
Conmo

Release 1.0.1

Grupo de Metrología y Modelos (MyM)

Jul 01, 2022

CONTENTS

1	What is Conmo?	3
2	User Guide	5
2.1	Quickstart Guide	5
3	Examples	9
3.1	Examples	9
4	API Reference	13
4.1	API Reference	13
5	Development Guide	73
5.1	Development guide	73
6	Known Issues & Limitations	83
6.1	Known Issues & Limitations	83
7	Frequently Asked Questions	85
7.1	Frequently Asked Questions	85
8	Release Notes	87
8.1	Release Notes	87
	Python Module Index	89
	Index	91

Release v1.0.1.

WHAT IS CONMO?

Conmo is a framework developed in Python whose main objective is to facilitate the execution and comparison of different experiments mainly related to Anomaly Detection and Condition Monitoring. These experiments consist of a series of concatenated stages forming a pipeline architecture, i.e. the output of one stage is the input of the next one. This framework aims to provide a way to standarize machine learning experiments, thus being able to reconstruct result tables of scientific papers.

2.1 Quickstart Guide

2.1.1 Requirements

Conmo was developed under Python version 3.7.11 so it should work with similar or more recent versions. However, we cannot claim this to be true, so we recommend using the same version. To be able to use **Conmo** you need to have installed a Python interpreter and the following libraries on your computer:

- Numpy
- Pandas
- Tensorflow
- Scikit-Learn
- Scipy
- Requests
- Pyarrow

If you want to make a contribution by modifying code and documentation you need to include these libraries as well:

- Sphinx
- Sphinx-rtd-theme
- Isort
- Autopep8

We suggest to create a new virtual enviroment using the Conda package manager and install there all dependences.

2.1.2 Installation

The fastest way to get started with Conmo is to install it via the pip command.

```
pip install conmo
```

And then you will be able to open a Python interpreter and try running.

```
import conmo
```

Some TensorFlow warnings might come up if your computer doesn't have installed a GPU, although that's not a problem for running Conmo.

You can also install Conmo manually downloading the source code from the Github repository.

```
git clone https://github.com/MyM-Uniovi/conmo.git
cd conmo
```

Then if you haven't prepared manually a conda environment, you can execute the shell-script `install_conmo_conda.sh` to install all the dependences and create a Conda environment with Python 3.7.

```
cd scripts
./install_conmo_conda.sh conda_env_name
```

If your operating system is MacOS, please, check [Known issues & Limitations section](#) for more information about compatibility of Conmo with Apple M1 and M2 CPUs.

If your operating system is not Unix-like and you are using Windows 10/11 OS you can create the Conda environment manually or use the Windows Subsystem for Linux (WSL) tool. For more information about its installation, please refer to [Microsoft's official documentation](#).

To check if the Conda environment is activated you should see a `(conda_env_name)` in your command line. If it is not activated, then you can activate it using:

```
conda activate conda_env_name
```

2.1.3 Overview

The experiments in Conmo have a pipeline-based architecture. A pipeline consists of a chain of processes connected in such a way that the output of each element of the chain is the input of the next, thus creating a data flow. Each of these processes represents one of the typical generic steps in Machine Learning experiments:

Datasets

Defines the dataset used in the experiment which will be the starting data of the chain. Here the dataset will be loaded and parsed to a standard format.

Splitters

Typically in Machine Learning problems the data has to be splitted into train data and test data. Also here you can apply Cross-Validation techniques.

Preprocesses

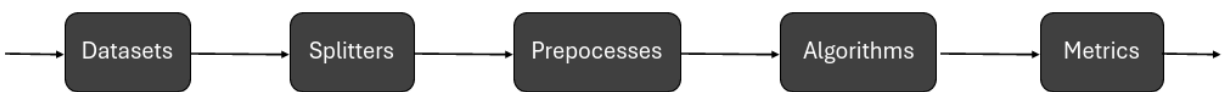
Defines the sequence of preprocesses to be applied over the dataset to manipulate the data before any algorithm is executed.

Algorithms

Defines the different algorithms which will be executed over the same input data stream (as a result of the previous stage). It can be one or several.

Metrics

Defines the different metrics that can be used to evaluate the results obtained from the algorithms.



Further details and documentation about modules, functions and parameters are provided in the [API Reference](#).

2.1.4 Running an experiment

Here is a brief example on how to use the different conmo modules to reproduce an experiment. In this case with the predefined splitter of the Server Machine Dataset, Sklearn's MinMaxScaler as preprocessing, PCAMahalanobis as algorithm and accuracy as metric.

1. Import the module if it hasn't been imported yet and other dependences:

```
1 from sklearn.preprocessing import MinMaxScaler
2
3 from conmo import Experiment, Pipeline
4 from conmo.algorithms import PCAMahalanobis
5 from conmo.datasets import ServerMachineDataset
6 from conmo.metrics import Accuracy
7 from conmo.preprocesses import SklearnPreprocess
8 from conmo.splitters import SklearnSplitter
9 from sklearn.model_selection import PredefinedSplit
10 from sklearn.preprocessing import MinMaxScaler
```

2. Configure the different stages of the pipeline:

```
1 dataset = ServerMachineDataset('1-01')
2 splitter = SklearnSplitter(splitter=PredefinedSplit(dataset.sklearn_
↳ predefined_split()))
3 preprocesses = [
4     SklearnPreprocess(to_data=True, to_labels=False,
5                       test_set=True, preprocess=MinMaxScaler()),
6 ]
7 algorithms = [
8     PCAMahalanobis()
9 ]
10 metrics = [
11     Accuracy()
12 ]
13 pipeline = Pipeline(dataset, splitter, preprocesses, algorithms, metrics)
```

3. Create an experiment with the configured pipeline. The first parameter is a list of the pipelines that will be included in the experiment. It can be one or more. The second parameter is for statistical testing between results, but this part is still under development and therefore it cannot be used:

```
1 experiment = Experiment([pipeline], [])
```

4. Start running the experiment by calling `launch()` method:

```
1 experiment.launch()
```

5. As a result of the execution of the experiment a specific folder structure will be created in `~/conmo`:

/data

This directory contains the various datasets that have already been imported (downloaded and parsed) and are therefore already available for use. *They are stored in parquet format for better compression.* For each of the subdatasets included in each dataset, there will be a data file and a labels file.

/experiments

This directory contains all the executions of an experiment in Conmo in chronological order. Each directory corresponds to an experiment and has in its name a timestamp with the time and day when this experiment was run. Within each experiment directory there will be another one for each pipeline, and within this one there will be as many directories as the number of steps each pipeline has been determined to contain. These folders contain the input and output data used by each step of the pipeline. They are also stored in parquet format, in the same way as the datasets in the /data folder.

EXAMPLES

3.1 Examples

A handful of example experiments can be found in the “examples” directory of the repository. These are listed below:

3.1.1 NASA TurboFan Degradation

This example can be found in *nasa_cmapss.py* file. The chosen dataset is NASA’s Turbofan engine degradation simulation data set. It is a dataset widely used in multivariate time series anomaly detection and condition monitoring problems. The splitter used is the Sklearn Predefined Split. For more information see the [Scikit-Learn documentation](#). Regarding preprocessing, several have been used. The Savitzky-Golay filter, RUL Imputation and Binarizer are already implemented in Conmo. The MinMaxScaler is a Sklearn preprocessing (more information [here](#)) that has been packaged using SklearnPreprocess. Finally, two custom preprocesses for data cleaning and label renaming have been defined using the CustomPreprocess wrapper. To create these preprocesses just create a function that has as parameters the Pandas Dataframes for data and labels. The algorithms used were dimensionality reduction with PCA together with Mahalanobis distance calculation and One Class Support Vector Machine. Finally, the metric used was Accuracy.

```
1 import pandas as pd
2 from sklearn.model_selection import PredefinedSplit
3 from sklearn.preprocessing import MinMaxScaler
4
5 from conmo import Experiment, Pipeline
6 from conmo.algorithms import OneClassSVM, PCAMahalanobis
7 from conmo.datasets import NASATurbofanDegradation
8 from conmo.metrics import Accuracy
9 from conmo.preprocesses import (Binarizer, CustomPreprocess, RULImputation,
10                                SavitzkyGolayFilter, SklearnPreprocess)
11 from conmo.splitters import SklearnSplitter
12
13 # First custom preprocess definition
14 def data_cleanup(data: pd.DataFrame, labels: pd.DataFrame) -> (pd.DataFrame, pd.
15     ↳ DataFrame):
16     # Reduce columns
17     columns = ['T30', 'T50', 'P30']
18     sub_data = data.loc[:, columns]
19
20     # Rename columns
21     sub_data = sub_data.rename(columns={'T50': 'TGT'})
22
23     # Calculate FF
```

(continues on next page)

(continued from previous page)

```

23     sub_data.loc[:, 'FF'] = data.loc[:, 'Ps30'] * data.loc[:, 'phi']
24     sub_data.head()
25
26     return sub_data, labels
27
28 # Second custom preprocess definition
29 def rename_labels(data: pd.DataFrame, labels: pd.DataFrame) -> (pd.DataFrame, pd.
30     DataFrame):
31     # Rename labels from 'rul' to 'anomaly'
32     labels.rename(columns={'rul': 'anomaly'}, inplace=True)
33
34     return data, labels
35
36 # Select FD001 subdataset of NASA Turbofan Degradation dataset
37 dataset = NASATurbofanDegradation(subdataset="FD001")
38
39 # Split dataset using predefined dataset split
40 splitter = SklearnSplitter(splitter=PredefinedSplit(dataset.sklearn_predefined_split()))
41
42 # Preprocesses definition
43 preprocesses = [
44     CustomPreprocess(data_cleanup),
45     SklearnPreprocess(to_data=True, to_labels=False,
46         test_set=True, preprocess=MinMaxScaler()),
47     SavitzkyGolayFilter(to_data=True, to_labels=False,
48         test_set=True, window_length=7, polyorder=2),
49     RULImputation(threshold=125),
50     Binarizer(to_data=False, to_labels=[
51         'rul'], test_set=True, threshold=50),
52     CustomPreprocess(rename_labels)
53 ]
54
55 # Algorithms definition with default parameters
56 algorithms = [
57     PCAMahalanobis(),
58     OneClassSVM()
59 ]
60
61 metrics = [
62     Accuracy()
63 ]
64 # Pipeline with all steps
65 pipeline = Pipeline(dataset, splitter, preprocesses, algorithms, metrics)
66
67 # Experiment definition and launch
68 experiment = Experiment([pipeline], [])
69 experiment.launch()

```

3.1.2 Batteries Degradation

This experiment can be found in the file `batteries_degradation.py` and reproduces the results obtained in a paper to estimate the level of degradation of some types of lithium batteries. The dataset used is Batteries Degradation. This is not a time series, although it is somewhat similar since it measures different types of degradation in three types of batteries as they are gradually used. It is a local dataset, so it is necessary to pass the path in which it is located, and also the type of battery to be selected (LFP) and the test set, in this case 1. The splitter used is the Sklearn Predefined Split and it does not have any preprocessing since during the parsing of the local files to the Conmo format the data is already normalised. The algorithms used are the same as those used in the paper: Random Forest, Multilayer Perceptron and Convolutional Neural Network. In all cases the pre-trained models are used, so it is necessary to pass the path to the files as a parameter. The metric used is Root Mean Square Percentage Error.

```

1 from conmo import Experiment, Pipeline
2 from conmo.algorithms import PretrainedRandomForest, PretrainedCNN1D,
  ↳ PretrainedMultilayerPerceptron
3 from conmo.datasets import BatteriesDataset
4 from conmo.metrics import RMSPE
5 from conmo.splitters import SklearnSplitter
6 from sklearn.model_selection import PredefinedSplit
7
8 # Pipeline definition
9 # Change path to our local dataset files, specify chemistry of the batteries (LFP, NCA,
  ↳ NMC) and test set
10 dataset = BatteriesDataset('/path/to/batteries/dataset/', 'LFP', 1)
11 splitter = SklearnSplitter(splitter=PredefinedSplit(dataset.sklearn_predefined_split()))
12 preprocesses = None
13 # Changes the path to the files where the pre-trained models are stored (usually h5,
  ↳ h5py or joblib formats).
14 algorithms = [
15     PretrainedRandomForest(pretrained=True, path='/path/to/saved/model-RF.joblib'),
16     PretrainedMultilayerPerceptron(pretrained=True, input_len=128, path='/path/to/saved/
  ↳ model-MLP.h5'),
17     PretrainedCNN1D(pretrained=True, input_len=128, path='/path/to/saved/model-CNN.h5')
18 ]
19 metrics = [
20     RMSPE()
21 ]
22 pipeline = Pipeline(dataset, splitter, preprocesses, algorithms, metrics)
23
24
25 # Experiment definition and launch
26 experiment = Experiment([pipeline], [])
27 experiment.launch()

```

3.1.3 Server Machine Dataset with PCAMahalanobis

This experiment can be found in the file *omni_anomaly_smd.py*. The Server Machine Dataset used in this experiment has been obtained from the OmniAnomaly repository. In their [Github](#) you can find more information about the dataset as well as the implementation of other anomaly detection and time series data mining algorithms. The splitter used is the Sklearn Predefined Split and the preprocessing is the MinMaxScaler from Sklearn. The algorithms is PCA with Mahalanobis distance. Finally, the metric is the Accuracy.

```
1 from sklearn.preprocessing import MinMaxScaler
2
3 from conmo import Experiment, Pipeline
4 from conmo.algorithms import PCAMahalanobis
5 from conmo.datasets import ServerMachineDataset
6 from conmo.metrics import Accuracy
7 from conmo.preprocesses import SklearnPreprocess
8 from conmo.splitters import SklearnSplitter
9 from sklearn.model_selection import PredefinedSplit
10 from sklearn.preprocessing import MinMaxScaler
11
12 # Pipeline definition
13 dataset = ServerMachineDataset('1-01')
14 splitter = SklearnSplitter(splitter=PredefinedSplit(dataset.sklearn_predefined_split()))
15 preprocesses = [
16     SklearnPreprocess(to_data=True, to_labels=False,
17                       test_set=True, preprocess=MinMaxScaler()),
18 ]
19 algorithms = [
20     PCAMahalanobis()
21 ]
22 metrics = [
23     Accuracy()
24 ]
25 pipeline = Pipeline(dataset, splitter, preprocesses, algorithms, metrics)
26
27
28 # Experiment definition and launch
29 experiment = Experiment([pipeline], [])
30 experiment.launch()
```


API REFERENCE

4.1 API Reference

This is the API Reference documentation of the package, including modules, classes and functions.

4.1.1 conmo.experiments

This is the main submodule of the package and it is the responsible of create the intermediary directories of the experiment and take care of creating and executing the configured pipeline.

experiment.Experiment(pipelines, analytics)

experiment.Pipeline(dataset, splitter, ...)

conmo.experiment.Experiment

class conmo.experiment.**Experiment**(*pipelines: Iterable[Pipeline]*, *analytics: Iterable*,
 name='2022_07_01-10_50_50')

__init__(*pipelines: Iterable[Pipeline]*, *analytics: Iterable*, *name='2022_07_01-10_50_50'*)

generate_dirs() → Iterable[str]

 Generates directories both for the experiment itself and for the pipelines it contains.

Returns

pipes_dirs – Array containing the names of the directories of the pipes of the current experiment.

Return type

 Iterable[str]

launch()

 Launchs the current experiment.

Methods

<code>__init__(pipelines, analytics[, name])</code>	
<code>generate_dirs()</code>	Generates directories both for the experiment itself and for the pipelines it contains.
<code>launch()</code>	Launches the current experiment.

conmo.experiment.Pipeline

class conmo.experiment.Pipeline(*dataset: Dataset, splitter: Optional[Splitter], preprocesses: Optional[Iterable[Preprocess]], algorithms: Iterable[Algorithm], metrics: Iterable[Metric]*)

__init__(*dataset: Dataset, splitter: Optional[Splitter], preprocesses: Optional[Iterable[Preprocess]], algorithms: Iterable[Algorithm], metrics: Iterable[Metric]*) → None

generate_dirs(*pipe_dir: str*) → None

Auxiliary method to generate directories for each of the steps in the current pipeline.

Parameters

pipe_dir (*str*) – Name of the pipe directory.

run(*pipe_dir: str, pipe_num: int, pipes: int*) → None

Contains all the logic for the execution of a particular pipeline, creating intermediate directories for data passing and executing the relevant methods for each step.

Parameters

- **pipe_dir** (*str*) – Name of the current pipeline directory.
- **pipe_num** (*int*) – Index of the current pipelines.
- **pipes** (*int*) – Total number of pipelines in the current experiment.

Methods

<code>__init__(dataset, splitter, preprocesses, ...)</code>	
<code>generate_dirs(pipe_dir)</code>	Auxiliary method to generate directories for each of the steps in the current pipeline.
<code>run(pipe_dir, pipe_num, pipes)</code>	Contains all the logic for the execution of a particular pipeline, creating intermediate directories for data passing and executing the relevant methods for each step.

Attributes

ALGORITHMS_FOLDER

DATASET_FOLDER

METRICS_FOLDER

PREPROCESSES_FOLDER

SPLITTER_FOLDER

4.1.2 conmo.datasets

The *conmo.datasets* submodule takes care of downloading the dataset and parsing it to the Conmo's format.

<i>datasets.dataset.Dataset</i> (name)	Abstract base class for a Dataset.
<i>datasets.dataset.RemoteDataset</i> (url, ...)	Abstract base class for a RemoteDataset (downloadable).
<i>datasets.dataset.LocalDataset</i> (path)	Abstract base class for a LocalDataset (loadable).
<i>datasets.MarsScienceLaboratoryMission</i> (channel)	
<i>datasets.SoilMoistureActivePassiveSatellite</i> (channel)	
<i>datasets.ServerMachineDataset</i> (subdataset)	
<i>datasets.NASATurbofanDegradation</i> (subdataset)	
<i>datasets.BatteriesDataset</i> (path, chemistry, ...)	This is a dataset obtained from measurements of certain types of degradation of three types of batteries. Since it belongs to the local datasets, to launch any experiment with it, it must be stored on disk with the following directory structure: - DTW-Li-ion-Diagnosis - data : Data and labels for the three types of batteries are stored here. - mat: - LFP: - diagnosis: - V.mat - test: - V_references.mat - x_test_0.mat - x_test_1.mat - x_test_2.mat - x_test_3.mat - y_test.mat - NCA: - diagnosis - test - NMC: - The same as NCA and LFP - Q.mat.

conmo.datasets.dataset.Dataset

class conmo.datasets.dataset.Dataset(*name: str*)

Abstract base class for a Dataset.

This class is an abstract class from which other subclasses inherit and must not be instantiated directly.

__init__(*name: str*) → None

Main constructor of the class.

Parameters

name (*str*) – The name given to the dataset.

abstract dataset_files() → Iterable

Iterable of files included in the dataset.

abstract fetch(out_dir: str) → None

Fetch data to feed the pipeline.

Parameters

out_dir (str) – Directory where the dataset will be stored.

is_dataset_ready() → bool

Check if dataset has been already loaded/downloaded and parsed to package format.

show_start_message() → None

Show starting step info message.

Methods

<code>__init__(name)</code>	Main constructor of the class.
<code>dataset_files()</code>	Iterable of files included in the dataset.
<code>fetch(out_dir)</code>	Fetch data to feed the pipeline.
<code>is_dataset_ready()</code>	Check if dataset has been already loaded/downloaded and parsed to package format.
<code>show_start_message()</code>	Show starting step info message.

conmo.datasets.dataset.RemoteDataset

class conmo.datasets.dataset.RemoteDataset(url: str, file_format: str, checksum: str, checksum_format: str)

Abstract base class for a RemoteDataset (downloadable).

