mdcgenpy Documentation

Daniel Ferreira

May 01, 2023

Contents:

CHAPTER 1

mdcgenpy package

1.1 Subpackages

1.1.1 mdcgenpy.clusters package

Submodules

mdcgenpy.clusters.distributions module

class mdcgenpy.clusters.distributions.**Distribution**(*f*, ***kwargs*)
Bases: object

mdcgenpy.clusters.distributions.**check_input**(*distributions*)
Checks if the input distributions are valid. That is, check if they are either strings or functions. If they are strings, also check if they are contained in *distributions_list*.

Parameters **distributions** (*list of list of (str or function)*) – Distributions given as input.

Returns Functions for the distributions given as input.

Return type (list of list of function)

mdcgenpy.clusters.distributions.**distributions_list = {'gamma': <function <lambda>>, 'gap'**
List of distributions for which you can just provide a string as input.

mdcgenpy.clusters.distributions.**gap**(*shape*, *param*)

mdcgenpy.clusters.distributions.**get_dist_function**(*d*)
Transforms distribution name into respective function.

Parameters **d** (*str or function*) – Input distribution str/function.

Returns Actual function to compute the intended distribution.

Return type function

mdcgenpy.clusters.distributions.**valid_distributions = ['gaussian', 'normal', 'triangular',**
List of valid strings for distributions.

mdcgenpy.clusters.generate module

mdcgenpy.clusters.generate.**compute_batch**(*clus_cfg*, *n_samples*)
Generates one batch of data.

Parameters

- **clus_cfg** (*clusters.DataConfig*) – Configuration.
- **n_samples** (*int*) – Number of samples in the batch.

Returns Generated sample.

Return type np.array

mdcgenpy.clusters.generate.**generate_clusters**(*clus_cfg*, *batch_size=0*)
Generate data.

Parameters

- **clus_cfg** (*clusters.DataConfig*) – Configuration.
- **batch_size** (*int*) – Number of samples for each batch.

Yields *np.array* – Generated samples. np.array: Labels for the samples.

mdcgenpy.clusters.generate.**generate_mass**(*clus_cfg*)
Get the number of samples to generate for each cluster.

Parameters **clus_cfg** (*clusters.DataConfig*) – Configuration

Returns

Array with len == nr of clusters, where each entry is the number of samples in the corresponding to generate in the corresponding cluster.

Return type np.array

mdcgenpy.clusters.generate.**get_rotation_matrix**(*n_feats*)

mdcgenpy.clusters.generate.**locate_centroids**(*clus_cfg*)
Generate locations for the centroids of the clusters.

Parameters **clus_cfg** (*clusters.DataConfig*) – Configuration.

Returns Matrix (n_clusters, n_feats) with positions of centroids.

Return type np.array

Module contents

class mdcgenpy.clusters.**Cluster**(*cfg*, *idx*, *corr_matrix=None*)
Bases: object

Contains the parameters of an individual cluster.

__init__(*cfg*, *idx*, *corr_matrix=None*)

Parameters

- **cfg** (ClusterGenerator) – Configuration of the data.

- **idx** (*int*) – Index of a cluster.
- **corr_matrix** (*np.array*) – Valid correlation matrix to use in this cluster.

compactness_factor

corr

distributions

generate_data(*samples*)

mv

n_feats

n_noise

rotate

scale

settables = ['distributions', 'mv', 'corr', 'compactness_factor', 'scale', 'rotate', '
List of settable properties of Cluster. These are the parameters which can be set at a cluster level, and override the parameters of the cluster generator.

class mdcgenpy.clusters.**ClusterGenerator**(*seed=1*, *n_samples=2000*, *n_feats=2*, *k=5*, *min_samples=0*, *possible_distributions=None*, *distributions=None*, *mv=True*, *corr=0.0*, *compactness_factor=0.1*, *alpha_n=1*, *scale=True*, *outliers=50*, *rotate=True*, *add_noise=0*, *n_noise=None*, *ki_coeff=3.0*, ***kwargs*)

Bases: object

Structure to handle the input and create clusters according to it.

__init__(*seed=1*, *n_samples=2000*, *n_feats=2*, *k=5*, *min_samples=0*, *possible_distributions=None*, *distributions=None*, *mv=True*, *corr=0.0*, *compactness_factor=0.1*, *alpha_n=1*, *scale=True*, *outliers=50*, *rotate=True*, *add_noise=0*, *n_noise=None*, *ki_coeff=3.0*, ***kwargs*)

Parameters

- **seed** (*int*) – Seed for the generation of random values. Useful for consistency.
- **n_samples** (*int*) – Number of samples to generate.
- **n_feats** (*int*) – Number of dimensions/features for each sample.
- **k** (*int or list of int*) – Number of clusters to generate. If input is a list, each element in it specifies the number of samples in each cluster. In that case, the number of clusters will be the length of the list.
- **min_samples** (*int*) – Minimum number of samples in each cluster. If 0, the default minimum for a cluster with N samples is $N/($

generate_data(*batch_size=0*)

get_cluster_configs()

mass

class mdcgenpy.clusters.**ScheduledClusterGenerator**(*schedule*, **args*, ***kwargs*)

Bases: *mdcgenpy.clusters.ClusterGenerator*

This cluster generator takes a schedule and all the ClusterGenerator arguments, and activates only the specified clusters in the schedule, for each time step. A time step is defined as one get call to `self.mass`, which is done when generating each new batch. That is, one time step is one call to *generate.compute_batch()*.

