(*len_seq_list*, *distance*, *max_array*, *min_array*, *list_parent_node*)

CHAPITRE 4

Node *i*SAX

4.1 Nœud Racine

class pyCFOFiSAX._node.**RootNode**(*tree*, *parent*, *sax*, *cardinality*)

The RootNode class creates the only node of the ancestor tree common to all other nodes

Paramètres

- **tree** (*tree_iSAX*) – the tree in which the node is contained
- **parent** (*Node*) – The parent parent node
- **sax** (*numpy.array*) – SAX values of the node
- **cardinality** (*numpy.array*) – Cardinality of SAX values

_do_bkpt()

The _do_bkpt function calculates the min and max terminals of the node on each dimension of the node.

Renvoie an array containing the min terminals and one containing the max terminals

Type renvoyé numpy.array, numpy.array

get_nb_sequences() → int

Returns the number of sequences contained in the node and its descendants

Renvoie The number of sequences of the subtree

Type renvoyé int

get_sequences()

Returns the sequences contained in the node (leaf only) or its descendants

Renvoie Sequences

Type renvoyé numpy.ndarray

id_global = 0

Attribute to define an ID for each node

insert_paa(*new_paa*)

The insert_paa(new_paa) function to insert a new converted sequence into PAA

Paramètres **new_paa** – The converted sequence in PAA to insert

nb_sequences

The incremental computing part for CFOF

4.2 Nœud Interne

class pyCFOFiSAX._node.**InternalNode**(*tree*, *parent*, *sax*, *cardinality*, *sequences*)

The InternalNode class creates the internal nodes having at least one direct descendant, and a single direct ascendant

Paramètres

- **tree** (*tree_iSAX*) – the tree in which the node is contained
- **parent** (*Node*) – The parent parent node
- **sax** (*list*) – SAX values of the node
- **cardinality** (*numpy.array*) – Cardinality of Sax Values
- **sequences** (*numpy.ndarray*) – The sequences to be inserted in this node

split(*next_cardinality*, *mean*, *stdev*)

Calcule the next cardinality and split in two

Paramètres

- **next_cardinality** (*numpy.array*) – The list of next cardinalities
- **mean** (*numpy.array*) – The list of averages of distribution of sequence values on each dimension
- **stdev** (*numpy.array*) – The list of different types of distribution of sequence values on each dimension

4.3 Nœud Feuille

class pyCFOFiSAX._node.**TerminalNode**(*tree*, *parent*, *sax*, *cardinality*)

The TerminalNode class creates the leaves nodes having no descendant, and a single direct ascendant

Paramètres

- **tree** (*tree_iSAX*) – the tree in which the node is contained
- **parent** (*Node*) – The parent parent node
- **sax** (*list*) – SAX values of the node
- **cardinality** (*numpy.array*) – Cardinality of Sax Values

get_sequences()

Returns the sequences contained in the node

Renvoie The sequences contained in the node

Type renvoyé list

insert_paa(*ts_paa*)

Function that inserts a new sequence in PAA format

Paramètres **ts_paa** – The new Paa sequence

CHAPITRE 5

*i*SAX

class pyCFOFiSAX._isax.**IndexableSymbolicAggregateApproximation**(*n_segments*, *alphabet_size_min=2*, *mean=0.0*, *std=1.0*)

Indexable Symbolic Aggregate approXimation (iSAX) transformation.

First presented by J. Shieh & E. Keogh in *i*SAX : Indexing and Mining Terabyte Sized Time Series. Class that inherits the class `PiecewiseAggregateApproximation` proposed by Romain Tavenard in ``tslearn`` Available here <https://tslearn.readthedocs.io/en/stable/>`_.

Paramètres

- **n_segments** (*int*) – The number of letters in the word sax
- **alphabet_size_min** (*int*) – The minimum size of the Sax alphabet at initialization (2 default)
- **mean** (*float*) – The average of the distribution of encoder sequences (0.0 default)
- **std** (*float*) – The standard deviation of the distribution of encoder sequences (0.0 default)

_card_to_bkpt(*max_cardinality*)

Returns the breakpoints associated with the cardinations <= max_cardinality. The function calculates and stores the BKPT if they have never been calculated.

Paramètres **max_cardinality** (*int*) – Maximum cardinality

Renvoie Breakpoints associated with cardinality <= max_cardinality

Type renvoyé dict

_card_to_bkpt_only(*max_cardinality*)

Returns the breakpoints associated with cardinality == max_cardinality. The function calculates and stores the BKPs if they have never been calculated.