__init__(url: str, file_format: str, checksum: str, checksum_format: str) → None

Main constructor of the class.

Parameters

name (str) – The name given to the dataset.

check_checksum(response: object) → bool

Checks if the checksum of the downloaded file corresponds to the one provided in the class. For security e integrity issues. Currently only the md5 algorithm is integrated.

Parameters

response (Object) – Response object returned by the get method of the Requests library.

Return type

Boolean variable indicating whether the comparison of the hash with the checksum was successful or not.

abstract dataset_files() → Iterable

Iterable of files included in the dataset.

download(out_dir: str) → None

Download a Dataset from a remote URL.

extract_data(response: object, out_dir: str) → None

Extracts the contents of a compressed file in zip format.

Parameters

- **response** (*Object*) – Response object returned by the get method of the Requests library.
- **out_dir** (*str*) – Directory where the zip file will be unzipped.

abstract feed_pipeline(*out_dir: str*) → None

Copy selected data file to pipeline step folder.

fetch(*out_dir: str*) → None

Fetch data to feed the pipeline.

Parameters

out_dir (*str*) – Directory where the dataset will be stored.

is_dataset_ready() → bool

Check if dataset has been already loaded/downloaded and parsed to package format.

abstract parse_to_package(*raw_dir: str*) → None

Parse raw dataset to package format. Data and labels must be saved in parquet format. More information about parquet format: <https://parquet.apache.org/>

Parameters

raw_dir – Directory where the dataset was downloaded from its source.

show_start_message() → None

Show starting step info message.

Methods

<code>__init__(url, file_format, checksum, ...)</code>	Main constructor of the class.
<code>check_checksum(response)</code>	Checks if the checksum of the downloaded file corresponds to the one provided in the class.
<code>dataset_files()</code>	Iterable of files included in the dataset.
<code>download(out_dir)</code>	Download a Dataset from a remote URL.
<code>extract_data(response, out_dir)</code>	Extracts the contents of a compressed file in zip format.
<code>feed_pipeline(out_dir)</code>	Copy selected data file to pipeline step folder.
<code>fetch(out_dir)</code>	Fetch data to feed the pipeline.
<code>is_dataset_ready()</code>	Check if dataset has been already loaded/downloaded and parsed to package format.
<code>parse_to_package(raw_dir)</code>	Parse raw dataset to package format.
<code>show_start_message()</code>	Show starting step info message.

conmo.datasets.dataset.LocalDataset

class conmo.datasets.dataset.**LocalDataset**(*path: str*)

Abstract base class for a LocalDataset (loadable).

__init__(*path: str*) → None

Constructor of local dataset.

Parameters

path (*str*) – Absolute path to the folder where the dataset is located in your disk.

See the example of batteries_degradation.py.

abstract dataset_files() → Iterable

Iterable of files included in the dataset.

abstract feed_pipeline(*out_dir: str*) → None

Copy selected data file to pipeline step folder.

Parameters

out_dir – Directory where the dataset was originally stored.

fetch(*out_dir: str*) → None

Fetch data to feed the pipeline.

Parameters

out_dir (*str*) – Directory where the dataset will be stored.

is_dataset_ready() → bool

Check if dataset has been already loaded/downloaded and parsed to package format.

abstract load() → None

Parse raw dataset to package format. Data and labels must be saved in parquet format. More information about parquet format: <https://parquet.apache.org/>

show_start_message() → None

Show starting step info message.

Methods

<code>__init__(path)</code>	Constructor of local dataset.
<code>dataset_files()</code>	Iterable of files included in the dataset.
<code>feed_pipeline(out_dir)</code>	Copy selected data file to pipeline step folder.
<code>fetch(out_dir)</code>	Fetch data to feed the pipeline.
<code>is_dataset_ready()</code>	Check if dataset has been already loaded/downloaded and parsed to package format.
<code>load()</code>	Parse raw dataset to package format.
<code>show_start_message()</code>	Show starting step info message.

conmo.datasets.MarsScienceLaboratoryMission

class conmo.datasets.MarsScienceLaboratoryMission(*channel: str*)

__init__(*channel: str*) → None

Main constructor of the class.

Parameters

name (*str*) – The name given to the dataset.

check_checksum(*response: object*) → bool

Checks if the checksum of the downloaded file corresponds to the one provided in the class. For security e integrity issues. Currently only the md5 algorithm is integrated.

Parameters

response (*Object*) – Response object returned by the get method of the Requests library.

Return type

Boolean variable indicating whether the comparison of the hash with the checksum was successful or not.

check_checksum_lbl(*response: object, checksum: str*) → bool

Checks if the checksum of the downloaded file corresponds to the one provided in the class. For security e integrity issues. Currently only the md5 algorithm is integrated. Since in the MLS

dataset the labels are obtained from a different file, it's necessary to use another method to pass the checksum of that file.

Parameters

- **response** (*Object*) – Response object returned by the get method of the Requests library.
- **checksum** (*str*) – String containing the labels checksum.

Returns

Boolean variable indicating whether the comparison of the hash with the checksum was successful or not.

Return type

bool

dataset_files() → Iterable

Iterable of files included in the dataset.

download(out_dir: str) → None

Download a Dataset from a remote URL.

download_anomalies_file(raw_dir: str) → Iterable[DataFrame]

Method in charge of downloading and parsing the MLS dataset labels files. This is because the tags are located at a different URL than the data.

Parameters

raw_dir (*str*) – Directory where the unparsed data of SMAP dataset is stored until it's processed.

Returns

labeled_anomalies – Anomalous intervals in the MSL dataset.

Return type

Pandas Dataframe

extract_data(response: object, out_dir: str) → None

Extracts the contents of a compressed file in zip format.

Parameters

- **response** (*Object*) – Response object returned by the get method of the Requests library.
- **out_dir** (*str*) – Directory where the zip file will be unzipped.

feed_pipeline(out_dir: str) → None

Copy selected data file to pipeline step folder.

fetch(out_dir: str) → None

Fetch data to feed the pipeline.

Parameters

out_dir (*str*) – Directory where the dataset will be stored.

is_dataset_ready() → bool

Check if dataset has been already loaded/downloaded and parsed to package format.

parse_to_package(raw_dir: str) → None

Parse raw dataset to package format. Data and labels must be saved in parquet format. More information about parquet format: <https://parquet.apache.org/>

Parameters

raw_dir – Directory where the dataset was downloaded from its source.

represent_anomalies(labels: Iterable[DataFrame], channel: str, labeled_anomalies: Iterable[DataFrame]) → Iterable[DataFrame]

Represent anomalies in the label's dataset following the anomalous intervals of 'labeled_anomalies.csv'

Parameters

- **labels** (*Pandas Dataframe*) – Dataframe with the shape of the labels but filled with zeros.
- **channel** (*str*) – Channel identifier (subdataset).
- **labeled_anomalies** (*Pandas Dataframe*) – Anomalous intervals in the MSL dataset.

Returns

labels – Labels dataset correctly filled.

Return type

Pandas Dataframe

show_start_message() → None

Show starting step info message.

sklearn_predefined_split() → Iterable[int]

Generates array of indexes of same length as sequences to be used with 'PredefinedSplit' MSL dataset has only 2 sequences: one for train and another for test.

Returns

List with the index for each sequence of the dataset.

Return type

array

Methods

<code>__init__(channel)</code>	Main constructor of the class.
<code>check_checksum(response)</code>	Checks if the checksum of the downloaded file corresponds to the one provided in the class.
<code>check_checksum_lbl(response, checksum)</code>	Checks if the checksum of the downloaded file corresponds to the one provided in the class.
<code>dataset_files()</code>	Iterable of files included in the dataset.
<code>download(out_dir)</code>	Download a Dataset from a remote URL.
<code>download_anomalies_file(raw_dir)</code>	Method in charge of downloading and parsing the MLS dataset labels files.
<code>extract_data(response, out_dir)</code>	Extracts the contents of a compressed file in zip format.
<code>feed_pipeline(out_dir)</code>	Copy selected data file to pipeline step folder.
<code>fetch(out_dir)</code>	Fetch data to feed the pipeline.
<code>is_dataset_ready()</code>	Check if dataset has been already loaded/downloaded and parsed to package format.
<code>parse_to_package(raw_dir)</code>	Parse raw dataset to package format.
<code>represent_anomalies(labels, channel, ...)</code>	Represent anomalies in the label's dataset following the anomalous intervals of 'labeled_anomalies.csv'
<code>show_start_message()</code>	Show starting step info message.
<code>sklearn_predefined_split()</code>	Generates array of indexes of same length as sequences to be used with 'PredefinedSplit' MSL dataset has only 2 sequences: one for train and another for test.

Attributes

CHANNELS
CHECKSUM
CHECKSUM_FORMAT
FILE_FORMAT
LABEL
SEQUENCE_COLUMN
TIME_COLUMN
URL
VARIABLES

conmo.datasets.SoilMoistureActivePassiveSatellite

class conmo.datasets.**SoilMoistureActivePassiveSatellite**(*channel: str*)

__init__(*channel: str*) → None

Main constructor of the class.

Parameters

name (*str*) – The name given to the dataset.

check_checksum(*response: object*) → bool

Checks if the checksum of the downloaded file corresponds to the one provided in the class. For security e integrity issues. Currently only the md5 algorithm is integrated.

Parameters

response (*Object*) – Response object returned by the get method of the Requests library.

Return type

Boolean variable indicating whether the comparison of the hash with the checksum was successful or not.

check_checksum_lbl(*response: object, checksum: str*) → bool

Checks if the checksum of the downloaded file corresponds to the one provided in the class. For security e integrity issues. Currently only the md5 algorithm is integrated. Since in the SMAP dataset the labels are obtained from a different file, it's necessary to use another method to pass the checksum of that file.

Parameters

- **response** (*object*) – Response object returned by the get method of the Requests library.
- **checksum** (*str*) – String containing the labels' checksum.

Returns

Boolean variable indicating whether the comparison of the hash with the checksum was successful or not.

Return type

bool

dataset_files() → Iterable

Iterable of files included in the dataset.

download(*out_dir: str*) → None

Download a Dataset from a remote URL.

download_anomalies_file(*raw_dir: str*) → Iterable[DataFrame]

Method in charge of downloading and parsing the SMAP dataset labels files. This is because the tags are located at a different URL than the data.

Parameters**raw_dir** (*str*) – Directory where the unparsed data of SMAP dataset is stored until it's processed.**Returns****labeled_anomalies** – Anomalous intervals in the SMAP dataset.**Return type**

Pandas Dataframe

extract_data(*response: object, out_dir: str*) → None

Extracts the contents of a compressed file in zip format.

Parameters

- **response** (*Object*) – Response object returned by the get method of the Requests library.
- **out_dir** (*str*) – Directory where the zip file will be unzipped.

feed_pipeline(*out_dir: str*) → None

Copy selected data file to pipeline step folder.

fetch(*out_dir: str*) → None

Fetch data to feed the pipeline.

Parameters**out_dir** (*str*) – Directory where the dataset will be stored.**is_dataset_ready**() → bool

Check if dataset has been already loaded/downloaded and parsed to package format.

parse_to_package(*raw_dir: str*) → NoneParse raw dataset to package format. Data and labels must be saved in parquet format. More information about parquet format: <https://parquet.apache.org/>**Parameters****raw_dir** – Directory where the dataset was downloaded from its source.**represent_anomalies**(*labels: Iterable[DataFrame], channel: str, labeled_anomalies: Iterable[DataFrame]*) → Iterable[DataFrame]

Represent anomalies in the label's dataset following the anomalous intervals of 'labeled_anomalies.csv'

Parameters

- **labels** (*Pandas Dataframe*) – Dataframe with the shape of the labels but filled with zeros.
- **channel** (*str*) – Channel identifier (subdataset)
- **labeled_anomalies** (*Pandas Dataframe*) – Anomalous intervals in the SMAP dataset.

Returns**labels** – Labels dataset correctly filled.

Return type

Pandas Dataframe

show_start_message() → None

Show starting step info message.

sklearn_predefined_split() → Iterable[int]

Generates array of indexes of same length as sequences to be used with 'PredefinedSplit' SMAP dataset has only 2 sequences: one for train and another for test.

Returns

List with the index for each sequence of the dataset.

Return type

array

Methods

<code>__init__(channel)</code>	Main constructor of the class.
<code>check_checksum(response)</code>	Checks if the checksum of the downloaded file corresponds to the one provided in the class.
<code>check_checksum_lbl(response, checksum)</code>	Checks if the checksum of the downloaded file corresponds to the one provided in the class.
<code>dataset_files()</code>	Iterable of files included in the dataset.
<code>download(out_dir)</code>	Download a Dataset from a remote URL.
<code>download_anomalies_file(raw_dir)</code>	Method in charge of downloading and parsing the SMAP dataset labels files.
<code>extract_data(response, out_dir)</code>	Extracts the contents of a compressed file in zip format.
<code>feed_pipeline(out_dir)</code>	Copy selected data file to pipeline step folder.
<code>fetch(out_dir)</code>	Fetch data to feed the pipeline.
<code>is_dataset_ready()</code>	Check if dataset has been already loaded/downloaded and parsed to package format.
<code>parse_to_package(raw_dir)</code>	Parse raw dataset to package format.
<code>represent_anomalies(labels, channel, ...)</code>	Represent anomalies in the label's dataset following the anomalous intervals of 'labeled_anomalies.csv'
<code>show_start_message()</code>	Show starting step info message.
<code>sklearn_predefined_split()</code>	Generates array of indexes of same length as sequences to be used with 'PredefinedSplit' SMAP dataset has only 2 sequences: one for train and another for test.

Attributes

CHANNELS
CHECKSUM
CHECKSUM_FORMAT
FILE_FORMAT
LABEL
SEQUENCE_COLUMN
TIME_COLUMN
URL
VARIABLES

conmo.datasets.ServerMachineDataset

class conmo.datasets.**ServerMachineDataset**(*subdataset: str*)

__init__(*subdataset: str*) → None

Main constructor of the class.

Parameters

name (*str*) – The name given to the dataset.

check_checksum(*response: object*) → bool

Checks if the checksum of the downloaded file corresponds to the one provided in the class. For security e integrity issues. Currently only the md5 algorithm is integrated.

Parameters

response (*Object*) – Response object returned by the get method of the Requests library.

Return type

Boolean variable indicating whether the comparison of the hash with the checksum was successful or not.

dataset_files() → Iterable

Iterable of files included in the dataset.

download(*out_dir: str*) → None

Download a Dataset from a remote URL.

extract_data(*response: object, out_dir: str*) → None

Extracts the contents of a compressed file in zip format.

Parameters

- **response** (*Object*) – Response object returned by the get method of the Requests library.
- **out_dir** (*str*) – Directory were the zip file will be unzipped.

feed_pipeline(*out_dir: str*) → None

Copy selected data file to pipeline step folder.

fetch(*out_dir: str*) → None

Fetch data to feed the pipeline.

Parameters

out_dir (*str*) – Directory where the dataset will be stored.

is_dataset_ready() → bool

Check if dataset has been already loaded/downloaded and parsed to package format.

parse_to_package(*raw_dir: str*) → None

Parse raw dataset to package format. Data and labels must be saved in parquet format. More information about parquet format: <https://parquet.apache.org/>

Parameters

raw_dir – Directory where the dataset was downloaded from its source.

show_start_message() → None

Show starting step info message.

sklearn_predefined_split() → Iterable[int]

Generates array of indexes of same length as sequences to be used with 'PredefinedSplit' SMD dataset has only 2 sequences: one for train and another for test.

Returns

List with the index for each sequence of the dataset.

Return type

array

Methods

<code>__init__(subdataset)</code>	Main constructor of the class.
<code>check_checksum(response)</code>	Checks if the checksum of the downloaded file corresponds to the one provided in the class.
<code>dataset_files()</code>	Iterable of files included in the dataset.
<code>download(out_dir)</code>	Download a Dataset from a remote URL.
<code>extract_data(response, out_dir)</code>	Extracts the contents of a compressed file in zip format.
<code>feed_pipeline(out_dir)</code>	Copy selected data file to pipeline step folder.
<code>fetch(out_dir)</code>	Fetch data to feed the pipeline.
<code>is_dataset_ready()</code>	Check if dataset has been already loaded/downloaded and parsed to package format.
<code>parse_to_package(raw_dir)</code>	Parse raw dataset to package format.
<code>show_start_message()</code>	Show starting step info message.
<code>sklearn_predefined_split()</code>	Generates array of indexes of same length as sequences to be used with 'PredefinedSplit' SMD dataset has only 2 sequences: one for train and another for test.

Attributes

CHECKSUM
CHECKSUM_FORMAT
FILE_FORMAT
LABEL
SEQUENCE_COLUMN
SUBDATASETS
TIME_COLUMN
URL
VARIABLES

conmo.datasets.NASATurbofanDegradation

class conmo.datasets.NASATurbofanDegradation(*subdataset: str*)

__init__(*subdataset: str*) → None

Main constructor of the class.

Parameters

name (*str*) – The name given to the dataset.

check_checksum(*response: object*) → bool

Checks if the checksum of the downloaded file corresponds to the one provided in the class. For security e integrity issues. Currently only the md5 algorithm is integrated.

Parameters

response (*Object*) – Response object returned by the get method of the Requests library.

Return type

Boolean variable indicating whether the comparison of the hash with the checksum was successful or not.

dataset_files() → Iterable

Iterable of files included in the dataset.

download(*out_dir: str*) → None

Download a Dataset from a remote URL.

extract_data(*response: object, out_dir: str*) → None

Extracts the contents of a compressed file in zip format.

Parameters

- **response** (*Object*) – Response object returned by the get method of the Requests library.
- **out_dir** (*str*) – Directory were the zip file will be unzipped.

feed_pipeline(*out_dir: str*) → None

Copy selected data file to pipeline step folder.

fetch(*out_dir: str*) → None

Fetch data to feed the pipeline.

Parameters

out_dir (*str*) – Directory where the dataset will be stored.

is_dataset_ready() → bool

Check if dataset has been already loaded/downloaded and parsed to package format.

parse_to_package(*raw_dir: str*) → None

Parse raw dataset to package format. Data and labels must be saved in parquet format. More information about parquet format: <https://parquet.apache.org/>

Parameters

raw_dir – Directory where the dataset was downloaded from its source.

show_start_message() → None

Show starting step info message.

sklearn_predefined_split() → Iterable[int]

Generates array of indexes of same length as sequences to be used with 'PredefinedSplit'

Returns

List with the index for each sequence of the dataset.

