# TimeEval

*Release 1.4.1*

**Sebastian Schmidl and Phillip Wenig**

**Apr 29, 2024**

# CONTENTS


1 User Guides 1

2 Concepts 13

3 API Reference 29

4 Contributor's Guide 185

5 Overview 187

6 User Guide 191

7 TimeEval Concepts 193

8 API Reference 195

9 Contributor's Guide 197

10 Additional Links 199

Bibliography 201

Python Module Index 203

Index 205

CHAPTER

ONE

# USER GUIDES

This part of the TimeEval documentation includes a couple of usage guides to get you started on using TimeEval for your own projects. The guides teach you TimeEval's APIs and their usage, but they do not get into detail about how TimeEval works. You can find the detailed descriptions of TimeEval concepts *here*.

## 1.1 Using TimeEval to evaluate algorithms

TimeEval is an evaluation tool for time series anomaly detection algorithms. We provide a large collection of compatible datasets and algorithms. The following section describes how you can set up TimeEval to perform your own experiments using the provided datasets and algorithms. The process consists of three steps: *preparing the datasets*, *preparing the algorithms*, and writing the *experiment script*.

### 1.1.1 Prepare datasets

This section assumes that you want to use the TimeEval datasets. If you want to use your own datasets with TimeEval, please read *How to use your own datasets in TimeEval*.

For the evaluation of time series anomaly detection algorithms, we collected univariate and multivariate time series datasets from various sources. We looked out for real-world as well as synthetically generated datasets with real-valued values and anomaly annotations. We included datasets with direct anomaly annotations (points or subsequences are labelled as either normal (0) or anomalous (1)) and indirect anomaly annotations. For the latter, we included datasets with categorical labels, where a single class (or low number of classes) is clearly underrepresented and can be assigned to unwanted, erroneous, or anomalous behavior. One example for this is an ECG signal with beat annotations, where most beats are annotated as normal beats, but some beats are premature or superventricular heart beats. The premature or superventricular heart beats can then be labelled as anomalous while the rest of the time series is normal behavior. Overall, we collected 1354 datasets (as of May 2022). For more details about the datasets, we refer you to the *Datasets* page of the repeatability website of our evaluation paper (doi:10.14778/3538598.3538602).

We grouped the datasets into 24 different dataset collection for easier download and management. The collections group datasets from a common source together, and you can download each dataset collection separately. Each dataset is thus identified by the tuple of collection name **and** dataset name.

TimeEval uses an index-File to discover datasets. It contains the links to the time series data and summarizes meta-data about them, such as number of anomalies, contamination, input dimensionality, support for supervised or semi-supervised training of algorithms, or the time series length. The index-File (named `datasets.csv`) for the paper's benchmark datasets can be downloaded from the repeatability page as well.

> **Warning:** The *GutenTAG* dataset collection comes with its own index-file!

The GutenTAG collection contains synthetically generated datasets using the GutenTAG dataset generator. It is compatible to TimeEval and generates TimeEval-compatible datasets and also the necessary metadata for the index-File.

The downloadable ZIP-archives contain the correct folder structure, but your extraction tool might place the contained files into a new folder that is named based on the ZIP-archive-name. The idea is that you download the index-File (`datasets.csv`) and just the dataset collections that you require, extract them all into the same folder, and then place the `datasets.csv` there. Please note or remember the name of your datasets folder. We will need it later, and we will refer to it as `<datasets-folder>`

**Example:**

Scenario: You want to use the datasets from the CalIt2 and Daphnet collections.