__init__(*schedule*, **args*, ***kwargs*)

Parameters

- **schedule** (*list*) – List in which each element contains the indexes of the clusters active in the respective time step.
- ***args** – args for *ClusterGenerator.__init__()*.
- ****kwargs** – kwargs for *ClusterGenerator.__init__()*.

mass

1.1.2 mdcgenpy.interface package

Submodules

mdcgenpy.interface.json_processing module

mdcgenpy.interface.json_processing.**get_cluster_generator**(*input_file*)
Parses a JSON file and generates a ClusterGenerator. :param input_file: Input file. :type input_file: str or file

Returns Resulting ClusterGenerator from the input file.

Return type *ClusterGenerator*

mdcgenpy.interface.json_processing.**get_input_data**(*input_file*)
Helper function for get_cluster_generator() :param input_file: Input file. :type input_file: str or file

Returns data corresponding to JSON input.

Return type dict

Module contents

1.1.3 mdcgenpy.mdcgenutils package

Module contents

mdcgenpy.mdcgenutils.**initialize**(*clust_config*)

1.2 Module contents

CHAPTER 2

JSON Format Specification

2.1 Example

The following is an example JSON configuration file for mdcgenpy:

```
{
    "n_samples": 3000,
    "n_feats": 2,
    "k": 3,
    "possible_distributions":["gaussian", "uniform"],
    "mv": true,
    "corr": 0.0,
    "compactness_factor": 0.1,
    "alpha_n": 1,
    "outliers": 50,
    "rotate": true,
    "clusters": [
        {"distributions": "uniform", "corr": 0.5},
        {},
        {"mv": null, "rotate": false}
    ]
}
```

As is usual in mdcgenpy, all parameters are optional.

2.2 General Format

In general, the format of an input JSON file must be something of this type:

```
{
  GENERATOR_PARAM_1: VAL,
  GENERATOR_PARAM_2: VAL,
```

(continues on next page)

(continued from previous page)

```
  ...
  GENERATOR_PARAM_N: VAL,
  "clusters": [
    {CLUSTER_1_PARAM_1: VAL, ..., CLUSTER_1_PARAM_N: VAL},
    {CLUSTER_2_PARAM_1: VAL, ..., CLUSTER_2_PARAM_N: VAL},
    ...
    {CLUSTER_M_PARAM_1: VAL, ..., CLUSTER_M_PARAM_N: VAL},
  ]
}
```

The generator parameters are as defined in the *Cluster Generator class*. The `"clusters"` keyword is for overriding the generator parameters for specific clusters.

All the parameters in the JSON file are optional (including the `"clusters"` keyword).

2.3 Overriding Parameters for Specific Clusters

After the generator parameters, there is an (optional) `"clusters"` keyword. If the `"clusters"` keyword is supplied, a list of at most the same length as the number of clusters must be supplied. Each element of this list contains cluster-specific parameters, which overrule the general parameters of the cluster generator, for that cluster.

For a complete list of parameters which are acceptable for specific clusters, check *settables*.

CHAPTER 3

mdcgenpy

mdcgenpy is a **M**ultidimensional **D**ataset for **C**lustering **Gen**erator. This tool is aimed at researchers looking for synthetic datasets, in particular for testing clustering algorithms. A variety of customization options are available, in order to allow for a wide range of use cases.

Using the generator is simple, and can even be used without parameters:

```
import mdcgenpy

# Initialize cluster generator (all parameters are optional)
cluster_gen = mdcgenpy.clusters.ClusterGenerator()

# Get tuple with a numpy array with samples and another with labels
data = cluster_gen.generate_data()
```

3.1 Generating data outside Python

It is also possible to use mdcgenpy without knowing python.

To do this, you just need to give as input a JSON file (check specification details *here*). Using the `mdcgenpy.py` script, the output will be sent in CSV format to stdout.

Example:

```
$ ./mdcgenpy.py input_parameters.json > output.csv
```

3.2 Features

- Efficient code, compatible with Python 2 and Python 3.
- Various possible distributions for the clusters are available out-of-the-box, and custom distributions are also allowed.

- Parameters allow for control over the overlap of the clusters, outliers, noise, correlation inside each cluster, etc.

3.3 Installation

3.4 Support

3.5 License

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

Symbols

C

D

G

I

L

M

N

R

S

V