Return type

array

Methods

<code>__init__(subdataset)</code>	Main constructor of the class.
<code>check_checksum(response)</code>	Checks if the checksum of the downloaded file corresponds to the one provided in the class.
<code>dataset_files()</code>	Iterable of files included in the dataset.
<code>download(out_dir)</code>	Download a Dataset from a remote URL.
<code>extract_data(response, out_dir)</code>	Extracts the contents of a compressed file in zip format.
<code>feed_pipeline(out_dir)</code>	Copy selected data file to pipeline step folder.
<code>fetch(out_dir)</code>	Fetch data to feed the pipeline.
<code>is_dataset_ready()</code>	Check if dataset has been already loaded/downloaded and parsed to package format.
<code>parse_to_package(raw_dir)</code>	Parse raw dataset to package format.
<code>show_start_message()</code>	Show starting step info message.
<code>sklearn_predefined_split()</code>	Generates array of indexes of same length as sequences to be used with 'PredefinedSplit'

Attributes

CHECKSUM
CHECKSUM_FORMAT
FILE_FORMAT
LABEL
SEQUENCE_COLUMN
SUBDATASETS
TIME_COLUMN
URL
VARIABLES

conmo.datasets.BatteriesDataset

class conmo.datasets.**BatteriesDataset**(*path: str, chemistry: str, test_set: int*)

This is a dataset obtained from measurements of certain types of degradation of three types of batteries. Since it belongs to the local datasets, to launch any experiment with it, it must be stored on disk with the following directory structure: - DTW-Li-ion-Diagnosis

- **data**
[Data and labels for the three types of batteries] are stored here.
- **mat:**
 - **LFP:**
 - * **diagnosis:**
 - V.mat
 - * **test:**
 - V_references.mat
 - x_test_0.mat
 - x_test_1.mat
 - x_test_2.mat
 - x_test_3.mat
 - y_test.mat
 - **NCA:**
 - * diagnosis
 - * test
 - **NMC:**

* The same as NCA and LFP

– Q.mat

IC(*u*: ~numpy.ndarray, *q*: ~numpy.ndarray, *ui_step*: float = 0.0005, *minV*: float = 3.2, *maxV*: float = 3.5) -> (<class 'numpy.ndarray'>, <class 'numpy.ndarray'>)

Get the ICA data for a given voltage curve

Parameters

- **u** (numpy array) – Voltage curve.
- **q** (numpy array) – Capacity curve.
- **ui_step** (float) – Step of interpolation.
- **minV** (float) – Minimum voltage of the IC curve.
- **maxV** (float) – Maximum voltage of the IC curve.

Returns

ui, dqi – Interpolated voltage and derivative of capacity

Return type

numpy arrays

__init__(*path*: str, *chemistry*: str, *test_set*: int) → None

Constructor of local dataset.

Parameters

path (str) – Absolute path to the folder where the dataset is located in your disk. See the example of batteries_degradation.py.

convert_to_input_data(*ui_new*: list, *Q*: list, *size*: int, *material*: int) → ndarray

Converts the voltage values of the real cells to the input data for the neural network

Parameters

- **ui_new** (array) – Voltage values of the cell at each cycle in percentage.
- **Q** (array) – Capacity percentages from 0 to 100 from the simulated dataset.
- **size** (int) – The length of the curves.
- **material** (str) – Chemistry of the cell.

Returns

x_test – The input data for the neural network.

Return type

array

dataset_files() → Iterable

Iterable of files included in the dataset.

feed_pipeline(*out_dir*: str) → None

Copy selected data file to pipeline step folder.

Parameters

out_dir – Directory where the dataset was originally stored.

fetch(*out_dir: str*) → None

Fetch data to feed the pipeline.

Parameters

out_dir (*str*) – Directory where the dataset will be stored.

get_minmaxV(*material: ~numpy.ndarray*) -> (<class 'int'>, <class 'int'>, <class 'str'>)

Returns the range voltage in which to study the IC curves

Parameters

material (*numpy array*) – Chemistry to study.

Returns

min_v, max_v, path – Min and max voltage values and path where data is located,

Return type

numpy arrays, str

is_dataset_ready() → bool

Check if dataset has been already loaded/downloaded and parsed to package format.

load() → None

Parse dataset train/test data to match Conmo's standard.

normalise_data(*data: ndarray, min_val: float, max_val: float, low: int = 0, high: int = 1*) → float

Normalises the data to the range [low, high]

Parameters

- **data** (*numpy array*) – Data to normalise.
- **min** (*float*) – Minimum value of data.
- **max** (*float*) – Maximum value of data.
- **low** (*float*) – Minimum value of the range.
- **high** (*float*) – Maximum value of the range.

Returns

normalised_data – normalised data

Return type

float

reduce_size(*ui: ndarray, dqi: ndarray, size: int*) → ndarray

Reduces the length of the IC data to a given size

Parameters

- **ui** (*numpy array*) – Voltage curve.
- **dqi** (*numpy array*) – Derivative of capacity (IC).
- **size** (*int*) – Size at which to reduce the IC data.

Returns

Reduced IC.

Return type

numpy array

show_start_message() → None

Show starting step info message.

sklearn_predefined_split() → Iterable[int]

Generates array of indexes of same length as sequences to be used with 'PredefinedSplit'

Return type

array, list with the index for each sequence of the dataset.

Methods

<i>IC</i> (u, q[, ui_step, minV, maxV])	Get the ICA data for a given voltage curve
<i>__init__</i> (path, chemistry, test_set)	Constructor of local dataset.
<i>convert_to_input_data</i> (ui_new, Q, size, material)	Converts the voltage values of the real cells to the input data for the neural network
<i>dataset_files</i> ()	Iterable of files included in the dataset.
<i>feed_pipeline</i> (out_dir)	Copy selected data file to pipeline step folder.
<i>fetch</i> (out_dir)	Fetch data to feed the pipeline.
<i>get_minmaxV</i> (material)	Returns the range voltage in which to study the IC curves
<i>is_dataset_ready</i> ()	Check if dataset has been already loaded/downloaded and parsed to package format.
<i>load</i> ()	Parse dataset train/test data to match Conmo's standard.
<i>normalise_data</i> (data, min_val, max_val[, ...])	Normalises the data to the range [low, high]
<i>reduce_size</i> (ui, dqi, size)	Reduces the length of the IC data to a given size
<i>show_start_message</i> ()	Show starting step info message.
<i>sklearn_predefined_split</i> ()	Generates array of indexes of same length as sequences to be used with 'PredefinedSplit'

Attributes

CHEMISTRY_LIST
MAX_V
MAX_V_LFP
MAX_V_NCA
MAX_V_NMC
MIN_V
MIN_V_LFP
MIN_V_NCA
MIN_V_NMC
SIZE
UI_STEP

4.1.3 conmo.splitters

Once the dataset has been loaded, it is necessary to separate the training and test parts. The *conmo.splitters* submodule permits generate new splitters or use predefined ones from the Scikit-Learn library.

splitters.splitter.Splitter()

splitters.SklearnSplitter(splitter[, groups])

conmo.splitters.splitter.Splitter

class conmo.splitters.splitter.Splitter

 __init__()

already_split(df: DataFrame) → bool

 Checks if the dataset was already splitted.

Parameters

df (Pandas DataFrame) – Input dataset.

Returns

 True in case the dataset was already splitted, False otherwise.

Return type

 bool

Raises

RuntimeError – If the dataset isn't splitted and doesn't follow Conmo's format.

load_input(*in_dir: str*) -> (<class 'pandas.core.frame.DataFrame'>, <class 'pandas.core.frame.DataFrame'>)

Read parquet data and labels files of the chosen dataset.

Parameters

in_dir (*str*) – Input directory where the files are located.

Returns

- **data** (*Pandas Dataframe*) – Loaded data file.
- **labels** (*Pandas Dataframe*) – Loaded labels file.

Raises

If data and labels have different sequences values. –

save_output(*out_dir: str, data: DataFrame, labels: DataFrame*) → None

Save splitted dataset to parquet format.

Parameters

- **out_dir** (*str*) – Output directory where the results will be saved.
- **data** (*Pandas Dataframe*) – Splitted data.
- **labels** (*Pandas Dataframe*) – Splitted labels.

show_start_message()

Simple method to print on the terminal the name of the selected splitter.

abstract split(*in_dir: str, out_dir: str*) → None

Performs the split to both data and labels of the dataset.

Parameters

- **in_dir** (*str*) – Input directory of the before step.
- **out_dir** (*str*) – Output directory where te split data will be stored.

Methods

<code>__init__()</code>	
<code>already splitted(df)</code>	Checks if the dataset was already splitted.
<code>load_input(in_dir)</code>	Read parquet data and labels files of the chosen dataset.
<code>save_output(out_dir, data, labels)</code>	Save splitted dataset to parquet format.
<code>show_start_message()</code>	Simple method to print on the terminal the name of the selected splitter.
<code>split(in_dir, out_dir)</code>	Performs the split to both data and labels of the dataset.

conmo.splitters.SklearnSplitter

```
class conmo.splitters.SklearnSplitter(splitter: Union[GroupKFold, GroupShuffleSplit, KFold,
    LeaveOneGroupOut, LeavePGroupsOut, LeaveOneOut,
    LeavePOut, PredefinedSplit, RepeatedKFold,
    RepeatedStratifiedKFold, ShuffleSplit, StratifiedKFold,
    StratifiedShuffleSplit, TimeSeriesSplit], groups:
    Optional[Iterable[int]] = None)
```

```
__init__(splitter: Union[GroupKFold, GroupShuffleSplit, KFold, LeaveOneGroupOut,
    LeavePGroupsOut, LeaveOneOut, LeavePOut, PredefinedSplit, RepeatedKFold,
    RepeatedStratifiedKFold, ShuffleSplit, StratifiedKFold, StratifiedShuffleSplit,
    TimeSeriesSplit], groups: Optional[Iterable[int]] = None) → None
```

```
already_split(df: DataFrame) → bool
```

Checks if the dataset was already splitted.

Parameters

df (Pandas DataFrame) – Input dataset.

Returns

True in case the dataset was already splitted, False otherwise.

Return type

bool

Raises

RuntimeError – If the dataset isn't splitted and doesn't follow Conmo's format.

```
extract_fold(df: ~pandas.core.frame.DataFrame, sequences: ~numpy.ndarray, fold: int,
    train_idx: ~numpy.ndarray, test_idx: ~numpy.ndarray) -> (<class
    'numpy.ndarray'>, <class 'numpy.ndarray'>)
```

```
load_input(in_dir: str) -> (<class 'pandas.core.frame.DataFrame'>, <class
    'pandas.core.frame.DataFrame'>)
```

Read parquet data and labels files of the chosen dataset.

Parameters

in_dir (str) – Input directory where the files are located.

Returns

- **data** (Pandas DataFrame) – Loaded data file.
- **labels** (Pandas DataFrame) – Loaded labels file.

Raises

If data and labels have different sequences values. –

```
save_output(out_dir: str, data: DataFrame, labels: DataFrame) → None
```

Save splitted dataset to parquet format.

Parameters

- **out_dir** (str) – Output directory where the results will be saved.
- **data** (Pandas DataFrame) – Splitted data.
- **labels** (Pandas DataFrame) – Splitted labels.

show_start_message()

Simple method to print on the terminal the name of the selected splitter.

split(*in_dir*: str, *out_dir*: str) → None

Performs the split to both data and labels of the dataset.

Parameters

- **in_dir** (str) – Input directory of the before step.
- **out_dir** (str) – Output directory where te split data will be stored.

to_dataframe(*df*: DataFrame, *data*: ndarray, *index*: ndarray) → DataFrame**Methods**

<code>__init__(splitter[, groups])</code>	
<code>already_splittd(df)</code>	Checks if the dataset was already splitted.
<code>extract_fold(df, sequences, fold, train_idx, ...)</code>	
<code>load_input(in_dir)</code>	Read parquet data and labels files of the chosen dataset.
<code>save_output(out_dir, data, labels)</code>	Save splitted dataset to parquet format.
<code>show_start_message()</code>	Simple method to print on the terminal the name of the selected splitter.
<code>split(in_dir, out_dir)</code>	Performs the split to both data and labels of the dataset.
<code>to_dataframe(df, data, index)</code>	

4.1.4 conmo.preprocesses

The aim of the `conmo.preprocesses` submodule is to apply a series of transformations to the data set before it is used as input to the algorithms. Several types of preprocesses implemented are usually used in time series anomaly detection problems.

<code>preprocesses.preprocess.Preprocess()</code>	Abstract base class for a Preprocess.
<code>preprocesses.preprocess.ExtendedPreprocess(...)</code>	Specific class to implement preprocessing which consists of applying certain transformations on some columns of the dataset.
<code>preprocesses.Binarizer(to_data, to_labels, ...)</code>	
<code>preprocesses.CustomPreprocess(fn)</code>	Core class used to implement self-created preprocess.
<code>preprocesses.RULImputation(threshold)</code>	
<code>preprocesses.SavitzkyGolayFilter(to_data, ...)</code>	
<code>preprocesses.SklearnPreprocess(to_data, ...)</code>	Class used to wrap existing preprocess in the Scikit-Learn library.

conmo.preprocesses.preprocess.Preprocess**class** conmo.preprocesses.preprocess.Preprocess

Abstract base class for a Preprocess.

This class is an abstract class from which other subclasses inherit and must not be instantiated directly.

__init__()**abstract apply**(*in_dir: str, out_dir: str*) → None

Applies the preprocess to the given dataset.

Parameters

- **in_dir** (*str*) – Input directory where the files are located. Usually, this is the output directory of the splitter step.
- **out_dir** (*str*) – Output directory where the files will be saved.

load_input(*in_dir: str*) -> (<class 'pandas.core.frame.DataFrame'>, <class 'pandas.core.frame.DataFrame'>)

Read parquet data and labels files of the chosen dataset before it's split.

Parameters**in_dir** (*str*) – Input directory where the files are located.**Returns**

- **data** (*Pandas Dataframe*) – Loaded data file.
- **labels** (*Pandas Dataframe*) – Loaded labels file.

save_output(*out_dir: str, data: DataFrame, labels: DataFrame*) → None

Save preprocessed dataset to parquet format.

Parameters

- **out_dir** (*str*) – Output directory where the results will be saved.
- **data** (*Pandas Dataframe*) – Preprocessed data.
- **labels** (*Pandas Dataframe*) – Preprocessed labels.

show_start_message() → None

Simple method to print on the terminal the name of the selected splitter.

Methods

<code>__init__()</code>	
<code>apply(in_dir, out_dir)</code>	Applies the preprocess to the given dataset.
<code>load_input(in_dir)</code>	Read parquet data and labels files of the chosen dataset before it's split.
<code>save_output(out_dir, data, labels)</code>	Save preprocessed dataset to parquet format.
<code>show_start_message()</code>	Simple method to print on the terminal the name of the selected splitter.

conmo.preprocesses.preprocess.ExtendedPreprocess

```
class conmo.preprocesses.preprocess.ExtendedPreprocess(to_data: Union[bool, Iterable[str]],
                                                    to_labels: Union[bool, Iterable[str]],
                                                    test_set: bool)
```

Specific class to implement preprocessing which consists of applying certain transformations on some columns of the dataset. The preprocessing that inherit from this class have in the constructor to_data, to_labels and test_set to indicate the columns on which to apply the DATA and LABELS preprocessing respectively, and if the TEST ones are included or not.

```
__init__(to_data: Union[bool, Iterable[str]], to_labels: Union[bool, Iterable[str]], test_set:
          bool) → None
```

```
apply(in_dir: str, out_dir: str) → None
```

Applies the preprocess to the given dataset.

Parameters

- **in_dir** (str) – Input directory where the files are located. Usually, this is the output directory of the splitter step.
- **out_dir** (str) – Output directory where the files will be saved.

```
extract_columns(df: DataFrame, columns: Union[bool, Iterable[str]]) → Iterable[str]
```

Returns a list containing all the column's name of the data.

Parameters

- **df** (Pandas DataFrame) – Dataframe containing the data.
- **columns** (Union[bool, Iterable[str]]) – Bool value if the dataframe has columns or the list of columns.

Returns

columns – List containing the names of the dataframe's columns.

Return type

Iterable[str]

```
load_input(in_dir: str) -> (<class 'pandas.core.frame.DataFrame'>, <class
                             'pandas.core.frame.DataFrame'>)
```

Read parquet data and labels files of the chosen dataset before it's split.

Parameters

in_dir (str) – Input directory where the files are located.

Returns

- **data** (Pandas DataFrame) – Loaded data file.
- **labels** (Pandas DataFrame) – Loaded labels file.

```
save_output(out_dir: str, data: DataFrame, labels: DataFrame) → None
```

Save preprocessed dataset to parquet format.

Parameters

- **out_dir** (str) – Output directory where the results will be saved.
- **data** (Pandas DataFrame) – Preprocessed data.
- **labels** (Pandas DataFrame) – Preprocessed labels.

show_start_message() → None

Simple method to print on the terminal the name of the selected splitter.

abstract transform(*df: DataFrame, columns: Iterable[str]*) → DataFrame

Performs the preprocess over the dataframe with the given columns.

Parameters

- **df** (*Pandas Dataframe*) – Dataframe containing the data or the labels of the dataset.
- **columns** (*Iterable[str]*) – List of columns that will be used in the preprocess. Also the columns of the final dataframe.

Returns

Dataframe preprocessed.

Return type

Pandas Dataframe

Methods

<code>__init__(to_data, to_labels, test_set)</code>	
<code>apply(in_dir, out_dir)</code>	Applies the preprocess to the given dataset.
<code>extract_columns(df, columns)</code>	Returns a list containig all the column's name of the data.
<code>load_input(in_dir)</code>	Read parquet data and labels files of the chosen dataset before it's split.
<code>save_output(out_dir, data, labels)</code>	Save preprocessed dataset to parquet format.
<code>show_start_message()</code>	Simple method to print on the terminal the name of the selected splitter.
<code>transform(df, columns)</code>	Performs the preprocess over the dataframe with the given columns.

conmo.preprocesses.Binarizer

class conmo.preprocesses.**Binarizer**(*to_data: Union[bool, Iterable[str]], to_labels: Union[bool, Iterable[str]], test_set: bool, threshold: int*)

__init__(*to_data: Union[bool, Iterable[str]], to_labels: Union[bool, Iterable[str]], test_set: bool, threshold: int*) → None

apply(*in_dir: str, out_dir: str*) → None

Applies the preprocess to the given dataset.

Parameters

- **in_dir** (*str*) – Input directory where the files are located. Usually, this is the output directory of the splitter step.
- **out_dir** (*str*) – Output directory where the files will be saved.

extract_columns(*df: DataFrame, columns: Union[bool, Iterable[str]]*) → Iterable[str]

Returns a list containig all the column's name of the data.

Parameters

- **df** (*Pandas Dataframe*) – Dataframe containing the data.
- **columns** (*Union[bool, Iterable[str]]*) – Bool value if the dataframe has columns or the list of columns.

Returns

columns – List containing the names of the dataframe's columns.

Return type

`Iterable[str]`

load_input(*in_dir: str*) -> (`<class 'pandas.core.frame.DataFrame'>`, `<class 'pandas.core.frame.DataFrame'>`)

Read parquet data and labels files of the chosen dataset before it's split.

Parameters

in_dir (*str*) – Input directory where the files are located.

Returns

- **data** (*Pandas Dataframe*) – Loaded data file.
- **labels** (*Pandas Dataframe*) – Loaded labels file.

save_output(*out_dir: str, data: DataFrame, labels: DataFrame*) → None

Save preprocessed dataset to parquet format.

Parameters

- **out_dir** (*str*) – Output directory where the results will be saved.
- **data** (*Pandas Dataframe*) – Preprocessed data.
- **labels** (*Pandas Dataframe*) – Preprocessed labels.

show_start_message() → None

Simple method to print on the terminal the name of the selected splitter.

transform(*df: DataFrame, columns: Iterable[str]*) → DataFrame

Performs the preprocess over the dataframe with the given columns.

Parameters

- **df** (*Pandas Dataframe*) – Dataframe containing the data or the labels of the dataset.
- **columns** (*Iterable[str]*) – List of columns that will be used in the preprocess. Also the columns of the final dataframe.

Returns

Dataframe preprocessed.