Dataset download and folder structure:

```
# Download CalIt2.zip, Daphnet.zip and datasets.csv
$ mkdir timeeval-datasets
$ mv datasets.csv timeeval-datasets/
$ unzip CalIt2.zip -d timeeval-datasets
$ unzip Daphnet.zip -d timeeval-datasets
$ tree timeeval-datasets
timeeval-datasets
├── datasets.csv
└── multivariate
    ├── CalIt2
    │   ├── CalIt2-traffic.metadata.json
    │   └── CalIt2-traffic.test.csv
    └── Daphnet
        ├── S01R01E0.metadata.json
        ├── S01R01E0.test.csv
        ├── S01R01E1.metadata.json
        ├── S01R01E1.test.csv
        ├── S01R02E0.metadata.json
        ├── S01R02E0.test.csv
        ├── [...]
        ├── S10R01E1.metadata.json
        └── S10R01E1.test.csv

3 directories, 77 files
```

To examine multiple datasets at once, TimeEval configuration would be:

```
dm = MultiDatasetManager([Path("timeeval-datasets")])
datasets = []
datasets.extend(dm.select(collection="CalIt2"))
datasets.extend(dm.select(collection="Daphnet"))
#...
```

A list of dataset with respective download links are given in *Datasets* page.

### 1.1.2 Prepare algorithms

This section assumes that you want to use the TimeEval algorithms. If you want to integrate your own algorithm into TimeEval, please read *How to integrate your own algorithm into TimeEval*.

We collected over 70 time series anomaly detection algorithms and integrated them into TimeEval (as of May 2022). All of our algorithm implementation make use of the `DockerAdapter` to allow you to use all features of TimeEval with them (such as resource restrictions, timeout, and fair runtime measurements). You can find the TimeEval algorithm implementations on GitHub: https://github.com/TimeEval/TimeEval-algorithms. Using Docker images to bundle an algorithm for TimeEval also allows easy integration of new algorithms because there are no requirements regarding programming languages, frameworks, or tools. Besides the many benefits, using Docker images to bundle algorithms makes preparing them for use with TimeEval a bit more cumbursome.

At the moment, we don't have the capacity to publish and maintain the algorithm's Docker images to a public Docker registry. This means that you have to build the Docker images from scratch before you can use the algorithms with TimeEval.

---

**Note:** If the community demand for pre-built TimeEval algorithm images rises, we will proudly assist in publishing and mainting publicly hosted images. However, this should be a community effort.

---

Please follow the following steps to prepare the algorithms to be evaluated with TimeEval. For further details about the Algorithm integration concept, please read the concept page *Algorithms*.

1. Clone or download the timeeval-algorithms repository
2. Build the base Docker image for your algorithm. You can find the image dependencies in the README-file of the repository. The base images are located in the folder `0-base-images`. Please make sure that you tag your image correctly (the image name must match the `FROM`-clause in your algorithm image; **this includes the image tag**). To be sure, you can tag the images based on our naming scheme, which uses the prefix `registry.gitlab.hpi.de/akita/i/`.
3. Optionally build an intermediate image, such as `registry.gitlab.hpi.de/akita/i/tsmp`, required for some algorithms.
4. Build the algorithm image.

Repeat the above steps for all algorithms that you want to execute with TimeEval. Creating a script to build all algorithm images is left as an exercise for the reader (tip: use `find` to get the correct folder and image names, and iterate over them). The README of the timeeval-algorithms repository contains example calls to test the algorithm Docker images.

### 1.1.3 Configure evaluation run

After we have prepared the datasets folder and the algorithm Docker images, we can install TimeEval and write an evaluation run script. You can install TimeEval from PyPI:

```
pip install TimeEval
```

We recommend to create a virtual environment with conda or virtualenv for TimeEval. The software requirements of TimeEval can be found on *the home page*.

When TimeEval ist installed, we can use its Python API to configure an evaluation run. We recommend to create a single Python-script for each evaluation run. The following snippet shows the main configuration options of TimeEval:

```
#!/usr/bin/env python3

from pathlib import Path
```

(continues on next page)