Paramètres **max_cardinality** (*int*) – cardinality

Renvoie Breakpoints associated with cardinality == max_cardinality

Type renvoyé list

_row_sax_word_array(*ntss_tmp*, *bigger_cardinality*, *size_word*)

Convert all sequences according to the different cardinality of the tree. For each cardinality, uses `transform_sax()`.

Paramètres

— **ntss_tmp** – The sequences to be analyzed
— **bigger_cardinality** (*int*) – The greatest cardinality *i*SAX of the tree
— **size_word** (*int*) – The size of the SAX sequences of the tree

Renvoie SAX words from all ntss_tmp sequences according to all the cardinalities of the tree, a dict returning the cardinality index *i*SAX

Type renvoyé numpy.ndarray, dict

_transform(*X*, *card*)
Transforms X data in parameter first into PAA and then in cardinate cardinality card.

Paramètres
— **X** (*numpy.ndarray*) – Data to transform
— **card** (*int*) – Cardinality to use for processing

Renvoie Transformed data in SAX

Type renvoyé numpy.ndarray

_transform_paa_to_isax(*X_paa*, *card*)
Transforms X_paa data into *i*SAX parameters according to cardinality card.

Paramètres
— **X_paa** (*numpy.ndarray*) – PAA data to transform into *i*SAX
— **card** (*list*) – Cardinalities to use for processing

Renvoie Transformed data in SAX

Type renvoyé numpy.ndarray

_transform_sax(*X*, *card*)
Transforms X data in parameter first into PAA and then in cardinate cardinality card.

Paramètres
— **X** (*numpy.ndarray*) – Data to transform
— **card** (*int*) – Cardinality to use for processing

Renvoie Transformed data in SAX

Type renvoyé numpy.ndarray

fit(*X*)
Prepares the data for encoding *i*SAX according to``PiecewiseAggregateApproximation``

Renvoie Received data for encoding, defined by tslearn

Type renvoyé numpy.ndarray of PiecewiseAggregateApproximation

fit_transform(*X*, *card*, ***fit_params*)
Prepares the X data provided in parameter for encoding``tslearn``. Then transforms the X data provided as a parameter first in PAA and then in cardinate cardinality card.

Paramètres
— **X** (*numpy.ndarray*) – Data to transform
— **card** (*int*) – Cardinality to use for processing

Renvoie data transformed into SAX

Type renvoyé numpy.ndarray

transform(*X*, *card*)
Prepares the X data provided in parameter for encoding tslearn. Then transforms X data in parameter first into PAA and then in cardinatlity of cardinality card.

Paramètres
— **X** (*numpy.ndarray*) – Data to transform
— **card** (*int*) – Cardinality to use for processing

Renvoie Transformed data in SAX

Type renvoyé numpy.ndarray

transform_paa(*X*)

Prepares the X data provided in parameter for encoding``tslearn``. Then transforms X data into parameter in PAA.

Paramètres **X** (*numpy.ndarray*) – Data to transform

Renvoie Transformed data in PAA

Type renvoyé numpy.ndarray

transform_paa_to_isax(*X_paa*, *card*)

Prepares X_paa data provided as a parameter for encoding``tslearn``. Then transforms X_paa data into *i*SAX parameter according to cardinalities card.

Paramètres

— **X_paa** (*numpy.ndarray*) – PAA data to transform into *i*SAX

— **card** (*list*) – Cardinalities to use for processing

Renvoie Transformed data in SAX

Type renvoyé numpy.ndarray

transform_sax(*X*, *card*)

Prepares the X data provided in parameter for encoding``tslearn``. Then transforms X data in parameter first into PAA and then in cardinate cardinality``card``.

Paramètres

— **X** (*numpy.ndarray*) – Data to transform

— **card** (*int*) – Cardinality to use for processing

Renvoie Transformed data in SAX

Type renvoyé numpy.ndarray

CHAPITRE 6

Pour commencer

6.1 Installation

Lancer `pip install -r requirements.txt` ou `python3 -m pip install -r requirements.txt`.