Return type

Pandas Dataframe

Methods

<code>__init__(to_data, to_labels, test_set, threshold)</code>	
<code>apply(in_dir, out_dir)</code>	Applies the preprocess to the given dataset.
<code>extract_columns(df, columns)</code>	Returns a list containig all the column's name of the data.
<code>load_input(in_dir)</code>	Read parquet data and labels files of the chosen dataset before it's split.
<code>save_output(out_dir, data, labels)</code>	Save preprocessed dataset to parquet format.
<code>show_start_message()</code>	Simple method to print on the terminal the name of the selected splitter.
<code>transform(df, columns)</code>	Performs the preprocess over the dataframe with the given columns.

conmo.preprocesses.CustomPreprocess

```
class conmo.preprocesses.CustomPreprocess(fn: Callable[[DataFrame, DataFrame], Tuple[DataFrame, DataFrame]])
```

Core class used to implement self-created preprocess. Such preprocess will be wrapped in a function that will be passed as an argument to the constructor of this class.

```
__init__(fn: Callable[[DataFrame, DataFrame], Tuple[DataFrame, DataFrame]]) → None
```

```
apply(in_dir: str, out_dir: str) → None
```

Applies the custom preprocess to labels and data.

Parameters

- **in_dir** (*str*) – Input directory where the files are located. Usually, this is the output directory of the splitter step.
- **out_dir** (*str*) – Output directory where the files will be saved.

```
load_input(in_dir: str) -> (<class 'pandas.core.frame.DataFrame'>, <class 'pandas.core.frame.DataFrame'>)
```

Read parquet data and labels files of the chosen dataset before it's split.

Parameters

in_dir (*str*) – Input directory where the files are located.

Returns

- **data** (*Pandas Dataframe*) – Loaded data file.
- **labels** (*Pandas Dataframe*) – Loaded labels file.

```
save_output(out_dir: str, data: DataFrame, labels: DataFrame) → None
```

Save preprocessed dataset to parquet format.

Parameters

- **out_dir** (*str*) – Output directory where the results will be saved.
- **data** (*Pandas Dataframe*) – Preprocessed data.
- **labels** (*Pandas Dataframe*) – Preprocessed labels.

show_start_message() → None

Simple method to print on the terminal the name of the selected splitter.

Methods

<code>__init__(fn)</code>	
<code>apply(in_dir, out_dir)</code>	Applies the custom preprocess to labels and data.
<code>load_input(in_dir)</code>	Read parquet data and labels files of the chosen dataset before it's split.
<code>save_output(out_dir, data, labels)</code>	Save preprocessed dataset to parquet format.
<code>show_start_message()</code>	Simple method to print on the terminal the name of the selected splitter.

conmo.preprocesses.RULImputation

class conmo.preprocesses.**RULImputation**(*threshold: int*)

__init__(*threshold: int*) → None

apply(*in_dir, out_dir*) → None

RUL imputation per TIME sample based on SEQUENCE labels, generating labels for each TIME.

load_input(*in_dir: str*) -> (<class 'pandas.core.frame.DataFrame'>, <class 'pandas.core.frame.DataFrame'>)

Read parquet data and labels files of the chosen dataset before it's split.

Parameters

in_dir (*str*) – Input directory where the files are located.

Returns

- **data** (*Pandas Dataframe*) – Loaded data file.
- **labels** (*Pandas Dataframe*) – Loaded labels file.

save_output(*out_dir: str, data: DataFrame, labels: DataFrame*) → None

Save preprocessed dataset to parquet format.

Parameters

- **out_dir** (*str*) – Output directory where the results will be saved.
- **data** (*Pandas Dataframe*) – Preprocessed data.
- **labels** (*Pandas Dataframe*) – Preprocessed labels.

show_start_message() → None

Simple method to print on the terminal the name of the selected splitter.

Methods

<code>__init__(threshold)</code>	
<code>apply(in_dir, out_dir)</code>	RUL imputation per TIME sample based on SEQUENCE labels, generating labels for each TIME.
<code>load_input(in_dir)</code>	Read parquet data and labels files of the chosen dataset before it's split.
<code>save_output(out_dir, data, labels)</code>	Save preprocessed dataset to parquet format.
<code>show_start_message()</code>	Simple method to print on the terminal the name of the selected splitter.

conmo.preprocesses.SavitzkyGolayFilter

class conmo.preprocesses.SavitzkyGolayFilter(*to_data: Union[bool, Iterable[str]], to_labels: Union[bool, Iterable[str]], test_set: bool, window_length: int, polyorder: int, deriv: Optional[int] = 0, delta: Optional[float] = 1.0, mode: Optional[str] = 'interp', cval: Optional[float] = 0.0*)

__init__(*to_data: Union[bool, Iterable[str]], to_labels: Union[bool, Iterable[str]], test_set: bool, window_length: int, polyorder: int, deriv: Optional[int] = 0, delta: Optional[float] = 1.0, mode: Optional[str] = 'interp', cval: Optional[float] = 0.0*)

apply(*in_dir: str, out_dir: str*) → None

Applies the preprocess to the given dataset.

Parameters

- **in_dir** (*str*) – Input directory where the files are located. Usually, this is the output directory of the splitter step.
- **out_dir** (*str*) – Output directory where the files will be saved.

extract_columns(*df: DataFrame, columns: Union[bool, Iterable[str]]*) → Iterable[str]

Returns a list containing all the column's name of the data.

Parameters

- **df** (*Pandas Dataframe*) – Dataframe containing the data.
- **columns** (*Union[bool, Iterable[str]]*) – Bool value if the dataframe has columns or the list of columns.

Returns

columns – List containing the names of the dataframe's columns.

Return type

Iterable[str]

load_input(*in_dir: str*) -> (<class 'pandas.core.frame.DataFrame'>, <class 'pandas.core.frame.DataFrame'>)

Read parquet data and labels files of the chosen dataset before it's split.

Parameters

in_dir (*str*) – Input directory where the files are located.

Returns

- **data** (*Pandas Dataframe*) – Loaded data file.
- **labels** (*Pandas Dataframe*) – Loaded labels file.

save_output(*out_dir: str, data: DataFrame, labels: DataFrame*) → None

Save preprocessed dataset to parquet format.

Parameters

- **out_dir** (*str*) – Output directory where the results will be saved.
- **data** (*Pandas Dataframe*) – Preprocessed data.
- **labels** (*Pandas Dataframe*) – Preprocessed labels.

show_start_message() → None

Simple method to print on the terminal the name of the selected splitter.

transform(*df: DataFrame, columns: Iterable[str]*) → DataFrame

Performs the preprocess over the dataframe with the given columns.

Parameters

- **df** (*Pandas Dataframe*) – Dataframe containing the data or the labels of the dataset.
- **columns** (*Iterable[str]*) – List of columns that will be used in the preprocess. Also the columns of the final dataframe.

Returns

Dataframe preprocessed.

Return type

Pandas Dataframe

Methods

<code>__init__(to_data, to_labels, test_set, ...)</code>	
<code>apply(in_dir, out_dir)</code>	Applies the preprocess to the given dataset.
<code>extract_columns(df, columns)</code>	Returns a list containig all the column's name of the data.
<code>load_input(in_dir)</code>	Read parquet data and labels files of the chosen dataset before it's split.
<code>save_output(out_dir, data, labels)</code>	Save preprocessed dataset to parquet format.
<code>show_start_message()</code>	Simple method to print on the terminal the name of the selected splitter.
<code>transform(df, columns)</code>	Performs the preprocess over the dataframe with the given columns.

conmo.preprocesses.SklearnPreprocess

```
class conmo.preprocesses.SklearnPreprocess(to_data: Union[bool, Iterable[str]], to_labels: Union[bool, Iterable[str]], test_set: bool, preprocess: Union[Binarizer, FunctionTransformer, KBinsDiscretizer, KernelCenterer, LabelBinarizer, LabelEncoder, MultiLabelBinarizer, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PolynomialFeatures, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler])
```

Class used to wrap existing preprocess in the Scikit-Learn library. It also allows this preprocess to be applied to certain columns of the dataset.

```
__init__(to_data: Union[bool, Iterable[str]], to_labels: Union[bool, Iterable[str]], test_set: bool, preprocess: Union[Binarizer, FunctionTransformer, KBinsDiscretizer, KernelCenterer, LabelBinarizer, LabelEncoder, MultiLabelBinarizer, MaxAbsScaler, MinMaxScaler, Normalizer, OneHotEncoder, OrdinalEncoder, PolynomialFeatures, PowerTransformer, QuantileTransformer, RobustScaler, StandardScaler]) → None
```

```
apply(in_dir: str, out_dir: str) → None
```

Applies the preprocess to the given dataset.

Parameters

- **in_dir** (*str*) – Input directory where the files are located. Usually, this is the output directory of the splitter step.
- **out_dir** (*str*) – Output directory where the files will be saved.

```
extract_columns(df: DataFrame, columns: Union[bool, Iterable[str]]) → Iterable[str]
```

Returns a list containing all the column's name of the data.

Parameters

- **df** (*Pandas Dataframe*) – Dataframe containing the data.
- **columns** (*Union[bool, Iterable[str]]*) – Bool value if the dataframe has columns or the list of columns.

Returns

columns – List containing the names of the dataframe's columns.

Return type

Iterable[str]

```
load_input(in_dir: str) -> (<class 'pandas.core.frame.DataFrame'>, <class 'pandas.core.frame.DataFrame'>)
```

Read parquet data and labels files of the chosen dataset before it's split.

Parameters

in_dir (*str*) – Input directory where the files are located.

Returns

- **data** (*Pandas Dataframe*) – Loaded data file.
- **labels** (*Pandas Dataframe*) – Loaded labels file.

```
save_output(out_dir: str, data: DataFrame, labels: DataFrame) → None
```

Save preprocessed dataset to parquet format.

Parameters

- **out_dir** (*str*) – Output directory where the results will be saved.
- **data** (*Pandas Dataframe*) – Preprocessed data.
- **labels** (*Pandas Dataframe*) – Preprocessed labels.

show_start_message() → None

Simple method to print on the terminal the name of the selected splitter.

transform(*df: DataFrame, columns: Iterable[str]*) → DataFrame

Performs the preprocess over the dataframe with the given columns.

Parameters

- **df** (*Pandas Dataframe*) – Dataframe containing the data or the labels of the dataset.
- **columns** (*Iterable[str]*) – List of columns that will be used in the preprocess. Also the columns of the final dataframe.

Returns

Dataframe preprocessed.

Return type

Pandas Dataframe

Methods

<code>__init__(to_data, to_labels, test_set, ...)</code>	
<code>apply(in_dir, out_dir)</code>	Applies the preprocess to the given dataset.
<code>extract_columns(df, columns)</code>	Returns a list containig all the column's name of the data.
<code>load_input(in_dir)</code>	Read parquet data and labels files of the chosen dataset before it's split.
<code>save_output(out_dir, data, labels)</code>	Save preprocessed dataset to parquet format.
<code>show_start_message()</code>	Simple method to print on the terminal the name of the selected splitter.
<code>transform(df, columns)</code>	Performs the preprocess over the dataframe with the given columns.

4.1.5 conmo.algorithms

The `conmo.algorithms` submodule contains everything related to algorithms in Conmo, from abstract classes to introduce new algorithms in Conmo to implementations of some of the algorithms used in the example experiments.

`algorithms.algorithm.Algorithm()`

`algorithms.algorithm.
AnomalyDetectionThresholdBasedAlgorithm(...)`

`algorithms.algorithm.
AnomalyDetectionClassBasedAlgorithm()`

`algorithms.PCAMahalanobis([n_components, ...])`

`algorithms.OneClassSVM([kernel, degree, ...])`

`algorithms.KerasAutoencoder([encoding_dim,
...])`

`algorithms.PretrainedRandomForest(pretrained)`

`algorithms.PretrainedMultilayerPerceptron(...)`

`algorithms.PretrainedCNN1D(pretrained, in-
put_len)`

conmo.algorithms.algorithm.Algorithm

class conmo.algorithms.algorithm.**Algorithm**

__init__()

execute(*idx: int, in_dir: str, out_dir: str*) → str

Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.

Parameters

- **idx** (*int*) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.
- **in_dir** (*str*) – Intermediate directory where the input data to the algorithm is stored.
- **out_dir** (*str*) – Intermediate directory where the output data (predictions of the algorithm) will be stored.

Returns

Name of the output directory.

Return type

str

abstract fit_predict(*data_train: DataFrame, data_test: DataFrame, labels_train: DataFrame, labels_test: DataFrame*) → DataFrame

Trains the model with train data and then performs predictions with the trained algorithm over the test data.

Parameters

- **data_train** (*Pandas Dataframe*) – Train data.
- **data_test** (*Pandas Dataframe*) – Test data.
- **labels_train** (*Pandas Dataframe*) – Train labels.

- **labels_test** (*Pandas Dataframe*) – Test labels.

Returns

Results of the predictions made on the test set.

Return type

Pandas Dataframe

labels_per_sequence(*labels: DataFrame*) → bool

Use only with time series datasets. Checks if the labels file of the chosen dataset has an index format with sequences only or sequences and time. *This method in future updates will be changed to a specific class for time series.*

Parameters

labels (*Pandas Dataframe*) – Labels file of the dataset.

Returns

True if the labels contains 1 level of index with sequence or False if the labels file contains 2 levels with sequence and time.

Return type

bool

Raises

RuntimeError – If the number of index levels is invalid.

load_input(*in_dir: str*) -> (<class 'pandas.core.frame.DataFrame'>, <class 'pandas.core.frame.DataFrame'>)

Read parquet data and labels files of the chosen dataset.

Parameters

in_dir (*str*) – Input directory where the files are located.

Returns

- **data** (*Pandas Dataframe*) – Loaded data file.
- **labels** (*Pandas Dataframe*) – Loaded labels file.

save_output(*results: DataFrame, out_dir: str, idx: int*) → str

Save algorithms output to parquet format.

Parameters

- **results** (*Pandas Dataframe*) – Dataframe with the results of the execution.
- **out_dir** (*str*) – Output directory where the results will be saved.
- **idx** (*int*) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.

show_start_message()

Simple method to print on the terminal the name of the algorithm to be executed.

Methods

<code>__init__()</code>	
<code>execute(idx, in_dir, out_dir)</code>	Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.
<code>fit_predict(data_train, data_test, ...)</code>	Trains the model with train data and then performs predictions with the trained algorithm over the test data.
<code>labels_per_sequence(labels)</code>	Use only with time series datasets.
<code>load_input(in_dir)</code>	Read parquet data and labels files of the chosen dataset.
<code>save_output(results, out_dir, idx)</code>	Save algorithms output to parquet format.
<code>show_start_message()</code>	Simple method to print on the terminal the name of the algorithm to be executed.

conmo.algorithms.algorithm.AnomalyDetectionThresholdBasedAlgorithm

```
class conmo.algorithms.algorithm.AnomalyDetectionThresholdBasedAlgorithm(threshold_mode: str,
                                                                           threshold_value:
                                                                           float)
```

```
__init__(threshold_mode: str, threshold_value: float)
```

```
execute(idx: int, in_dir: str, out_dir: str) → str
```

Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.

Parameters

- **idx** (*int*) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.
- **in_dir** (*str*) – Intermediate directory where the input data to the algorithm is stored.
- **out_dir** (*str*) – Intermediate directory where the output data (predictions of the algorithm) will be stored.

Returns

Name of the output directory.

Return type

str

```
find_anomaly_threshold(values: ndarray) → float
```

Finds anomaly threshold for threshold based algorithms. 3 different approaches are currently implemented.

Parameters

values (*Numpy ndarray*) – Results of the algorithm execution.

Returns

Calculated threshold value.

Return type

float

abstract fit_predict(*data_train: DataFrame, data_test: DataFrame, labels_train: DataFrame, labels_test: DataFrame*) → DataFrame

Trains the model with train data and then performs predictions with the trained algorithm over the test data.

Parameters

- **data_train** (*Pandas Dataframe*) – Train data.
- **data_test** (*Pandas Dataframe*) – Test data.
- **labels_train** (*Pandas Dataframe*) – Train labels.
- **labels_test** (*Pandas Dataframe*) – Test labels.

Returns

Results of the predictions made on the test set.

Return type

Pandas Dataframe

labels_per_sequence(*labels: DataFrame*) → bool

Use only with time series datasets. Checks if the labels file of the chosen dataset has an index format with sequences only or sequences and time. *This method in future updates will be changed to a specific class for time series.*

Parameters

labels (*Pandas Dataframe*) – Labels file of the dataset.

Returns

True if the labels contains 1 level of index with sequence or False if the labels file contains 2 levels with sequence and time.

Return type

bool

Raises

RuntimeError – If the number of index levels is invalid.

load_input(*in_dir: str*) -> (<class 'pandas.core.frame.DataFrame'>, <class 'pandas.core.frame.DataFrame'>)

Read parquet data and labels files of the chosen dataset.

Parameters

in_dir (*str*) – Input directory where the files are located.

Returns

- **data** (*Pandas Dataframe*) – Loaded data file.
- **labels** (*Pandas Dataframe*) – Loaded labels file.

save_output(*results: DataFrame, out_dir: str, idx: int*) → str

Save algorithms output to parquet format.

Parameters

- **results** (*Pandas Dataframe*) – Dataframe with the results of the execution.
- **out_dir** (*str*) – Output directory where the results will be saved.
- **idx** (*int*) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.

show_start_message()

Simple method to print on the terminal the name of the algorithm to be executed.

Methods

<code>__init__(threshold_mode, threshold_value)</code>	
<code>execute(idx, in_dir, out_dir)</code>	Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.
<code>find_anomaly_threshold(values)</code>	Finds anomaly threshold for threshold based algorithms.
<code>fit_predict(data_train, data_test, ...)</code>	Trains the model with train data and then performs predictions with the trained algorithm over the test data.
<code>labels_per_sequence(labels)</code>	Use only with time series datasets.
<code>load_input(in_dir)</code>	Read parquet data and labels files of the chosen dataset.
<code>save_output(results, out_dir, idx)</code>	Save algorithms output to parquet format.
<code>show_start_message()</code>	Simple method to print on the terminal the name of the algorithm to be executed.

conmo.algorithms.algorithm.AnomalyDetectionClassBasedAlgorithm

```
class conmo.algorithms.algorithm.AnomalyDetectionClassBasedAlgorithm
```

```
    __init__()
```

```
    execute(idx: int, in_dir: str, out_dir: str) → str
```

Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.

Parameters

- **idx** (*int*) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.
- **in_dir** (*str*) – Intermediate directory where the input data to the algorithm is stored.
- **out_dir** (*str*) – Intermediate directory where the output data (predictions of the algorithm) will be stored.

Returns

Name of the output directory.

Return type

str

```
    abstract fit_predict(data_train: DataFrame, data_test: DataFrame, labels_train:
                        DataFrame, labels_test: DataFrame) → DataFrame
```

Trains the model with train data and then performs predictions with the trained algorithm over the test data.

Parameters

- **data_train** (*Pandas Dataframe*) – Train data.
- **data_test** (*Pandas Dataframe*) – Test data.
- **labels_train** (*Pandas Dataframe*) – Train labels.
- **labels_test** (*Pandas Dataframe*) – Test labels.

Returns

Results of the predictions made on the test set.

Return type

Pandas Dataframe

labels_per_sequence(*labels: DataFrame*) → bool

Use only with time series datasets. Checks if the labels file of the chosen dataset has an index format with sequences only or sequences and time. *This method in future updates will be changed to a specific class for time series.*

Parameters

labels (*Pandas Dataframe*) – Labels file of the dataset.