(continued from previous page)

```
from timeeval import TimeEval, MultiDatasetManager, DefaultMetrics, Algorithm,␣
↪TrainingType, InputDimensionality, ResourceConstraints
from timeeval.adapters import DockerAdapter
from timeeval.params import FixedParameters
from timeeval.resource_constraints import GB


def main():
    ####################
    # Load and select datasets
    ####################
    dm = MultiDatasetManager([
        Path("<datasets-folder>")  # e.g. ./timeeval-datasets
        # you can add multiple folders with an index-File to the MultiDatasetManager
    ])
    # A DatasetManager reads the index-File and allows you to access dataset metadata,
    # the datasets itself, or provides utilities to filter datasets by their metadata.
    # - select ALL available datasets
    # datasets = dm.select()
    # - select datasets from Daphnet collection
    datasets = dm.select(collection="Daphnet")
    # - select datasets with at least 2 anomalies
    # datasets = dm.select(min_anomalies=2)
    # - select multivariate datasets with a maximum contamination of 10%
    # datasets = dm.select(input_dimensionality=InputDimensionality.MULTIVARIATE, max_
↪contamination=0.1)

    # limit to 5 datasets for this example
    datasets = datasets[:5]

    ####################
    # Load and configure algorithms
    ####################
    # create a list of algorithm-definitions, we use a single algorithm in this example
    algorithms = [Algorithm(
        name="LOF",
        main=DockerAdapter(
            image_name="ghcr.io/timeeval/lof",
            tag="0.3.0",  # please use a specific tag instead of "latest" for␣
↪reproducibility
            skip_pull=True  # set to True because the image is already present from the␣
↪previous section
        ),
        # The hyperparameters of the algorithm are specified here. If you want to␣
↪perform a parameter
        # search, you can also perform simple grid search with TimeEval using␣
↪FullParameterGrid or
        # IndependentParameterGrid.
        param_config=FixedParameters({
            "n_neighbors": 50,
            "random_state": 42
```

(continues on next page)

(continued from previous page)

```
        }),
        # required by DockerAdapter
        data_as_file=True,
        # You must specify the algorithm metadata here. The categories for all TimeEval␣
↪algorithms can
        # be found in their README or their manifest.json-File.
        # UNSUPERVISED --> no training, SEMI_SUPERVISED --> training on normal data,␣
↪SUPERVISED --> training on anomalies
        # if SEMI_SUPERVISED or SUPERVISED, the datasets must have a corresponding␣
↪training time series
        training_type=TrainingType.UNSUPERVISED,
        # MULTIVARIATE (multidimensional TS) or UNIVARIATE (just a single dimension is␣
↪supported)
        input_dimensionality=InputDimensionality.MULTIVARIATE
    )]

    ####################
    # Configure evaluation run
    ####################
    # set the number of repetitions of each algorithm-dataset combination (e.g. for␣
↪runtime measurements):
    repetitions = 1
    # set resource constraints
    rcs = ResourceConstraints(
        task_memory_limit = 2 * GB,
        task_cpu_limit = 1.0,
    )

    # create TimeEval object and pass all the options
    timeeval = TimeEval(dm, datasets, algorithms,
        repetitions=repetitions,
        resource_constraints=rcs,
        # you can choose from different metrics:
        metrics=[DefaultMetrics.ROC_AUC, DefaultMetrics.FIXED_RANGE_PR_AUC],
    )

    # With run(), you can start the evaluation.
    timeeval.run()
    # You can access the overall evaluation results with:
    results = timeeval.get_results()
    print(results)

    # Detailed results are automatically stored in your current working directory at ./
↪results/<datestring>.


if __name__ == "__main__":
    main()
```

You can find more details about all exposed configuration options and methods in the *API Reference*.

If you are able to successfully execute the previous example evaluation, you can find more information at the following locations:

- *Add your own algorithm to TimeEval*
- *Add your own datasets to TimeEval*
- *Configure TimeEval and resource constraints*
- *Configure algorithm hyperparameters*
- *Using custom metrics*
- *Measuring algorithm runtime*
- *Executing TimeEval distributedly*

## 1.2 How to use your own datasets in TimeEval

You can use your own datasets with TimeEval. There are two ways to achieve this: using **custom datasets** or preparing your datasets as a TimeEval dataset collection. Either way, please familiarize yourself with the dataset format used by TimeEval described in the concept page *Time series datasets*.