6.2 Utilisation

```
>>> from pyCFOFiSAX import CFOFiSAX
>>> cfof_isax = CFOFiSAX()
```

Un jeu de données artificiel *Clust2* contenant 10000 séquences en dimension 200 est disponible dans le projet avec les résultats CFOF (attention : ce ne sont pas les scores CFOFiSAX) pour $\varrho = [0.01, 0.05, 0.1]$:

```
>>> import numpy as np
>>> # chargement du jeu, avec usecols=list(range(0, 200)) pour ne pas charger les scores
>>> ndarray_dataset = np.genfromtxt("pyCFOFiSAX/tests/data_test/data/clust2_200d_
↪20200319_125226_withrealcfof.csv",
>>                                  delimiter=',',
>>                                  skip_header=1,
>>                                  usecols=list(range(0, 200)))
>>> ndarray_dataset.shape
(10000, 200)
```

Initialisation de la forêt avec 20 arbres *i*SAX :

```
>>> cfof_isax.init_forest_isax(size_word=200,
>>                            threshold=30,
>>                            data_ts=ndarray_dataset,
>>                            base_cardinality=2, number_tree=20)
```

Insertion des données :

```
>>> cfof_isax.forest_isax.index_data(ndarray_dataset)
```

Puis pré-traitement :

```
>>> # va afficher les temps de pré-traitement pour chaque arbre et afficher le nombre de␣
↪nœuds dans chaque arbre
>>> cfof_isax.forest_isax.preprocessing_forest_for_icfof(ndarray_dataset,
>>>                                                       bool_print=True, count_num_
↪node=True)
```

Ensuite, le calcul de l'approximation *i*CFOF, pour la ième séquence est effectué avec la fonction :

```
>>> tmp_ite = np.random.randint(0,10000)
>>> tmp_ite
8526
>>> score = cfof_isax.score_icfof(
>>>          ndarray_dataset[tmp_ite], ndarray_dataset,
>>>          rho=[0.1], each_tree_score=True,
>>>          fast_method=True)
>>> # les approximations pour chaque valeur de rho sont dans score[0]
>>> score[0]
array([0.21357929])
>>> # si each_tree_score=True,
>>> # il est possible de regarder les scores obtenus dans chaque arbre avec score[1]
```

Pour connaître le nombre de nœuds visités dans chaque arbre lors du calcul :

```
>>> cfof_isax.forest_isax.number_nodes_visited(ndarray_dataset[tmp_ite],
>>>                                            ndarray_dataset)
array([1086.    , 1044.    , 1061.    , 1087.    , 1013.    , 1115.    ,
       1067.    , 1101.    , 1069.    , 1065.    , 1104.    , 1104.    ,
       1115.    , 1059.    , 1009.    , 1001.    , 1097.    , 1018.    ,
       1081.    , 1084.    ,  838.215 ,  804.8092,  826.3052,  853.1703,
        769.1603,  870.4031,  829.9387,  853.9431,  833.5567,  830.6371,
        849.1152,  861.5018,  864.5723,  820.4267,  762.5597,  763.3633,
        863.6749,  785.884 ,  850.051 ,  832.9658])
```

CHAPITRE 7

Réferences

— Scoring Message Stream Anomalies in Railway Communication Systems, L.Foulon et al., 2019, ICDMWorkshop
— Approximation du score CFOF de détection d'anomalie dans un arbre d'indexation iSAX: Application au contexte SI de la SNCF, L.Foulon et al., 2019, Actes de la conférence EGC'2019
— CFOF: A Concentration Free Measure for Anomaly Detection, F. Angiulli, 2020, TKDD
— Concentration Free Outlier Detection, F. Angiulli, 2017, ECML-PKDD
— iSAX 2.0: Indexing and Mining One Billion Time Series, A. Camerrea et al., 2010, ICDM
— iSAX: Disk-Aware Mining and Indexing of Massive Time Series Datasets, J. Shieh et al., 2009, DMKD
— Unsupervised real-time anomaly detection for streaming data, Ahmad, S. et al., 2017, Neurocomputing
— Tslearn, A Machine Learning Toolkit for Time Series Data, R.Tavenard et al., 2020 , JMLR

CHAPITRE 8

Autres

— genindex

Index des modules Python

p

pyCFOFiSAX._node, 13

Index

Symboles

_card_to_bkpt() (*méthode pyCFOFiSAX._isax.IndexableSymbolicAggregateApproximation*), 15
_card_to_bkpt_only() (*méthode pyCFOFiSAX._isax.IndexableSymbolicAggregateApproximation*), 15
_count_nodes() (*méthode pyCFOFiSAX._forest_iSAX.ForestISAX*), 5
_do_bkpt() (*méthode pyCFOFiSAX._node.RootNode*), 13
_init_trees() (*méthode pyCFOFiSAX._forest_iSAX.ForestISAX*), 5
_minmax_nodes() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 7
_minmax_obj_vs_node() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 7
_minmax_obj_vs_nodeleaf() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 8
_row_sax_word_array() (*méthode pyCFOFiSAX._isax.IndexableSymbolicAggregateApproximation*), 15
_transform() (*méthode pyCFOFiSAX._isax.IndexableSymbolicAggregateApproximation*), 16
_transform_paa_to_isax() (*méthode pyCFOFiSAX._isax.IndexableSymbolicAggregateApproximation*), 16
_transform_sax() (*méthode pyCFOFiSAX._isax.IndexableSymbolicAggregateApproximation*), 16