Returns

True if the labels contains 1 level of index with sequence or False if the labels file contains 2 levels with sequence and time.

Return type

bool

Raises

RuntimeError – If the number of index levels is invalid.

load_input(*in_dir: str*) -> (<class 'pandas.core.frame.DataFrame'>, <class 'pandas.core.frame.DataFrame'>)

Read parquet data and labels files of the chosen dataset.

Parameters

in_dir (*str*) – Input directory where the files are located.

Returns

- **data** (*Pandas Dataframe*) – Loaded data file.
- **labels** (*Pandas Dataframe*) – Loaded labels file.

save_output(*results: DataFrame, out_dir: str, idx: int*) → str

Save algorithms output to parquet format.

Parameters

- **results** (*Pandas Dataframe*) – Dataframe with the results of the execution.
- **out_dir** (*str*) – Output directory where the results will be saved.
- **idx** (*int*) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.

show_start_message()

Simple method to print on the terminal the name of the algorithm to be executed.

Methods

<code>__init__()</code>	
<code>execute(idx, in_dir, out_dir)</code>	Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.
<code>fit_predict(data_train, data_test, ...)</code>	Trains the model with train data and then performs predictions with the trained algorithm over the test data.
<code>labels_per_sequence(labels)</code>	Use only with time series datasets.
<code>load_input(in_dir)</code>	Read parquet data and labels files of the chosen dataset.
<code>save_output(results, out_dir, idx)</code>	Save algorithms output to parquet format.
<code>show_start_message()</code>	Simple method to print on the terminal the name of the algorithm to be executed.

conmo.algorithms.PCAMahalanobis

class conmo.algorithms.PCAMahalanobis(*n_components: float = 0.95, robust_estimator: bool = False, threshold_mode: str = 'chi2', threshold_value: Optional[Union[int, float]] = 0.95*)

__init__(*n_components: float = 0.95, robust_estimator: bool = False, threshold_mode: str = 'chi2', threshold_value: Optional[Union[int, float]] = 0.95*)

execute(*idx: int, in_dir: str, out_dir: str*) → str

Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.

Parameters

- **idx** (*int*) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.
- **in_dir** (*str*) – Intermediate directory where the input data to the algorithm is stored.
- **out_dir** (*str*) – Intermediate directory where the output data (predictions of the algorithm) will be stored.

Returns

Name of the output directory.

Return type

str

find_anomaly_threshold(*values: ndarray, n_features: int*) → float

Finds anomaly threshold for threshold based algorithms. 3 different approaches are currently implemented.

Parameters

values (*Numpy ndarray*) – Results of the algorithm execution.

Returns

Calculated threshold value.

Return type

float

fit_predict(*data_train: DataFrame, data_test: DataFrame, labels_train: DataFrame, labels_test: DataFrame*) → DataFrame

Trains the model with train data and then performs predictions with the trained algorithm over the test data.

Parameters

- **data_train** (*Pandas Dataframe*) – Train data.
- **data_test** (*Pandas Dataframe*) – Test data.
- **labels_train** (*Pandas Dataframe*) – Train labels.
- **labels_test** (*Pandas Dataframe*) – Test labels.

Returns

Results of the predictions made on the test set.

Return type

Pandas Dataframe

labels_per_sequence(*labels: DataFrame*) → bool

Use only with time series datasets. Checks if the labels file of the chosen dataset has an index format with sequences only or sequences and time. *This method in future updates will be changed to a specific class for time series.*

Parameters

labels (*Pandas Dataframe*) – Labels file of the dataset.

Returns

True if the labels contains 1 level of index with sequence or False if the labels file contains 2 leves with sequence and time.

Return type

bool

Raises

RuntimeError – If the number of index levels is invalid.

load_input(*in_dir: str*) -> (<class 'pandas.core.frame.DataFrame'>, <class 'pandas.core.frame.DataFrame'>)

Read parquet data and labels files of the chosen dataset.

Parameters

in_dir (*str*) – Input directory where the files are located.

Returns

- **data** (*Pandas Dataframe*) – Loaded data file.
- **labels** (*Pandas Dataframe*) – Loaded labels file.

save_output(*results: DataFrame, out_dir: str, idx: int*) → str

Save algorithms output to parquet format.

Parameters

- **results** (*Pandas Dataframe*) – Dataframe with the results of the execution.
- **out_dir** (*str*) – Output directory where the results will be saved.

- **idx** (*int*) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.

show_start_message()

Simple method to print on the terminal the name of the algorithm to be executed.

Methods

<code>__init__([n_components, robust_estimator, ...])</code>	
<code>execute(idx, in_dir, out_dir)</code>	Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.
<code>find_anomaly_threshold(values, n_features)</code>	Finds anomaly threshold for threshold based algorithms.
<code>fit_predict(data_train, data_test, ...)</code>	Trains the model with train data and then performs predictions with the trained algorithm over the test data.
<code>labels_per_sequence(labels)</code>	Use only with time series datasets.
<code>load_input(in_dir)</code>	Read parquet data and labels files of the chosen dataset.
<code>save_output(results, out_dir, idx)</code>	Save algorithms output to parquet format.
<code>show_start_message()</code>	Simple method to print on the terminal the name of the algorithm to be executed.

conmo.algorithms.OneClassSVM

```
class conmo.algorithms.OneClassSVM(kernel: str = 'rbf', degree: int = 3, gamma: Union[str, float] = 'scale',
                                   coef0: float = 0.0, tol: float = 0.001, nu: float = 0.5)
```

```
    __init__(kernel: str = 'rbf', degree: int = 3, gamma: Union[str, float] = 'scale', coef0: float = 0.0, tol: float = 0.001, nu: float = 0.5)
```

```
    execute(idx: int, in_dir: str, out_dir: str) → str
```

Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.

Parameters

- **idx** (*int*) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.
- **in_dir** (*str*) – Intermediate directory where the input data to the algorithm is stored.
- **out_dir** (*str*) – Intermediate directory where the output data (predictions of the algorithm) will be stored.

Returns

Name of the output directory.

Return type

str

fit_predict(*data_train: DataFrame, data_test: DataFrame, labels_train: DataFrame, labels_test: DataFrame*) → DataFrame

Trains the model with train data and then performs predictions with the trained algorithm over the test data.

Parameters

- **data_train** (Pandas Dataframe) – Train data.
- **data_test** (Pandas Dataframe) – Test data.
- **labels_train** (Pandas Dataframe) – Train labels.
- **labels_test** (Pandas Dataframe) – Test labels.

Returns

Results of the predictions made on the test set.

Return type

Pandas Dataframe

labels_per_sequence(*labels: DataFrame*) → bool

Use only with time series datasets. Checks if the labels file of the chosen dataset has an index format with sequences only or sequences and time. *This method in future updates will be changed to a specific class for time series.*

Parameters

labels (Pandas Dataframe) – Labels file of the dataset.

Returns

True if the labels contains 1 level of index with sequence or False if the labels file contains 2 levels with sequence and time.

Return type

bool

Raises

RuntimeError – If the number of index levels is invalid.

load_input(*in_dir: str*) -> (<class 'pandas.core.frame.DataFrame'>, <class 'pandas.core.frame.DataFrame'>)

Read parquet data and labels files of the chosen dataset.

Parameters

in_dir (str) – Input directory where the files are located.

Returns

- **data** (Pandas Dataframe) – Loaded data file.
- **labels** (Pandas Dataframe) – Loaded labels file.

save_output(*results: DataFrame, out_dir: str, idx: int*) → str

Save algorithms output to parquet format.

Parameters

- **results** (Pandas Dataframe) – Dataframe with the results of the execution.
- **out_dir** (str) – Output directory where the results will be saved.
- **idx** (int) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.

show_start_message()

Simple method to print on the terminal the name of the algorithm to be executed.

Methods

<code>__init__</code> ([kernel, degree, gamma, coef0, tol, nu])	
<code>execute</code> (idx, in_dir, out_dir)	Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.
<code>fit_predict</code> (data_train, data_test, ...)	Trains the model with train data and then performs predictions with the trained algorithm over the test data.
<code>labels_per_sequence</code> (labels)	Use only with time series datasets.
<code>load_input</code> (in_dir)	Read parquet data and labels files of the chosen dataset.
<code>save_output</code> (results, out_dir, idx)	Save algorithms output to parquet format.
<code>show_start_message</code> ()	Simple method to print on the terminal the name of the algorithm to be executed.

conmo.algorithms.KerasAutoencoder

```
class conmo.algorithms.KerasAutoencoder(encoding_dim: int = 32, optimizer: str = 'Adam', loss_f: str =
    'mse', epochs: int = 2, batch_size: int = 64, random_seed: int =
    11, threshold_mode: str = 'chi2', threshold_value:
    Optional[Union[int, float]] = 0.95)
```

```
__init__(encoding_dim: int = 32, optimizer: str = 'Adam', loss_f: str = 'mse', epochs: int = 2,
    batch_size: int = 64, random_seed: int = 11, threshold_mode: str = 'chi2',
    threshold_value: Optional[Union[int, float]] = 0.95)
```

```
execute(idx: int, in_dir: str, out_dir: str) → str
```

Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.

Parameters

- **idx** (*int*) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.
- **in_dir** (*str*) – Intermediate directory where the input data to the algorithm is stored.
- **out_dir** (*str*) – Intermediate directory where the output data (predictions of the algorithm) will be stored.

Returns

Name of the output directory.

Return type

str

```
find_anomaly_threshold(values: ndarray, n_features: int) → float
```

Finds anomaly threshold for threshold based algorithms. 3 different approaches are currently implemented.

Parameters

values (*Numpy ndarray*) – Results of the algorithm execution.

Returns

Calculated threshold value.

Return type

float

fit_predict(*data_train: DataFrame, data_test: DataFrame, labels_train: DataFrame, labels_test: DataFrame*) → *DataFrame*

Trains the model with train data and then performs predictions with the trained algorithm over the test data.

Parameters

- **data_train** (*Pandas Dataframe*) – Train data.
- **data_test** (*Pandas Dataframe*) – Test data.
- **labels_train** (*Pandas Dataframe*) – Train labels.
- **labels_test** (*Pandas Dataframe*) – Test labels.

Returns

Results of the predictions made on the test set.

Return type

Pandas Dataframe

labels_per_sequence(*labels: DataFrame*) → *bool*

Use only with time series datasets. Checks if the labels file of the chosen dataset has an index format with sequences only or sequences and time. *This method in future updates will be changed to a specific class for time series.*

Parameters

labels (*Pandas Dataframe*) – Labels file of the dataset.

Returns

True if the labels contains 1 level of index with sequence or False if the labels file contains 2 levels with sequence and time.

Return type

bool

Raises

RuntimeError – If the number of index levels is invalid.

load_input(*in_dir: str*) -> (*<class 'pandas.core.frame.DataFrame'>*, *<class 'pandas.core.frame.DataFrame'>*)

Read parquet data and labels files of the chosen dataset.

Parameters

in_dir (*str*) – Input directory where the files are located.

Returns

- **data** (*Pandas Dataframe*) – Loaded data file.
- **labels** (*Pandas Dataframe*) – Loaded labels file.

save_output(*results: DataFrame, out_dir: str, idx: int*) → str

Save algorithms output to parquet format.

Parameters

- **results** (*Pandas Dataframe*) – Dataframe with the results of the execution.
- **out_dir** (*str*) – Output directory where the results will be saved.
- **idx** (*int*) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.

show_start_message()

Simple method to print on the terminal the name of the algorithm to be executed.

Methods

<code>__init__</code> (<i>encoding_dim, optimizer, loss_f, ...</i>)	
<code>execute</code> (<i>idx, in_dir, out_dir</i>)	Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.
<code>find_anomaly_threshold</code> (<i>values, n_features</i>)	Finds anomaly threshold for threshold based algorithms.
<code>fit_predict</code> (<i>data_train, data_test, ...</i>)	Trains the model with train data and then performs predictions with the trained algorithm over the test data.
<code>labels_per_sequence</code> (<i>labels</i>)	Use only with time series datasets.
<code>load_input</code> (<i>in_dir</i>)	Read parquet data and labels files of the chosen dataset.
<code>save_output</code> (<i>results, out_dir, idx</i>)	Save algorithms output to parquet format.
<code>show_start_message</code> ()	Simple method to print on the terminal the name of the algorithm to be executed.

conmo.algorithms.PretrainedRandomForest

class conmo.algorithms.PretrainedRandomForest(*pretrained: bool, max_depth: Optional[int] = None, random_state: Optional[int] = None, n_estimators: Optional[int] = None, path: Optional[str] = None*)

`__init__`(*pretrained: bool, max_depth: Optional[int] = None, random_state: Optional[int] = None, n_estimators: Optional[int] = None, path: Optional[str] = None*) → None

execute(*idx: int, in_dir: str, out_dir: str*) → str

Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.

Parameters

- **idx** (*int*) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.
- **in_dir** (*str*) – Intermediate directory where the input data to the algorithm is stored.

- **out_dir** (*str*) – Intermediate directory where the output data (predictions of the algorithm) will be stored.

Returns

Name of the output directory.

Return type

str

fit_predict(*data_train: DataFrame, data_test: DataFrame, labels_train: DataFrame, labels_test: DataFrame*) → *DataFrame*

Trains the model with train data and then performs predictions with the trained algorithm over the test data.

Parameters

- **data_train** (*Pandas Dataframe*) – Train data.
- **data_test** (*Pandas Dataframe*) – Test data.
- **labels_train** (*Pandas Dataframe*) – Train labels.
- **labels_test** (*Pandas Dataframe*) – Test labels.

Returns

Results of the predictions made on the test set.

Return type

Pandas Dataframe

labels_per_sequence(*labels: DataFrame*) → *bool*

Use only with time series datasets. Checks if the labels file of the chosen dataset has an index format with sequences only or sequences and time. *This method in future updates will be changed to a specific class for time series.*

Parameters

labels (*Pandas Dataframe*) – Labels file of the dataset.

Returns

True if the labels contains 1 level of index with sequence or False if the labels file contains 2 levels with sequence and time.

Return type

bool

Raises

RuntimeError – If the number of index levels is invalid.

load_input(*in_dir: str*) -> (*<class 'pandas.core.frame.DataFrame'>*, *<class 'pandas.core.frame.DataFrame'>*)

Read parquet data and labels files of the chosen dataset.

Parameters

in_dir (*str*) – Input directory where the files are located.

Returns

- **data** (*Pandas Dataframe*) – Loaded data file.
- **labels** (*Pandas Dataframe*) – Loaded labels file.

load_weights()

Load pretrained model/weights for the algorithm's path.

save_output(*results: DataFrame, out_dir: str, idx: int*) → str

Save algorithms output to parquet format.

Parameters

- **results** (*Pandas Dataframe*) – Dataframe with the results of the execution.
- **out_dir** (*str*) – Output directory where the results will be saved.
- **idx** (*int*) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.

show_start_message()

Simple method to print on the terminal the name of the algorithm to be executed.

Methods

<code>__init__</code> (<i>pretrained[, max_depth, ...]</i>)	
<code>execute</code> (<i>idx, in_dir, out_dir</i>)	Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.
<code>fit_predict</code> (<i>data_train, data_test, ...</i>)	Trains the model with train data and then performs predictions with the trained algorithm over the test data.
<code>labels_per_sequence</code> (<i>labels</i>)	Use only with time series datasets.
<code>load_input</code> (<i>in_dir</i>)	Read parquet data and labels files of the chosen dataset.
<code>load_weights</code> ()	Load pretrained model/weights for the algorithm's path.
<code>save_output</code> (<i>results, out_dir, idx</i>)	Save algorithms output to parquet format.
<code>show_start_message</code> ()	Simple method to print on the terminal the name of the algorithm to be executed.

conmo.algorithms.PretrainedMultilayerPerceptron

class conmo.algorithms.PretrainedMultilayerPerceptron(*pretrained: bool, input_len: int, random_seed: Optional[int] = None, path: Optional[str] = None*)

`__init__`(*pretrained: bool, input_len: int, random_seed: Optional[int] = None, path: Optional[str] = None*) → None

build_mlp() → Sequential

Auxiliary method for building the multilayer perceptron.

Returns

model – Keras model built.

Return type

tf.keras.Model

execute(*idx: int, in_dir: str, out_dir: str*) → str

Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.

Parameters

- **idx** (*int*) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.
- **in_dir** (*str*) – Intermediate directory where the input data to the algorithm is stored.
- **out_dir** (*str*) – Intermediate directory where the output data (predictions of the algorithm) will be stored.

Returns

Name of the output directory.

Return type

str

fit_predict(*data_train: DataFrame, data_test: DataFrame, labels_train: DataFrame, labels_test: DataFrame*) → DataFrame

Trains the model with train data and then performs predictions with the trained algorithm over the test data.

Parameters

- **data_train** (*Pandas Dataframe*) – Train data.
- **data_test** (*Pandas Dataframe*) – Test data.
- **labels_train** (*Pandas Dataframe*) – Train labels.
- **labels_test** (*Pandas Dataframe*) – Test labels.

Returns

Results of the predictions made on the test set.

Return type

Pandas Dataframe

labels_per_sequence(*labels: DataFrame*) → bool

Use only with time series datasets. Checks if the labels file of the chosen dataset has an index format with sequences only or sequences and time. *This method in future updates will be changed to a specific class for time series.*

Parameters

labels (*Pandas Dataframe*) – Labels file of the dataset.

Returns

True if the labels contains 1 level of index with sequence or False if the labels file contains 2 levels with sequence and time.

Return type

bool

Raises

RuntimeError – If the number of index levels is invalid.

load_input(*in_dir: str*) -> (<class 'pandas.core.frame.DataFrame'>, <class 'pandas.core.frame.DataFrame'>)

Read parquet data and labels files of the chosen dataset.

Parameters

in_dir (*str*) – Input directory where the files are located.

Returns

- **data** (*Pandas Dataframe*) – Loaded data file.
- **labels** (*Pandas Dataframe*) – Loaded labels file.

load_weights() → None

Load pretrained model/weights for the algorithm's path.

save_output(*results: DataFrame, out_dir: str, idx: int*) → *str*

Save algorithms output to parquet format.

Parameters

- **results** (*Pandas Dataframe*) – Dataframe with the results of the execution.
- **out_dir** (*str*) – Output directory where the results will be saved.
- **idx** (*int*) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.

show_start_message()

Simple method to print on the terminal the name of the algorithm to be executed.

Methods

<code>__init__(pretrained, input_len[, ...])</code>	
<code>build_mlp()</code>	Auxiliary method for building the multilayer perceptron.
<code>execute(idx, in_dir, out_dir)</code>	Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.
<code>fit_predict(data_train, data_test, ...)</code>	Trains the model with train data and then performs predictions with the trained algorithm over the test data.
<code>labels_per_sequence(labels)</code>	Use only with time series datasets.
<code>load_input(in_dir)</code>	Read parquet data and labels files of the chosen dataset.
<code>load_weights()</code>	Load pretrained model/weights for the algorithm's path.
<code>save_output(results, out_dir, idx)</code>	Save algorithms output to parquet format.
<code>show_start_message()</code>	Simple method to print on the terminal the name of the algorithm to be executed.

conmo.algorithms.PretrainedCNN1D

class conmo.algorithms.PretrainedCNN1D(*pretrained: bool, input_len: int, random_seed: Optional[int] = None, path: Optional[str] = None*)

__init__(*pretrained: bool, input_len: int, random_seed: Optional[int] = None, path: Optional[str] = None*) → None

build_cnn_1d() → Sequential

Auxiliary method for building the 1D convolutional neural network.