### 1.2.1 1. Custom datasets

**Important:** The time series CSV-files must follow the *TimeEval canonical file format*!

To tell the TimeEval tool where it can find your custom datasets, a configuration file is needed. The custom datasets config file contains all custom datasets organized by their identifier which is used later on. Each entry in the config file must contain the path to the test time series; optionally, one can add a path to the training time series, specify the dataset type, and supply the period size if known. The paths to the data files must be absolute or relative to the configuration file. Example file `custom_datasets.json`:

```
{
  "dataset_name": {
    "test_path": "/absolute/path/to/data.csv"
  },
  "other_supervised_dataset": {
    "test_path": "/absolute/path/to/test.csv",
    "train_path": "./train.csv",
    "type": "synthetic",
    "period": 20
  }
}
```

You can add custom datasets to the dataset manager using two ways:

```
from pathlib import Path

from timeeval import DatasetManager
from timeeval.constants import HPI_CLUSTER

custom_datasets_path = Path("/absolute/path/to/custom_datasets.json")

# Directly during initialization
```

(continues on next page)

(continued from previous page)

```
dm = DatasetManager(HPI_CLUSTER.akita_dataset_paths[HPI_CLUSTER.BENCHMARK], custom_
↪datasets_file=custom_datasets_path)

# Later on
dm = DatasetManager(HPI_CLUSTER.akita_dataset_paths[HPI_CLUSTER.BENCHMARK])
dm.load_custom_datasets(custom_datasets_path)
```

### 1.2.2 2. Create a TimeEval dataset collection

**Warning:** WIP

## 1.3 How to integrate your own algorithm into TimeEval

If your algorithm is written in Python, you could use our *FunctionAdapter* (*Example* of using the `FunctionAdapter`). However, this comes with some limitations (such as no way to limit resource usage or setting timeouts). We, therefore, highly recommend to use the *DockerAdapter*. This means that you have to create a Docker image for your algorithm before you can use it in TimeEval.

In the following, we assume that you want to create a Docker image with your algorithm to execute it with TimeEval. We provide base images for various programming languages. You can find them here.

### 1.3.1 Procedure

This section contains a short guide on how to integrate your own algorithm into TimeEval **using the `DockerAdapter`**. There are three main steps: (i) Preparing the base image, (ii) creating the algorithm image, and (iii) using the algorithm image within TimeEval.

#### (i) Prepare the base image

1. Clone the `timeeval-algorithms`-repository
2. Build the selected base image from `0-base-images`. Please make sure that you tag your image correctly (the image name must match the `FROM`-clause in your algorithm image).
   - change to the `0-base-images` folder: `cd 0-base-images`
   - build your desired base image, e.g. `docker build -t ghcr.io/timeeval/python3-base:0.3.0 ./python3-base`
   - (optionally: build derived base image, e.g. `docker build -t ghcr.io/timeeval/pyod:0.3.0 ./pyod`)
   - now you can build your algorithm image from the base image (see next section)

   Alternatively, you can also pull one of the existing base images from the registry, e.g.:

```
docker pull ghcr.io/timeeval/pyod:0.3.0
```

**Note:** Please contact the maintainers if there is no base image for your algorithm programming language or runtime.

### (ii) Integrate your algorithm into TimeEval by creating an algorithm image

You can use any algorithm in the `timeeval-algorithms`-repository as an example for that.