C

CFOFiSAX (*classe dans pyCFOFiSAX._cfofisax*), 3
count_nodes_by_tree() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 8

D

distrib_nn_for_cdf() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 8

F

fit() (*méthode pyCFOFiSAX._isax.IndexableSymbolicAggregateApproximation*), 16
fit_transform() (*méthode pyCFOFiSAX._isax.IndexableSymbolicAggregateApproximation*), 16
ForestISAX (*classe dans pyCFOFiSAX._forest_iSAX*), 5

G

get_level_max() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 8
get_list_nodes_and_barycentre() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 8
get_list_nodes_leaf() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 8
get_nb_sequences() (*méthode pyCFOFiSAX._node.RootNode*), 13
get_nodes_of_level() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 8
get_number_internal_and_terminal() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 8
get_sequences() (*méthode pyCFOFiSAX._node.RootNode*), 13
get_sequences() (*méthode pyCFOFiSAX._node.TerminalNode*), 14
get_size() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 8
get_size_and_width_and_number_types_nodes() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 8
get_width_of_all_level() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 9
get_width_of_level() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 9

I

id_global (*attribut pyCFOFiSAX._node.RootNode*), 13

index_data() (*méthode pyCFOFiSAX._forest_iSAX.ForestISAX*), 6
IndexableSymbolicAggregateApproximation (*classe dans pyCFOFiSAX._isax*), 15
init_forest_isax() (*méthode pyCFOFiSAX._cfofisax.CFOFiSAX*), 3
insert() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 9
insert_paa() (*méthode pyCFOFiSAX._node.RootNode*), 13
insert_paa() (*méthode pyCFOFiSAX._node.TerminalNode*), 14
insert_paa() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 9
InternalNode (*classe dans pyCFOFiSAX._node*), 14

L

list_nodes() (*méthode pyCFOFiSAX._forest_iSAX.ForestISAX*), 6

M

module
    pyCFOFiSAX._node, 13

N

nb_sequences (*attribut pyCFOFiSAX._node.RootNode*), 13
nodes_visited_for_all_seq_ref() (*dans le module pyCFOFiSAX._tree_iSAX*), 11
nodes_visited_for_seq_ref() (*dans le module pyCFOFiSAX._tree_iSAX*), 11
number_nodes_visited() (*méthode pyCFOFiSAX._forest_iSAX.ForestISAX*), 6
number_nodes_visited() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 9

P

preprocessing_for_icfof() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 9
preprocessing_forest_for_icfof() (*méthode pyCFOFiSAX._forest_iSAX.ForestISAX*), 6
pyCFOFiSAX._node
    module, 13

R

RootNode (*classe dans pyCFOFiSAX._node*), 13

S

score_icfof() (*méthode pyCFOFiSAX._cfofisax.CFOFiSAX*), 3
split() (*méthode pyCFOFiSAX._node.InternalNode*), 14

T

TerminalNode (*classe dans pyCFOFiSAX._node*), 14
transform() (*méthode pyCFOFiSAX._isax.IndexableSymbolicAggregateApproximation*), 16
transform_paa() (*méthode pyCFOFiSAX._isax.IndexableSymbolicAggregateApproximation*), 16
transform_paa_to_isax() (*méthode pyCFOFiSAX._isax.IndexableSymbolicAggregateApproximation*), 17
transform_sax() (*méthode pyCFOFiSAX._isax.IndexableSymbolicAggregateApproximation*), 17
TreeISAX (*classe dans pyCFOFiSAX._tree_iSAX*), 7

V

vrang_list() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 9
vrang_list_faster() (*méthode pyCFOFiSAX._tree_iSAX.TreeISAX*), 10
vrang_list_for_all_seq_ref() (*dans le module pyCFOFiSAX._tree_iSAX*), 10
vrang_seq_ref() (*dans le module pyCFOFiSAX._tree_iSAX*), 10