Returns

model – Keras model built.

Return type

tf.keras.Model

execute(*idx: int, in_dir: str, out_dir: str*) → str

Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.

Parameters

- **idx** (*int*) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.
- **in_dir** (*str*) – Intermediate directory where the input data to the algorithm is stored.
- **out_dir** (*str*) – Intermediate directory where the output data (predictions of the algorithm) will be stored.

Returns

Name of the output directory.

Return type

str

fit_predict(*data_train: DataFrame, data_test: DataFrame, labels_train: DataFrame, labels_test: DataFrame*) → DataFrame

Trains the model with train data and then performs predictions with the trained algorithm over the test data.

Parameters

- **data_train** (*Pandas Dataframe*) – Train data.
- **data_test** (*Pandas Dataframe*) – Test data.
- **labels_train** (*Pandas Dataframe*) – Train labels.
- **labels_test** (*Pandas Dataframe*) – Test labels.

Returns

Results of the predictions made on the test set.

Return type

Pandas Dataframe

labels_per_sequence(*labels: DataFrame*) → bool

Use only with time series datasets. Checks if the labels file of the chosen dataset has an index format with sequences only or sequences and time. *This method in future updates will be changed to a specific class for time series.*

Parameters

labels (*Pandas Dataframe*) – Labels file of the dataset.

Returns

True if the labels contains 1 level of index with sequence or False if the labels file contains 2 levels with sequence and time.

Return type

bool

Raises

RuntimeError – If the number of index levels is invalid.

load_input(*in_dir: str*) -> (<class 'pandas.core.frame.DataFrame'>, <class 'pandas.core.frame.DataFrame'>)

Read parquet data and labels files of the chosen dataset.

Parameters

in_dir (*str*) – Input directory where the files are located.

Returns

- **data** (*Pandas Dataframe*) – Loaded data file.
- **labels** (*Pandas Dataframe*) – Loaded labels file.

load_weights() → None

Load pretrained model/weights for the algorithm's path.

save_output(*results: DataFrame, out_dir: str, idx: int*) → str

Save algorithms output to parquet format.

Parameters

- **results** (*Pandas Dataframe*) – Dataframe with the results of the execution.
- **out_dir** (*str*) – Output directory where the results will be saved.
- **idx** (*int*) – Index of the algorithm in the Experiment. Useful in case you want to experiment with several algorithms.

show_start_message()

Simple method to print on the terminal the name of the algorithm to be executed.

Methods

<code>__init__(pretrained, input_len[, ...])</code>	
<code>build_cnn_1d()</code>	Auxiliary method for building the 1D convolutional neural network.
<code>execute(idx, in_dir, out_dir)</code>	Performs a complete execution of the algorithm, loading input data, performing a run through the folds and saving the results.
<code>fit_predict(data_train, data_test, ...)</code>	Trains the model with train data and then performs predictions with the trained algorithm over the test data.
<code>labels_per_sequence(labels)</code>	Use only with time series datasets.
<code>load_input(in_dir)</code>	Read parquet data and labels files of the chosen dataset.
<code>load_weights()</code>	Load pretrained model/weights for the algorithm's path.
<code>save_output(results, out_dir, idx)</code>	Save algorithms output to parquet format.
<code>show_start_message()</code>	Simple method to print on the terminal the name of the algorithm to be executed.

4.1.6 conmo.metrics

The `conmo.metrics` submodule contains everything necessary to add new ways of measuring the effectiveness of the implemented algorithms. Accuracy and RMSPE are currently implemented.

<code>metrics.metric.Metric()</code>
<code>metrics.Accuracy([normalize])</code>
<code>metrics.RMSPE([normalize])</code>

conmo.metrics.metric.Metric

class `conmo.metrics.metric.Metric`

`__init__()`

abstract `calculate(idx: int, algorithms: Iterable[str], last_preprocess_dir: str, algorithms_dir: str, metrics_dir: str) → None`

Calculates specific metric for each of the algorithms' results.

Parameters

- **idx** (*str*) – Index of the metric in the Experiment. Useful in case you want to calculate several metrics.
- **algorithms** (*Iterable[str]*) – List of names of the selected algorithms.
- **last_preprocess_dir** – Name of the directory where the ground truth is located

- **algorithms_dir** – Name of the directory where the results of the algorithms executions are stored.
- **metrics_dir** – Name of the directory where the results will be stored.

labels_per_sequence(*labels: DataFrame*) → bool

Use only with time series datasets. Checks if the labels file of the chosen dataset has an index format with sequences only or sequences and time. *This method in future updates will be changed to a specific class for time series.*

Parameters

labels (*Pandas Dataframe*) – Labels file of the dataset.

Returns

True if the labels contains 1 level of index with sequence or False if the labels file contains 2 levels with sequence and time.

Return type

bool

Raises

RuntimeError – If the number of index levels is invalid.

load_results(*algorithm: str, algorithms_dir: str*) → DataFrame

Load results for a specific algorithm.

Parameters

- **algorithm** (*str*) – Name of the selected algorithm.
- **algorithms_dir** (*str*) – Name of the directory where the results of the algorithms executions are stored.

Returns

Dataframe containing the results (predictions).

Return type

Pandas Dataframe

load_truth(*last_preprocess_dir: str*)

Load labels from the last preprocess directory.

Parameters

last_preprocess_dir (*str*) – Last directory where the labels dataframe was stored.

Returns

Dataframe containing the labels.

Return type

Pandas Dataframe

problem_label(*truth: DataFrame*) → str

Determinates the nature of the problem by identifying the column's name of the labels.

Parameters

truth (*Pandas Dataframe*) – Labels file of the dataset.

Returns

Returns the column for the metric.

Return type

str

Raises

RuntimeError – If the labels of the ground truth are invalid for the problem.

save_output(*metric: DataFrame, idx: int, metrics_dir: str*) → None

Save metric's output to disk.

Parameters

- **metric** (*Pandas Dataframe*) – Dataframe containing the metric's results.
- **idx** (*int*) – Index of the metric in the Experiment. Useful in case you want to calculate several metrics.
- **metrics_dir** (*str*) – Name of the directory where the results will be stored.

show_start_message()

Simple method to print on the terminal the name of the used metric.

Methods

<code>__init__()</code>	
<code>calculate(idx, algorithms, ...)</code>	Calculates specific metric for each of the algorithms' results.
<code>labels_per_sequence(labels)</code>	Use only with time series datasets.
<code>load_results(algorithm, algorithms_dir)</code>	Load results for a specific algorithm.
<code>load_truth(last_preprocess_dir)</code>	Load labels from the last preprocess directory.
<code>problem_label(truth)</code>	Determinates the nature of the problem by identifying the column's name of the labels.
<code>save_output(metric, idx, metrics_dir)</code>	Save metric's output to disk.
<code>show_start_message()</code>	Simple method to print on the terminal the name of the used metric.

conmo.metrics.Accuracy

class conmo.metrics.**Accuracy**(*normalize: bool = True*)

`__init__`(*normalize: bool = True*) → None

calculate(*idx: int, algorithms: Iterable[str], last_preprocess_dir: str, algorithms_dir: str, metrics_dir: str*) → None

Calculates specific metric for each of the algorithms' results.

Parameters

- **idx** (*str*) – Index of the metric in the Experiment. Useful in case you want to calculate several metrics.
- **algorithmss** (*Iterable[str]*) – List of names of the selected algorithms.
- **last_preprocess_dir** – Name of the directory where the ground truth is located

- **algorithms_dir** – Name of the directory where the results of the algorithms executions are stored.
- **metrics_dir** – Name of the directory where the results will be stored.

labels_per_sequence(*labels: DataFrame*) → bool

Use only with time series datasets. Checks if the labels file of the chosen dataset has an index format with sequences only or sequences and time. *This method in future updates will be changed to a specific class for time series.*

Parameters

labels (*Pandas Dataframe*) – Labels file of the dataset.

Returns

True if the labels contains 1 level of index with sequence or False if the labels file contains 2 levels with sequence and time.

Return type

bool

Raises

RuntimeError – If the number of index levels is invalid.

load_results(*algorithm: str, algorithms_dir: str*) → DataFrame

Load results for a specific algorithm.

Parameters

- **algorithm** (*str*) – Name of the selected algorithm.
- **algorithms_dir** (*str*) – Name of the directory where the results of the algorithms executions are stored.

Returns

Dataframe containing the results (predictions).

Return type

Pandas Dataframe

load_truth(*last_preprocess_dir: str*)

Load labels from the last preprocess directory.

Parameters

last_preprocess_dir (*str*) – Last directory where the labels dataframe was stored.

Returns

Dataframe containing the labels.

Return type

Pandas Dataframe

problem_label(*truth: DataFrame*) → str

Determinates the nature of the problem by identifying the column's name of the labels.

Parameters

truth (*Pandas Dataframe*) – Labels file of the dataset.

Returns

Returns the column for the metric.

Return type

str

Raises

RuntimeError – If the labels of the ground truth are invalid for the problem.

save_output(*metric: DataFrame, idx: int, metrics_dir: str*) → None

Save metric's output to disk.

Parameters

- **metric** (*Pandas Dataframe*) – Dataframe containing the metric's results.
- **idx** (*int*) – Index of the metric in the Experiment. Useful in case you want to calculate several metrics.
- **metrics_dir** (*str*) – Name of the directory where the results will be stored.

show_start_message()

Simple method to print on the terminal the name of the used metric.

Methods

<code>__init__</code> ([normalize])	
<code>calculate</code> (idx, algorithms, ...)	Calculates specific metric for each of the algorithms' results.
<code>labels_per_sequence</code> (labels)	Use only with time series datasets.
<code>load_results</code> (algorithm, algorithms_dir)	Load results for a specific algorithm.
<code>load_truth</code> (last_preprocess_dir)	Load labels from the last preprocess directory.
<code>problem_label</code> (truth)	Determinates the nature of the problem by identifying the column's name of the labels.
<code>save_output</code> (metric, idx, metrics_dir)	Save metric's output to disk.
<code>show_start_message</code> ()	Simple method to print on the terminal the name of the used metric.

conmo.metrics.RMSPE

class conmo.metrics.**RMSPE**(*normalize: bool = True*)

`__init__`(*normalize: bool = True*) → None

calculate(*idx: int, algorithms: Iterable[str], last_preprocess_dir: str, algorithms_dir: str, metrics_dir: str*) → None

Calculates specific metric for each of the algorithms' results.

Parameters

- **idx** (*str*) – Index of the metric in the Experiment. Useful in case you want to calculate several metrics.
- **algorithmss** (*Iterable[str]*) – List of names of the selected algorithms.
- **last_preprocess_dir** – Name of the directory where the ground truth is located

- **algorithms_dir** – Name of the directory where the results of the algorithms executions are stored.
- **metrics_dir** – Name of the directory where the results will be stored.

labels_per_sequence(*labels: DataFrame*) → bool

Use only with time series datasets. Checks if the labels file of the chosen dataset has an index format with sequences only or sequences and time. *This method in future updates will be changed to a specific class for time series.*

Parameters

labels (*Pandas Dataframe*) – Labels file of the dataset.

Returns

True if the labels contains 1 level of index with sequence or False if the labels file contains 2 levels with sequence and time.

Return type

bool

Raises

RuntimeError – If the number of index levels is invalid.

load_results(*algorithm: str, algorithms_dir: str*) → DataFrame

Load results for a specific algorithm.

Parameters

- **algorithm** (*str*) – Name of the selected algorithm.
- **algorithms_dir** (*str*) – Name of the directory where the results of the algorithms executions are stored.

Returns

Dataframe containing the results (predictions).

Return type

Pandas Dataframe

load_truth(*last_preprocess_dir: str*)

Load labels from the last preprocess directory.

Parameters

last_preprocess_dir (*str*) – Last directory where the labels dataframe was stored.

Returns

Dataframe containing the labels.

Return type

Pandas Dataframe

problem_label(*truth: DataFrame*) → str

Determinates the nature of the problem by identifying the column's name of the labels.

Parameters

truth (*Pandas Dataframe*) – Labels file of the dataset.

Returns

Returns the column for the metric.

Return type

str

Raises

RuntimeError – If the labels of the ground truth are invalid for the problem.

rmspe(*y_true: ndarray, y_pred: ndarray*) → ndarray

Compute Root Mean Square Percentage Error between two arrays.

save_output(*metric: DataFrame, idx: int, metrics_dir: str*) → None

Save metric's output to disk.

Parameters

- **metric** (*Pandas Dataframe*) – Dataframe containing the metric's results.
- **idx** (*int*) – Index of the metric in the Experiment. Useful in case you want to calculate several metrics.
- **metrics_dir** (*str*) – Name of the directory where the results will be stored.

show_start_message()

Simple method to print on the terminal the name of the used metric.

Methods

<code>__init__([normalize])</code>	
<code>calculate(idx, algorithms, ...)</code>	Calculates specific metric for each of the algorithms' results.
<code>labels_per_sequence(labels)</code>	Use only with time series datasets.
<code>load_results(algorithm, algorithms_dir)</code>	Load results for a specific algorithm.
<code>load_truth(last_preprocess_dir)</code>	Load labels from the last preprocess directory.
<code>problem_label(truth)</code>	Determinates the nature of the problem by identifying the column's name of the labels.
<code>rmspe(y_true, y_pred)</code>	Compute Root Mean Square Percentage Error between two arrays.
<code>save_output(metric, idx, metrics_dir)</code>	Save metric's output to disk.
<code>show_start_message()</code>	Simple method to print on the terminal the name of the used metric.

DEVELOPMENT GUIDE

5.1 Development guide

5.1.1 Possibilities of Conmo

Conmo framework has been designed to be user-friendly for recreating and evaluating experiments, but also for adding new algorithms, datasets, preprocesses, etc. This section explains the possibilities offered by this framework when implementing new submodules. We believe that using and contributing to Conmo can benefit all types of users, as well as helping to standardise comparisons between results from different scientific articles.

If you still have doubts about the implementation of new components to the framework, you can take a look at the API reference, examples or contact the developers.

5.1.2 Add a new dataset

Dataset is the core abstract class for every dataset in Conmo and contains basic methods and attributes that are common for all datasets. At the same time, two classes depend on it and differ according to where the original data is stored:

- ***LocalDataset*:**
Is the abstract class in charge of handling datasets that are stored locally on the computer where Conmo will be running. The main method of this class is *LocalDataset.load()*. It's in charge of parsing the original dataset files to Conmo's format and moving them to the data folder. It's an abstract method which needs to be implemented in every local dataset. There is also an abstract method *feed_pipeline()* to copy selected data to pipeline step folder.
- ***RemoteDataset*:**
In case the dataset to be implemented is originally located on a web server, a Git repository or other remote hosting, the *RemoteDataset* class is available in Conmo. Among all its methods, it's remarkable the *RemoteDataset.download()* to download the dataset from a remote URL.

For adding a new local dataset to the framework you need to create a new class that inherits from *LocalDataset* and override the following methods:

- ***LocalDataset.__init__()*:**
This is the constructor of the class. Here you can call the constructor of the father class to assign the path to the original dataset. Here you can also define some attributes of the class, like the label's columns names, feature's names. Also you can assign the subdataset that you want to instantiate.
- ***LocalDataset.dataset_files()*:**
This method must return a list with all the files (data and labels) that compounds the dataset.

- **`LocalDataset.load()`:**

This method must convert all raw dataset files to the appropriate format for Conmo's pipeline. For each of the datasets, first read and load the data and labels into Pandas dataframes, then concatenate them (e.g. train data and test data will be concatenated in one dataframe, the same for test) and finally save them in parquet format. Some considerations to take into account:

- Data and labels dataframes will have at least a multi-index for sequence and time. You can consult more information in the [Pandas documentation](#).
- The columns index must start at 1.
- If there dataset is only splitted into train and test, then there will be 2 sequences, one per set.
- In case the dataset is a time series with sequences, train sequences go after the test sequences.

- **`LocalDataset.feed_pipeline()`:**

This method is used to copy the dataset from *data* directory to the directory *Dataset* of the experiment.

- **`LocalDataset.sklearn_predefined_split()`:**

If you plan to use the Predefined Split from the Sklearn library your class must implement this method. It must generate an array of indexes of same length as sequences to be used with PredefinedSplit. The index must start at 0.

For adding a new remote dataset to the framework the procedure is almost identical to a local dataset. You need to create a new class that inherits from [RemoteDataset](#) and override the following methods:

- **`RemoteDataset.__init__()`:**

This is the constructor of the class. Here you can call the constructor of the father class to assign the path to the original dataset. You can also define some attributes of the class, like the label's columns names, features's names, file format, URL and checksum. Also you can assign the subdataset that you want to instantiate.

- **`RemoteDataset.dataset_files()`:**

This method must return a list with all the files (data and labels) that compounds the dataset.

- **`RemoteDataset.parse_to_package()`:**

Almost identical to [LocalDataset.load\(\)](#).

- **`RemoteDataset.feed_pipeline()`:**

This method is used to pass the dataset from *data* directory to the directory *Dataset* of the experiment.

- **`RemoteDataset.sklearn_predefined_split()`:**

If you plan to use the Predefined Split from the Sklearn library your class must implement this method. It must generate an array of indexes of same length as sequences to be used with PredefinedSplit. The index must start at 0.

5.1.3 Add a new algorithm

Conmo provides a core abstract class named [Algorithm](#) that contains the basic methods for the operation of any algorithm, mainly training with a training set, performing a prediction over test, loading and saving input and output data. Depending on the type of anomaly detection algorithm to be implemented, there are two classes depending on the operation of the method:

- **`AnomalyDetectionThresholdBasedAlgorithm`:**

If your algorithm needs to calculate a threshold to determine which samples are anomalous it must inherit from this class. For example: PCA Mahalanobis.

- **`AnomalyDetectionClassBasedAlgorithm`:**

If your algorithm identifies by classes the normal sequences from the anomalous ones, it must inherit from this class. For example: One Class SVM.

- **PretrainedAlgorithm:**

Check out this class if your algorithm was pre-trained prior to running an experiment, i.e. it is not necessary to train it during the experiment. It is required to be able to define the path where the pre-trained model is stored on disk.

For adding a new algorithm to the framework you need to create a new class that inherits from one of these classes depending of the type of the algorithm and override the following methods:

- **__init__():**

Constructor of the class. Here you can initialize all the hyperparameters needed for the algorithm. Also you can fix random seeds of Tensorflow, Numpy, etc here for reproducibility purposes.

- **fit_predict():**

Method responsible of building, training the model with the training data and testing it with the test set. In case your algorithm is threshold-based, it will be necessary to verify that each output in the test set exceeds that threshold to determine that it is anomalous. In the case of a class-based algorithm, depending on the output, it will be necessary to identify whether it is an outlier or an anomaly. Finally, the output dataframe has to be generated with the labels by sequence or by time.

- **find_anomaly_threshold():**

In case the algorithm is threshold based, the threshold selection can be customised overriding this method.

You can add auxiliary methods for model construction, weights loading, etc. in case the model structure is very complex.

5.1.4 Add a new splitter

The core abstract class is *Splitter* and provides some methods to load inputs, save outputs and check if the input was already splitted. For adding a new splitter you must create a new class that inherits from *Splitter* and implements the method `Transform()`. If the splitters you want to implement is available on Scikit-Learn library, we provide the class *SklearnSplitter* and indicating the name of the splitter to be used will allow you to use it in your experiment.