TimeEval uses a common interface to execute all its Docker algorithms. This interface describes data input and output as well as algorithm configuration. The calling-interface is described in *TimeEval algorithm interface*. Please read the section carefully and adapt your algorithm to the interface description. You could also create a wrapper script that takes care of the integration. Our canonical file format for time series datasets is described *here*. Once you are familiar with the concepts, you can adapt your algorithm and create its Docker image:

1. Create a `Dockerfile` for your algorithm that is based on your selected base image. Example:

```
FROM ghcr.io/timeeval/python3-base:0.3.0

LABEL maintainer="sebastian.schmidl@hpi.de"

ENV ALGORITHM_MAIN="/app/algorithm.py"

# install algorithm dependencies
COPY requirements.txt /app/
RUN pip install -r /app/requirements.txt

# add algorithm implementation
COPY algorithm.py /app/
```

2. Build your algorithm Docker image, e.g. `docker build -t my_algorithm:latest Dockerfile`
3. Check if your algorithm is compatible to TimeEval.
   - Check if your algorithm can read a time series using our common file format.
   - Check if the algorithm parameters are correctly set using TimeEval's call format.
   - Check if the anomaly scores are written in the correct format (an anomaly score value for each point of the original time series in a headerless CSV-file).

   The README of the `timeeval-algorithms`-repository provides further details and instructions on how to create and test TimeEval algorithm images, including example calls.

### (iii) Use algorithm image within TimeEval

Create an experiment script with your configuration of datasets and your own algorithm image. Make sure that you specify your algorithm's image and tag name correctly and use `skip_pull=True`. This prevents TimeEval from trying to update your algorithm image by fetching it from a Docker registry because your image was not published to any registry. In addition, `data_as_file` must also be enabled for all algorithms using the `DockerAdapter`. Please also specify the algorithm's learning type (whether it requires training and which training data) and input dimensionality (uni- or multivariate):

```
#!/usr/bin/env python3

from pathlib import Path

from timeeval import TimeEval, MultiDatasetManager, Algorithm, TrainingType,␣
↪InputDimensionality
from timeeval.adapters import DockerAdapter
from timeeval.params import FixedParameters
```

(continues on next page)

(continued from previous page)

```
def main():
    dm = MultiDatasetManager([Path("<datasets-folder>")])
    datasets = dm.select()

    ####################
    # Add your own algorithm
    ####################
    algorithms = [Algorithm(
        name="<my_algorithm>",
        main=DockerAdapter(
            image_name="<my_algorithm>",
            tag="latest",
            skip_pull=True  # must be set to True because your image is just available␣
↪locally
        ),
        # Set your custom parameters:
        param_config=FixedParameters({
            "random_state": 42,
            # ...
        }),
        # required by DockerAdapter
        data_as_file=True,
        # set the following metadata based on your algorithm:
        training_type=TrainingType.UNSUPERVISED,
        input_dimensionality=InputDimensionality.MULTIVARIATE
    )]


    timeeval = TimeEval(dm, datasets, algorithms)
    timeeval.run()
    results = timeeval.get_results()
    print(results)


if __name__ == "__main__":
    main()
```

## 1.4 Using custom evaluation metrics

> **Warning:** This section is still work in progress.
>
> Until we finish the documentation, please consult the API documentation of the base class *Metric* for a short explanation of the interface and an example.

## 1.5 Repetitive runs and measuring runtime

TimeEval has the ability to run an experiment multiple times to improve runtime measurements. Therefore, *timeeval.TimeEval* has the parameter `repetitions: int = 1`, which tells TimeEval how many times to execute each experiment (algorithm, hyperparameters, and dataset combination).

When measuring runtime, we highly recommend to use TimeEval's feature to limit each algorithm to a specific set of resources (meaning CPU and memory). This requires using the *timeeval.adapters.docker.DockerAdapter* for the algorithms. See the concept page *TimeEval configuration and resource restrictions* for more details.

To retrieve the aggregated results, you can call *timeeval.TimeEval.get_results()* with the parameter `aggregated: bool = True`. This aggregates the quality and runtime measurements using mean and standard deviation. Erroneous experiments are excluded from an aggregate. For example, if you have `repetitions = 5` and one of five experiments failed, the average is built only over the 4 successful runs.

## 1.6 (Advanced) Distributed execution of TimeEval

**Important:** Before continuing with this guide, please make sure that you have read and understood *this concept page*.

TimeEval uses Dask's SSHCluster to distribute tasks on a compute cluster. This means that *certain prerequisites* must be fulfilled before TimeEval can be executed in distributed mode.

We assume that the following requirements are already fulfilled for all hosts of the cluster (independent if the host has the *driver*, *scheduler*, or *worker* role):

- Python 3 and Docker is installed
- Every node has a virtual environment (Anaconda, virtualenv or similar) with the same name (e.g. `timeeval`) **and prefix**!
- The same TimeEval version is installed in all `timeeval` environments.
- All nodes can reach each other via network (especially via SSH).

Similar to the *local execution of TimeEval*, we also have to prepare the datasets and algorithms first.

### 1.6.1 Prepare time series datasets

1. Please create a dataset folder on each node using the same path. For example: `/data/timeeval-datasets`.
2. Copy your datasets and also the index-file (`datasets.csv`) to all nodes.
3. Test if TimeEval can access this folder and find your datasets on each node:

```
from timeeval import DatasetManager

dmgr = DatasetManager("/data/timeeval-datasets", create_if_missing=False)
dataset = dmgr.get(("<your-collection-name>", "<your-dataset-name>"))
```

### 1.6.2 Prepare algorithms

If you use plain **Python function**s as algorithm implementations and the *FunctionAdapter*, please make sure that your Python code is either installed as a module or that the algorithm implementation is part of your single script-file. Your Python script with the experiment configuration is not allowed to import any other **local** files (e.g., `from .util import xyz`). This is due to issues with the Python-Path on the remote machines.

If you use **Docker images** for your algorithms and the *DockerAdapter*, the algorithm images must be present on all nodes or Docker must be able to pull them from a remote registry (can be controlled with `skip_pull=False`).

There are different ways to get the Docker images to all hosts:

- Build the Docker images locally on each machine (e.g., using a terminal multiplexer)
- Build the Docker images on one machine and distribute them. This can be accomplished using image export and import. You can follow these rough outline of steps: `docker build`, `docker image save`, `rsync` to all machines, `docker image import`
- Push / publish image to a registry available to you (if it's public, you would be responsible for maintaining it)
- Host your own registry

At the end, TimeEval must be able to create the algorithms' Docker containers, otherwise it is not able to execute and evaluate them.

### 1.6.3 TimeEval configuration for distributed execution

Setting up TimeEval for distributed execution follows the same principles as for *local execution*. Two arguments to the *TimeEval-constructor* are relevant for the distributed setup: `distributed: bool = False` and `remote_config: Optional[RemoteConfiguration] = None`. You can enable the distributed execution with `distributed=True` and configure the cluster using the *RemoteConfiguration* object. The following snippet shows the available configuration options:

```
import sys
from timeeval import RemoteConfiguration

RemoteConfiguration(
    scheduler_host = "localhost",        # scheduler host
    worker_hosts = [],                   # list of worker hosts
    remote_python = sys.executable,      # path to the python executable (same on all␣
↪hosts)
    dask_logging_file_level = "INFO",    # logging level for the file-based logger
    dask_logging_console_level = "INFO", # logging level for the console logger
    dask_logging_filename = "dask.log",  # filename for the file-based logger used for␣
↪the Dask-logs
    kwargs_overwrites = {}               # advanced options for DaskSSHCluster
)
```

The *driver* host (executing TimeEval) must be able to open an SSH connection to all the other nodes using **passwordless SSH**, otherwise, TimeEval will not be able to reach the other nodes.

If you use resource constraints, please make sure that you set the number of tasks per hosts and the CPU und memory limits correctly. We highly discourage over-provisioning. For more details, see the *concept page about resource restrictions*.

## 1.7 (Advanced) Using hyperparameter heuristics

**Warning:** WIP

CHAPTER

TWO

# CONCEPTS

## 2.1 Time series datasets

TimeEval uses a canonical file format for datasets. Existing datasets in another format must first be transformed into the canonical format before they can be used with TimeEval.