5.1.5 Add a new preprocess

ExtendedPreprocess Class is used for the implementation of new preprocessings in the pipeline. *ExtendedPreprocess* inherits from the core abstract class *Preprocess* and provides a constructor in order to define which parts of the dataset will be modified by the preprocessing: labels, data, test or train. Also permits to apply the preprocess to a specific set of columns. To define a new preprocess you only need to create a new class that inherits from *ExtendedPreprocess* and implements the method `Transform()`, where the preprocessing will be applied to the dataset. If the preprocess you want to implement is available on Sklearn library, we provide the class *SklearnPreprocess* and indicating the name of the preprocessing to be used will allow you to use it in your experiment. In order to make things easier, the *CustomPreprocess* class is available to implement a preprocessing tool from a function, which will be passed as an argument in the constructor. For additional information you can have a look at the example *nasa_cmapps.py*.

5.1.6 Add a new metric

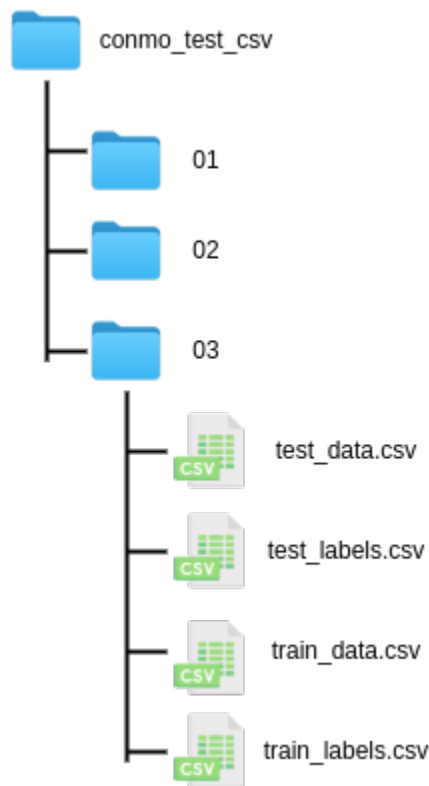
You can add a new metric by creating a new class that inherits from the abstract class *Metric*.

The only method you have to take care is:

- **calculate():**
Based on the outputs of the algorithms and the number of folds, the results are computed and the metrics dataframe is created and stored.

5.1.7 CSV dataset import example

A very common use case that Conmo users may encounter is to add a new dataset that is stored in CSV format. For this case we have developed this small guide, which includes a template as an example. The dataset is stored locally so it will inherit from LocalDataset. It contains three subdatasets stored in different directories, in all of them there are CSV files for data and labels, both for train and test:



The template:

```
1 import os
2 import shutil
3 from os import path
4 from typing import Iterable
5
6 import pandas as pd
7
```

(continues on next page)

(continued from previous page)

```

8 from conmo.conf import File, Index, Label
9 from conmo.datasets.dataset import LocalDataset
10
11
12 class CSV_Dataset(LocalDataset):
13     # -----
14     # Define constants here ...
15     # -----
16     #
17     EX_CONST = 22
18     EX_SUBDATASETS = ['01', '02', '03']
19     EX_COL_NAMES = ['A', 'B', 'C']
20
21     # -----
22     # Constructor of the class
23     # Call super class constructor to pass path where the raw dataset is stored
24     # Here you can initialize attributes with passed values
25     # and the specific subdataset to be used when instantiating
26     # -----
27     #
28
29     def __init__(self, path: str, subdataset: str) -> None:
30         super().__init__(path)
31         self.path = path
32         self.subdataset = subdataset
33
34     # -----
35     # Loads the original CSV files to Pandas dataframes,
36     # give them the appropriate format and finally save them to disk.
37     # -----
38     #
39     def load(self) -> None:
40         # SOME CONSIDERATIONS:
41         # - You can use Pandas utility read_csv()
42         # - Index must start at 1, not 0
43         # - Generate only 1 file for data and other for labels
44         # - Necessary a multi-index with two levels, an outer level of sequences and an
45         inner level of sequences.
46         # - If there is both train and test data, each of them shall form a sequence.

```

(continues on next page)

(continued from previous page)

```

46
47     # Iterate over files in the directory where the local original data is stored
48     for subdataset in os.listdir(self.path):
49         # -----
50         #
51         # Read data CSV and generate dataframe
52         train_data = pd.read_csv(path.join(
53             self.path, subdataset, 'train_data.csv'), sep=',', header=None,
54             names=self.EX_COL_NAMES)
55         test_data = pd.read_csv(path.join(
56             self.path, subdataset, 'test_data.csv'), sep=',', header=None,
57             names=self.EX_COL_NAMES)
58
59         # Reset index for starting from 1 (Conmos format)
60         train_data.index += 1
61         test_data.index += 1
62
63         # Concatenate train and test data into 1 dataframe. (Always first train data)
64         # Time is and old name and needs to be upgraded but the purpose is the same.
65         as a normal Index
66         data = pd.concat([train_data, test_data], keys=[
67             1, 2], names=[Index.SEQUENCE, Index.TIME])
68
69         # Sort index after concatenate
70         data.sort_index(inplace=True)
71
72         # -----
73         #
74         # Read labels CSV and generate dataframe
75         train_labels = pd.read_csv(path.join(
76             self.path, subdataset, 'train_labels.csv'), sep=',', header=None,
77             names=[Label.ANOMALY])
78         test_labels = pd.read_csv(path.join(
79             self.path, subdataset, 'train_labels.csv'), sep=',', header=None,
80             names=[Label.ANOMALY])
81
82         # Reset index for starting from 1 (Conmo's format)
83         train_labels.index += 1
84         test_labels.index += 1
85
86         # Concatenate train and test data into 1 dataframe. (Always first train data)
87         # Time is and old name and needs to be upgrade but the purpose is the same.
88         as a normal Index
89         labels = pd.concat([train_labels, test_labels], keys=[
90             1, 2], names=[Index.SEQUENCE, Index.TIME])
91
92         # Sort index after concatenate
93         labels.sort_index(inplace=True)
94
95         # -----
96         #
97         # Finally save dataframes to disk in /home/{username}/conmo/data/... in
98         parquet format

```

(continues on next page)

(continued from previous page)

```

89         data.to_parquet(path.join(self.dataset_dir, '{}_{}'.format(
90             subdataset, File.DATA)), compression='gzip', index=True)
91         labels.to_parquet(path.join(self.dataset_dir, '{}_{}'.format(
92             subdataset, File.LABELS)), compression='gzip', index=True)
93
94     # -----
95     #
96     # Method for adding to a list the different files that
97     #
98     # belong to the dataset
99     #
100    # Usually iterate over the subdatasets
101    #
102    # -----
103    #
104    #
105    def dataset_files(self) -> Iterable:
106        files = []
107        for key in self.EX_SUBDATASETS:
108            # Data
109            files.append(path.join(self.dataset_dir,
110                '{}_{}'.format(key, File.DATA)))
111            # Labels
112            files.append(path.join(self.dataset_dir,
113                '{}_{}'.format(key, File.LABELS)))
114        return files
115
116    # -----
117    #
118    # Method for adding to pipeline step folder
119    #
120    # Move from dataset_dir to out_dir data and labels
121    #
122    # -----
123    #
124    #
125    def feed_pipeline(self, out_dir: str) -> None:
126        # Data
127        shutil.copy(path.join(self.dataset_dir, '{}_{}'.format(
128            self.subdataset, File.DATA)), path.join(out_dir, File.DATA))
129        # Labels
130        shutil.copy(path.join(self.dataset_dir, '{}_{}'.format(
131            self.subdataset, File.LABELS)), path.join(out_dir, File.LABELS))
132
133    # -----
134    #
135    # OPTIONAL: Only implement if you plan to use
136    #
137    # PredefinedSplit method of Scikit-Learn library.
138    #
139    # Returns indexes of sequences:
140    #

```

(continues on next page)

(continued from previous page)

```

128     # -1 -> if the sequence will be excluded on test set
129     #
130     # 0 -> Test set
131     #
132     # -----
133     #
134     def sklearn_predefined_split(self) -> Iterable[int]:
135         return [-1, 0]

```

Once the class is ready, the respective import has to be added to the `__init__` file and the class name to the `__all__` list as follows:

```

1 from conmo.datasets.mars_science_laboratory_mission import MarsScienceLaboratoryMission
2 from conmo.datasets.nasa_turbofan_degradation import NASATurbofanDegradation
3 from conmo.datasets.server_machine_dataset import ServerMachineDataset
4 from conmo.datasets.soil_moisture_active_passive_satellite import _
5     ↳ SoilMoistureActivePassiveSatellite
6 from conmo.datasets.batteries_degradation import BatteriesDataset
7 #-----
8 # Add import to __init__ file of the module
9 from conmo.datasets.csv_dataset import CSV_Dataset
10 #-----
11 __all__ = [
12     'NASATurbofanDegradation'
13     'ServerMachineDataset'
14     'SoilMoistureActivePassiveSatellite'
15     'MarsScienceLaboratory'
16     'BatteriesDataset'
17     #-----
18     # Add class name here
19     'CSV_Dataset'
20     #-----
21 ]

```

Finally the dataset is ready to be used in an experiment:

```

1 import os
2 import sys
3
4 # Add package to path (Uncomment only in case you have downloaded Conmo from github_
5     ↳ repository)
6 sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
7
8 from sklearn.preprocessing import MinMaxScaler
9
10 from conmo.experiment import Experiment, Pipeline
11 from conmo.algorithms import OneClassSVM
12 from conmo.datasets import CSV_Dataset
13 from conmo.metrics import Accuracy
14 from conmo.preprocesses import SklearnPreprocess

```

(continues on next page)

(continued from previous page)

```

14 from conmo.splitters import SklearnSplitter
15 from sklearn.model_selection import PredefinedSplit
16 from sklearn.preprocessing import MinMaxScaler
17
18 # Pipeline definition
19 dataset = CSV_Dataset('/home/lucas/conmo_test_csv', '01')
20 splitter = SklearnSplitter(splitter=PredefinedSplit(dataset.sklearn_predefined_split()))
21 preproceses = [
22     SklearnPreprocess(to_data=True, to_labels=False,
23                       test_set=True, preprocess=MinMaxScaler()),
24 ]
25 algorithms = [
26     OneClassSVM()
27 ]
28 metrics = [
29     Accuracy()
30 ]
31 pipeline = Pipeline(dataset, splitter, preproceses, algorithms, metrics)
32
33
34 # Experiment definition and launch
35 experiment = Experiment([pipeline], [])
36 experiment.launch()

```

5.1.8 Coding conventions

The following tools are used to ensure that new software being added to Conmo meets minimum quality and format requirements:

- **Autopep8:** We use this tool to automatically format our Python code to conform to the PEP 8 style guide. It uses the pycodestyle utility to determine what parts of the code needs to be formatted.
- **Isort:** We use this library to sort imports alphabetically, and automatically separated into sections and by type.
- **Pytest:** To ensure that the output format of a new designed step (algorithm, dataset, etc) is correct we use Pytest framework to testing the new code. This tetsing frameworks is easy to use and supoort complex testing at the same. At the moment we are finishing the implementation of tests on the existing code, so there could be parts that may be modified in future updates.

KNOWN ISSUES & LIMITATIONS

6.1 Known Issues & Limitations

We are aware that Conmo is still in a very early stage of development, so it is likely that as its use increases, various bugs will appear. Bugs that are detected will be published on this page in order to make it easier for users to prevent them. However, the Conmo development team is actively looking for and fixing any detected bugs. Please, if you find a bug/issue that does not appear on this list, we would be grateful if you could email us at mym.inv.uniovi@gmail.com or post an issue on our Github. Thanks in advance.

Is-sue ID	Sever-ity	Description
001_split	low	There are some problems with the use of Scikit-Learn's Time Series Splitter in the experiments. We are working on resolving them.
002_run	Medium	run_rve.py example seems to be failing during the metric calculation step.
003_tf	Medium	If your computer has one of the new Apple processors (M1 or M2) with ARM-based architecture, it is likely that when you try to use Conmo, the Tensorflow dependency will fail. To fix this temporarily you can install Conmo without dependencies: 'pip install --no-deps conmo' and then manually install the branch provided by Google for ARM architectures tensorflow-macos.

FREQUENTLY ASKED QUESTIONS

7.1 Frequently Asked Questions

7.1.1 How can I contribute to Conmo?

Depending on your profile and your intended use, you can contribute in different ways: The simplest way to contribute to Conmo is to use it to reproduce some experiments and then cite it. However, you can also contribute by implementing new algorithms, datasets, etc. that can then be used by everyone to perform experiments. Finally, reporting bugs in the functioning of Conmo can also be considered a way to collaborate with the project.

7.1.2 I don't have a great knowledge of programming, can I still use Conmo?

Conmo intends to focus on all types of scientists, regardless of their specialisation. Generally speaking, we can distinguish two types of people who will use Conmo:

1. People who only want to reproduce experiments that are already integrated only need basic programming knowledge, since Python is a simple programming language and the complexity has been largely encapsulated.
2. People who want to collaborate by adding new algorithms, datasets, etc. need more in-depth programming knowledge. In particular Python language and object-oriented programming. However, the Conmo development team is actively looking for a way to simplify this kind of actions.

RELEASE NOTES

8.1 Release Notes

8.1.1 Conmo 1.0.1

Small documentation and CI errors fixed.

8.1.2 Conmo 1.0.0

First publicly available version.

PYTHON MODULE INDEX

C

`conmo.algorithms`, 45
`conmo.datasets`, 15
`conmo.experiment`, 13
`conmo.metrics`, 65
`conmo.preprocesses`, 35
`conmo.splitters`, 32

Symbols

`__init__()` (*conmo.algorithms.KerasAutoencoder* method), 56
`__init__()` (*conmo.algorithms.OneClassSVM* method), 54
`__init__()` (*conmo.algorithms.PCAMahalanobis* method), 52
`__init__()` (*conmo.algorithms.PretrainedCNN1D* method), 63
`__init__()` (*conmo.algorithms.PretrainedMultilayerPerceptron* method), 60
`__init__()` (*conmo.algorithms.PretrainedRandomForest* method), 58
`__init__()` (*conmo.algorithms.algorithm.Algorithm* method), 46
`__init__()` (*conmo.algorithms.algorithm.AnomalyDetectionClassBasedAlgorithm* method), 50
`__init__()` (*conmo.algorithms.algorithm.AnomalyDetectionThresholdBasedAlgorithm* method), 48
`__init__()` (*conmo.datasets.BatteriesDataset* method), 29
`__init__()` (*conmo.datasets.MarsScienceLaboratoryMission* method), 18
`__init__()` (*conmo.datasets.NASATurbofanDegradation* method), 26
`__init__()` (*conmo.datasets.ServerMachineDataset* method), 24
`__init__()` (*conmo.datasets.SoilMoistureActivePassiveSatellite* method), 21
`__init__()` (*conmo.datasets.dataset.Dataset* method), 15
`__init__()` (*conmo.datasets.dataset.LocalDataset* method), 17
`__init__()` (*conmo.datasets.dataset.RemoteDataset* method), 16
`__init__()` (*conmo.experiment.Experiment* method), 13
`__init__()` (*conmo.experiment.Pipeline* method), 14
`__init__()` (*conmo.metrics.Accuracy* method), 67
`__init__()` (*conmo.metrics.RMSPE* method), 69
`__init__()` (*conmo.metrics.metric.Metric* method), 65
`__init__()` (*conmo.preprocesses.Binarizer* method), 38
`__init__()` (*conmo.preprocesses.CustomPreprocess* method), 40
`__init__()` (*conmo.preprocesses.RULImputation* method), 41
`__init__()` (*conmo.preprocesses.SavitzkyGolayFilter* method), 42
`__init__()` (*conmo.preprocesses.SklearnPreprocess* method), 44
`__init__()` (*conmo.preprocesses.preprocess.ExtendedPreprocess* method), 37
`__init__()` (*conmo.preprocesses.preprocess.Preprocess* method), 36
`__init__()` (*conmo.splitters.SklearnSplitter* method), 34
`__init__()` (*conmo.splitters.splitter.Splitter* method), 32

A

`Accuracy` (class in *conmo.metrics*), 67
`AlgorithmBasedAlgorithm` (class in *conmo.algorithms.algorithm*), 46
`already_split()` (*conmo.splitters.SklearnSplitter* method), 34
`already_split()` (*conmo.splitters.splitter.Splitter* method), 32
`AnomalyDetectionClassBasedAlgorithm` (class in *conmo.algorithms.algorithm*), 50
`AnomalyDetectionThresholdBasedAlgorithm` (class in *conmo.algorithms.algorithm*), 48
`apply()` (*conmo.preprocesses.Binarizer* method), 38
`apply()` (*conmo.preprocesses.CustomPreprocess* method), 40
`apply()` (*conmo.preprocesses.preprocess.ExtendedPreprocess* method), 37
`apply()` (*conmo.preprocesses.preprocess.Preprocess* method), 36
`apply()` (*conmo.preprocesses.RULImputation* method), 41
`apply()` (*conmo.preprocesses.SavitzkyGolayFilter* method), 42
`apply()` (*conmo.preprocesses.SklearnPreprocess* method), 44

B

`BatteriesDataset` (class in *conmo.datasets*), 28

Binarizer (class in conmo.preprocesses), 38
 build_cnn_1d() (conmo.algorithms.PretrainedCNN1D
 method), 63
 build_mlp() (conmo.algorithms.PretrainedMultilayerPerceptron
 method), 60

C

calculate() (conmo.metrics.Accuracy method), 67
 calculate() (conmo.metrics.metric.Metric method), 65
 calculate() (conmo.metrics.RMSPE method), 69
 check_checksum() (conmo.datasets.dataset.RemoteDataset
 method), 16
 check_checksum() (conmo.datasets.MarsScienceLaboratoryMission
 method), 18
 check_checksum() (conmo.datasets.NASATurbofanDegradation
 method), 26
 check_checksum() (conmo.datasets.ServerMachineDataset
 method), 24
 check_checksum() (conmo.datasets.SoilMoistureActivePassiveSatellite
 method), 21
 check_checksum_lbl()
 (conmo.datasets.MarsScienceLaboratoryMission
 method), 18
 check_checksum_lbl()
 (conmo.datasets.SoilMoistureActivePassiveSatellite
 method), 21

conmo.algorithms
 module, 45
 conmo.datasets
 module, 15
 conmo.experiment
 module, 13
 conmo.metrics
 module, 65
 conmo.preprocesses
 module, 35
 conmo.splitters
 module, 32
 convert_to_input_data()
 (conmo.datasets.BatteriesDataset method),
 29

CustomPreprocess (class in conmo.preprocesses), 40

D

Dataset (class in conmo.datasets.dataset), 15
 dataset_files() (conmo.datasets.BatteriesDataset
 method), 29
 dataset_files() (conmo.datasets.dataset.Dataset
 method), 15
 dataset_files() (conmo.datasets.dataset.LocalDataset
 method), 17
 dataset_files() (conmo.datasets.dataset.RemoteDataset
 method), 16

dataset_files() (conmo.datasets.MarsScienceLaboratoryMission
 method), 19
 dataset_files() (conmo.datasets.NASATurbofanDegradation
 method), 26
 dataset_files() (conmo.datasets.ServerMachineDataset
 method), 24
 dataset_files() (conmo.datasets.SoilMoistureActivePassiveSatellite
 method), 22
 download() (conmo.datasets.dataset.RemoteDataset
 method), 16
 download() (conmo.datasets.MarsScienceLaboratoryMission
 method), 19
 download() (conmo.datasets.NASATurbofanDegradation
 method), 26
 download() (conmo.datasets.ServerMachineDataset
 method), 24
 download() (conmo.datasets.SoilMoistureActivePassiveSatellite
 method), 22
 download_anomalies_file()
 (conmo.datasets.MarsScienceLaboratoryMission
 method), 19
 download_anomalies_file()
 (conmo.datasets.SoilMoistureActivePassiveSatellite
 method), 22

E

execute() (conmo.algorithms.algorithm.Algorithm
 method), 46
 execute() (conmo.algorithms.algorithm.AnomalyDetectionClassBasedAlg
 method), 50
 execute() (conmo.algorithms.algorithm.AnomalyDetectionThresholdBase
 method), 48
 execute() (conmo.algorithms.KerasAutoencoder
 method), 56
 execute() (conmo.algorithms.OneClassSVM method),
 54
 execute() (conmo.algorithms.PCAMahalanobis
 method), 52
 execute() (conmo.algorithms.PretrainedCNN1D
 method), 63
 execute() (conmo.algorithms.PretrainedMultilayerPerceptron
 method), 60
 execute() (conmo.algorithms.PretrainedRandomForest
 method), 58
 Experiment (class in conmo.experiment), 13
 ExtendedPreprocess (class in
 conmo.preprocesses.preprocess), 37
 extract_columns() (conmo.preprocesses.Binarizer
 method), 38
 extract_columns() (conmo.preprocesses.preprocess.ExtendedPreproces
 method), 37
 extract_columns() (conmo.preprocesses.SavitzkyGolayFilter
 method), 42

[extract_columns\(\)](#) (*conmo.preprocesses.SklearnPreprocessor* method), 44
[extract_data\(\)](#) (*conmo.datasets.dataset.RemoteDataset* method), 16
[extract_data\(\)](#) (*conmo.datasets.MarsScienceLaboratoryMission* method), 19
[extract_data\(\)](#) (*conmo.datasets.NASATurbofanDegradation* method), 26
[extract_data\(\)](#) (*conmo.datasets.ServerMachineDataset* method), 24
[extract_data\(\)](#) (*conmo.datasets.SoilMoistureActivePassiveSatellite* method), 22
[extract_fold\(\)](#) (*conmo.splitters.SklearnSplitter* method), 34
F
[feed_pipeline\(\)](#) (*conmo.datasets.BatteriesDataset* method), 29
[feed_pipeline\(\)](#) (*conmo.datasets.dataset.LocalDataset* method), 17
[feed_pipeline\(\)](#) (*conmo.datasets.dataset.RemoteDataset* method), 17
[feed_pipeline\(\)](#) (*conmo.datasets.MarsScienceLaboratoryMission* method), 19
[feed_pipeline\(\)](#) (*conmo.datasets.NASATurbofanDegradation* method), 26
[feed_pipeline\(\)](#) (*conmo.datasets.ServerMachineDataset* method), 24
[feed_pipeline\(\)](#) (*conmo.datasets.SoilMoistureActivePassiveSatellite* method), 22
[fetch\(\)](#) (*conmo.datasets.BatteriesDataset* method), 29
[fetch\(\)](#) (*conmo.datasets.dataset.Dataset* method), 16
[fetch\(\)](#) (*conmo.datasets.dataset.LocalDataset* method), 18
[fetch\(\)](#) (*conmo.datasets.dataset.RemoteDataset* method), 17
[fetch\(\)](#) (*conmo.datasets.MarsScienceLaboratoryMission* method), 19
[fetch\(\)](#) (*conmo.datasets.NASATurbofanDegradation* method), 27
[fetch\(\)](#) (*conmo.datasets.ServerMachineDataset* method), 25
[fetch\(\)](#) (*conmo.datasets.SoilMoistureActivePassiveSatellite* method), 22
[find_anomaly_threshold\(\)](#) (*conmo.algorithms.algorithm.AnomalyDetectionThresholdBasedAlgorithm* method), 48
[find_anomaly_threshold\(\)](#) (*conmo.algorithms.KerasAutoencoder* method), 56
[find_anomaly_threshold\(\)](#) (*conmo.algorithms.PCAMahalanobis* method), 52
[fit_predict\(\)](#) (*conmo.algorithms.algorithm.Algorithm* method), 46
[fit_predict\(\)](#) (*conmo.algorithms.algorithm.AnomalyDetectionClassBasedAlgorithm* method), 50
[fit_predict\(\)](#) (*conmo.algorithms.algorithm.AnomalyDetectionThresholdBasedAlgorithm* method), 48
[fit_predict\(\)](#) (*conmo.algorithms.KerasAutoencoder* method), 57
[fit_predict\(\)](#) (*conmo.algorithms.OneClassSVM* method), 54
[fit_predict\(\)](#) (*conmo.algorithms.PCAMahalanobis* method), 53
[fit_predict\(\)](#) (*conmo.algorithms.PretrainedCNN1D* method), 63
[fit_predict\(\)](#) (*conmo.algorithms.PretrainedMultilayerPerceptron* method), 61
[fit_predict\(\)](#) (*conmo.algorithms.PretrainedRandomForest* method), 59
G
[generate_dirs\(\)](#) (*conmo.experiment.Experiment* method), 13
[generate_dirs\(\)](#) (*conmo.experiment.Pipeline* method), 14
[get_minmaxV\(\)](#) (*conmo.datasets.BatteriesDataset* method), 30
I
[is_dataset_ready\(\)](#) (*conmo.datasets.BatteriesDataset* method), 29
[is_dataset_ready\(\)](#) (*conmo.datasets.BatteriesDataset* method), 30
[is_dataset_ready\(\)](#) (*conmo.datasets.dataset.Dataset* method), 16
[is_dataset_ready\(\)](#) (*conmo.datasets.dataset.LocalDataset* method), 18
[is_dataset_ready\(\)](#) (*conmo.datasets.dataset.RemoteDataset* method), 17
[is_dataset_ready\(\)](#) (*conmo.datasets.MarsScienceLaboratoryMission* method), 19
[is_dataset_ready\(\)](#) (*conmo.datasets.NASATurbofanDegradation* method), 27
[is_dataset_ready\(\)](#) (*conmo.datasets.ServerMachineDataset* method), 25
[is_dataset_ready\(\)](#) (*conmo.datasets.SoilMoistureActivePassiveSatellite* method), 22
K
[KerasAutoencoder](#) (class in *conmo.algorithms*), 56
L
[labels_per_sequence\(\)](#) (*conmo.algorithms.algorithm.Algorithm* method), 47

`labels_per_sequence()` (`conmo.algorithms.algorithm.AnomalyDetectionClassBasedAlgorithm` method), 51
`labels_per_sequence()` (`conmo.algorithms.algorithm.AnomalyDetectionThresholdBasedAlgorithm` method), 49
`labels_per_sequence()` (`conmo.algorithms.KerasAutoencoder` method), 57
`labels_per_sequence()` (`conmo.algorithms.OneClassSVM` method), 55
`labels_per_sequence()` (`conmo.algorithms.PCAMahalanobis` method), 53
`labels_per_sequence()` (`conmo.algorithms.PretrainedCNNID` method), 63
`labels_per_sequence()` (`conmo.algorithms.PretrainedMultilayerPerceptron` method), 61
`labels_per_sequence()` (`conmo.algorithms.PretrainedRandomForest` method), 59
`labels_per_sequence()` (`conmo.metrics.Accuracy` method), 68
`labels_per_sequence()` (`conmo.metrics.metric.Metric` method), 66
`labels_per_sequence()` (`conmo.metrics.RMSPE` method), 70
`launch()` (`conmo.experiment.Experiment` method), 13
`load()` (`conmo.datasets.BatteriesDataset` method), 30
`load()` (`conmo.datasets.dataset.LocalDataset` method), 18
`load_input()` (`conmo.algorithms.algorithm.Algorithm` method), 47
`load_input()` (`conmo.algorithms.algorithm.AnomalyDetectionClassBasedAlgorithm` method), 51
`load_input()` (`conmo.algorithms.algorithm.AnomalyDetectionThresholdBasedAlgorithm` method), 49
`load_input()` (`conmo.algorithms.KerasAutoencoder` method), 57
`load_input()` (`conmo.algorithms.OneClassSVM` method), 55
`load_input()` (`conmo.algorithms.PCAMahalanobis` method), 53
`load_input()` (`conmo.algorithms.PretrainedCNNID` method), 64
`load_input()` (`conmo.algorithms.PretrainedMultilayerPerceptron` method), 61
`load_input()` (`conmo.algorithms.PretrainedRandomForest` method), 59
`load_input()` (`conmo.preprocesses.Binarizer` method), 39
`load_input()` (`conmo.preprocesses.CustomPreprocess` method), 40
`load_input()` (`conmo.preprocesses.preprocess.ExtendedPreprocess` method), 37
`load_input()` (`conmo.preprocesses.preprocess.Preprocess` method), 36
`load_input()` (`conmo.preprocesses.RULImputation` method), 41
`load_input()` (`conmo.preprocesses.SavitzkyGolayFilter` method), 42
`load_input()` (`conmo.preprocesses.SklearnPreprocess` method), 44
`load_input()` (`conmo.splitters.SklearnSplitter` method), 34
`load_input()` (`conmo.splitters.splitter.Splitter` method), 33
`load_results()` (`conmo.metrics.Accuracy` method), 68
`load_results()` (`conmo.metrics.metric.Metric` method), 66
`load_results()` (`conmo.metrics.RMSPE` method), 70
`load_truth()` (`conmo.metrics.Accuracy` method), 68
`load_truth()` (`conmo.metrics.metric.Metric` method), 66
`load_truth()` (`conmo.metrics.RMSPE` method), 70
`load_weights()` (`conmo.algorithms.PretrainedCNNID` method), 64
`load_weights()` (`conmo.algorithms.PretrainedMultilayerPerceptron` method), 62
`load_weights()` (`conmo.algorithms.PretrainedRandomForest` method), 59
`LocalDataset` (class in `conmo.datasets.dataset`), 17

M

`MarsScienceLaboratoryMission` (class in `conmo.datasets`), 18
`Metric` (class in `conmo.metrics.metric`), 65

N

`NASATurbofanDegradation` (class in `conmo.datasets`), 26
`normalise_data()` (`conmo.datasets.BatteriesDataset` method), 30

O

`OneClassSVM` (class in `conmo.algorithms`), 54

P

[parse_to_package\(\)](#) (*conmo.datasets.dataset.RemoteDataset* method), 17
[parse_to_package\(\)](#) (*conmo.datasets.MarsScienceLaboratoryMission* method), 19
[parse_to_package\(\)](#) (*conmo.datasets.NASATurbofanDegradation* method), 27
[parse_to_package\(\)](#) (*conmo.datasets.ServerMachineDataset* method), 25
[parse_to_package\(\)](#) (*conmo.datasets.SoilMoistureActivePassiveSatellite* method), 22
[PCAMahalanobis](#) (class in *conmo.algorithms*), 52
[Pipeline](#) (class in *conmo.experiment*), 14
[Preprocess](#) (class in *conmo.preprocesses.preprocess*), 36
[PretrainedCNN1D](#) (class in *conmo.algorithms*), 63
[PretrainedMultilayerPerceptron](#) (class in *conmo.algorithms*), 60
[PretrainedRandomForest](#) (class in *conmo.algorithms*), 58
[problem_label\(\)](#) (*conmo.metrics.Accuracy* method), 68
[problem_label\(\)](#) (*conmo.metrics.metric.Metric* method), 66
[problem_label\(\)](#) (*conmo.metrics.RMSPE* method), 70

R

[reduce_size\(\)](#) (*conmo.datasets.BatteriesDataset* method), 30
[RemoteDataset](#) (class in *conmo.datasets.dataset*), 16
[represent_anomalies\(\)](#) (*conmo.datasets.MarsScienceLaboratoryMission* method), 19
[represent_anomalies\(\)](#) (*conmo.datasets.SoilMoistureActivePassiveSatellite* method), 22
[RMSPE](#) (class in *conmo.metrics*), 69
[rmspe\(\)](#) (*conmo.metrics.RMSPE* method), 71
[RULImputation](#) (class in *conmo.preprocesses*), 41
[run\(\)](#) (*conmo.experiment.Pipeline* method), 14

S

[save_output\(\)](#) (*conmo.algorithms.algorithm.Algorithm* method), 47
[save_output\(\)](#) (*conmo.algorithms.algorithm.AnomalyDetectionClassBasedAlgorithm* method), 51
[save_output\(\)](#) (*conmo.algorithms.algorithm.AnomalyDetectionThresholdBasedAlgorithm* method), 49
[save_output\(\)](#) (*conmo.algorithms.KerasAutoencoder* method), 57
[save_output\(\)](#) (*conmo.algorithms.OneClassSVM* method), 55
[save_output\(\)](#) (*conmo.algorithms.PCAMahalanobis* method), 53
[save_output\(\)](#) (*conmo.algorithms.PretrainedCNN1D* method), 64
[save_output\(\)](#) (*conmo.algorithms.PretrainedMultilayerPerceptron* method), 62
[save_output\(\)](#) (*conmo.algorithms.PretrainedRandomForest* method), 59
[save_output\(\)](#) (*conmo.metrics.Accuracy* method), 69
[save_output\(\)](#) (*conmo.metrics.metric.Metric* method), 67
[save_output\(\)](#) (*conmo.metrics.RMSPE* method), 71
[save_output\(\)](#) (*conmo.preprocesses.Binarizer* method), 39
[save_output\(\)](#) (*conmo.preprocesses.CustomPreprocess* method), 40
[save_output\(\)](#) (*conmo.preprocesses.preprocess.ExtendedPreprocess* method), 37
[save_output\(\)](#) (*conmo.preprocesses.preprocess.Preprocess* method), 36
[save_output\(\)](#) (*conmo.preprocesses.RULImputation* method), 41
[save_output\(\)](#) (*conmo.preprocesses.SavitzkyGolayFilter* method), 43
[save_output\(\)](#) (*conmo.preprocesses.SklearnPreprocess* method), 44
[save_output\(\)](#) (*conmo.splitters.SklearnSplitter* method), 34
[save_output\(\)](#) (*conmo.splitters.splitter.Splitter* method), 33
[SavitzkyGolayFilter](#) (class in *conmo.preprocesses*), 42
[ServerMachineDataset](#) (class in *conmo.datasets*), 24
[show_start_message\(\)](#) (*conmo.algorithms.algorithm.Algorithm* method), 47
[show_start_message\(\)](#) (*conmo.algorithms.algorithm.AnomalyDetectionClassBasedAlgorithm* method), 51
[show_start_message\(\)](#) (*conmo.algorithms.algorithm.AnomalyDetectionThresholdBasedAlgorithm* method), 49
[show_start_message\(\)](#) (*conmo.algorithms.KerasAutoencoder* method), 58
[show_start_message\(\)](#) (*conmo.algorithms.OneClassSVM* method), 55
[show_start_message\(\)](#) (*conmo.algorithms.PCAMahalanobis* method), 54
[show_start_message\(\)](#) (*conmo.algorithms.PretrainedCNN1D* method), 64
[show_start_message\(\)](#) (*conmo.algorithms.PretrainedMultilayerPerceptron* method), 62

<code>show_start_message()</code> (<i>conmo.algorithms.PretrainedRandomForest</i> <i>method</i>), 60	<code>show_start_message()</code> (<i>conmo.splitters.SklearnSplitter</i> <i>method</i>), 34
<code>show_start_message()</code> (<i>conmo.datasets.BatteriesDataset</i> <i>method</i>), 30	<code>show_start_message()</code> (<i>conmo.splitters.splitter.Splitter</i> <i>method</i>), 33
<code>show_start_message()</code> (<i>conmo.datasets.dataset.Dataset</i> <i>method</i>), 16	<code>sklearn_predefined_split()</code> (<i>conmo.datasets.BatteriesDataset</i> <i>method</i>), 31
<code>show_start_message()</code> (<i>conmo.datasets.dataset.LocalDataset</i> <i>method</i>), 18	<code>sklearn_predefined_split()</code> (<i>conmo.datasets.MarsScienceLaboratoryMission</i> <i>method</i>), 20
<code>show_start_message()</code> (<i>conmo.datasets.dataset.RemoteDataset</i> <i>method</i>), 17	<code>sklearn_predefined_split()</code> (<i>conmo.datasets.NASATurbofanDegradation</i> <i>method</i>), 27
<code>show_start_message()</code> (<i>conmo.datasets.MarsScienceLaboratoryMission</i> <i>method</i>), 20	<code>sklearn_predefined_split()</code> (<i>conmo.datasets.ServerMachineDataset</i> <i>method</i>), 25
<code>show_start_message()</code> (<i>conmo.datasets.NASATurbofanDegradation</i> <i>method</i>), 27	<code>sklearn_predefined_split()</code> (<i>conmo.datasets.SoilMoistureActivePassiveSatellite</i> <i>method</i>), 23
<code>show_start_message()</code> (<i>conmo.datasets.ServerMachineDataset</i> <i>method</i>), 25	<code>SklearnPreprocess</code> (<i>class in conmo.preprocesses</i>), 44
<code>show_start_message()</code> (<i>conmo.datasets.SoilMoistureActivePassiveSatellite</i> <i>method</i>), 23	<code>SklearnSplitter</code> (<i>class in conmo.splitters</i>), 34
<code>show_start_message()</code> (<i>conmo.metrics.Accuracy</i> <i>method</i>), 69	<code>SoilMoistureActivePassiveSatellite</code> (<i>class in</i> <i>conmo.datasets</i>), 21
<code>show_start_message()</code> (<i>conmo.metrics.metric.Metric</i> <i>method</i>), 67	<code>split()</code> (<i>conmo.splitters.SklearnSplitter</i> <i>method</i>), 35
<code>show_start_message()</code> (<i>conmo.metrics.RMSPE</i> <i>method</i>), 71	<code>split()</code> (<i>conmo.splitters.splitter.Splitter</i> <i>method</i>), 33
<code>show_start_message()</code> (<i>conmo.preprocesses.Binarizer</i> <i>method</i>), 39	<code>Splitter</code> (<i>class in conmo.splitters.splitter</i>), 32
<code>show_start_message()</code> (<i>conmo.preprocesses.CustomPreprocess</i> <i>method</i>), 40	T
<code>show_start_message()</code> (<i>conmo.preprocesses.preprocess.ExtendedPreprocess</i> <i>method</i>), 37	<code>to_dataframe()</code> (<i>conmo.splitters.SklearnSplitter</i> <i>method</i>), 35
<code>show_start_message()</code> (<i>conmo.preprocesses.preprocess.Preprocess</i> <i>method</i>), 36	<code>transform()</code> (<i>conmo.preprocesses.Binarizer</i> <i>method</i>), 39
<code>show_start_message()</code> (<i>conmo.preprocesses.RULImputation</i> <i>method</i>), 41	<code>transform()</code> (<i>conmo.preprocesses.preprocess.ExtendedPreprocess</i> <i>method</i>), 38
<code>show_start_message()</code> (<i>conmo.preprocesses.SavitzkyGolayFilter</i> <i>method</i>), 43	<code>transform()</code> (<i>conmo.preprocesses.SavitzkyGolayFilter</i> <i>method</i>), 43
<code>show_start_message()</code> (<i>conmo.preprocesses.SklearnPreprocess</i> <i>method</i>), 45	<code>transform()</code> (<i>conmo.preprocesses.SklearnPreprocess</i> <i>method</i>), 45