### 2.1.1 Canonical file format

TimeEval's canonical file format is based on CSV. Each file requires a header, cells (values) are separated by commas (decimal seperator is `.`), and records are separated by newlines (unix-style LF: `\n`). The first column of the dataset is its index, either in integer- or datetime-format (multiple timestamp-formats are supported but RFC 3339 is preferred, e.g. `2017-03-22 15:16:45.433502912`). The index follows a single or multiple (if multivariate dataset) time series columns. The last column contains the annotations, `0` for normal points and `1` for anomalies. Usage of the `timestamp` and `is_anomaly` column headers is recommended.

```
timestamp,value,is_anomaly
0,12751.0,1
1,8767.0,0
2,7005.0,0
3,5257.0,0
4,4189.0,0
```

### 2.1.2 Registering datasets

TimeEval comes with its own collection of benchmark datasets (**currently not included**, download them from our website). They can directly be used from the dataset manager `DatasetManager`:

```
from pathlib import Path

from timeeval import DatasetManager
from timeeval.constants import HPI_CLUSTER

datasets_folder: Path = HPI_CLUSTER.akita_dataset_paths[HPI_CLUSTER.BENCHMARK]  # or␣
↪Path("./datasets-folder")
dm = DatasetManager(datasets_folder)
datasets = dm.select()
```

#### Custom datasets

**Important:** WIP!

Example file `custom_datasets.json`:

```
{
  "dataset_name": {
    "test_path": "/absolute/path/to/data.csv"
  },
  "other_supervised_dataset": {
    "test_path": "/absolute/path/to/test.csv",
    "train_path": "./train.csv",
    "type": "synthetic",
    "period": 20
  }
}
```

You can register custom datasets at the dataset manager using two ways:

```
from pathlib import Path

from timeeval import DatasetManager
from timeeval.constants import HPI_CLUSTER

custom_datasets_path = Path("/absolute/path/to/custom_datasets.json")

# Directly during initialization
dm = DatasetManager(HPI_CLUSTER.akita_dataset_paths[HPI_CLUSTER.BENCHMARK], custom_
↪datasets_file=custom_datasets_path)

# Later on
dm = DatasetManager(HPI_CLUSTER.akita_dataset_paths[HPI_CLUSTER.BENCHMARK])
dm.load_custom_datasets(custom_datasets_path)
```

### 2.1.3 Preparing datasets for TimeEval

Datasets in different formats should be transformed in TimeEval's canonical file format. TimeEval provides a utility script to perform this transformation: `scripts/preprocess_dataset.py`. You can download this scrip from its GitHub repository.

A single dataset can be provided in two Numpy-readable text files. The first text file contains the data. The labels must be in a separate text file. Hereby, the label file can either contain the actual labels for each point in the data file or only the line indices of the anomalies. Example source data files:

Data file

```
12751.0
8767.0
7005.0
5257.0
4189.0
```

Labels file (actual labels)

```
1
0
0
0
0
```

Labels file (line indices)

```
3
4
```

The script `scripts/preprocess_dataset.py` automatically generates the index column using an auto-incrementing integer value. The integer value can be substituted with a corresponding timestamp (auto-incrementing value is used as a time unit, such as seconds `s` or hours `h` from the unix epoch). See the tool documentation for further information:

```
python scripts/preprocess_dataset.py --help
```

## 2.2 Algorithms

Any algorithm that can be called with a numpy array as parameter and a numpy array as return value can be evaluated. TimeEval also supports passing only the filepath to an algorithm and let the algorithm perform the file reading and parsing. In this case, the algorithm must be able to read the *TimeEval canonical file format*. Use `data_as_file=True` as a keyword argument to the algorithm declaration.

The `main` function of an algorithm must implement the *timeeval.adapters.base.Adapter*-interface. TimeEval comes with four different adapter types described in section *Algorithm adapters*.

Each algorithm is associated with metadata including its learning type and input dimensionality. TimeEval distinguishes between the three learning types *timeeval.TrainingType.UNSUPERVISED* (default), *timeeval.TrainingType.SEMI_SUPERVISED*, and *timeeval.TrainingType.SUPERVISED* and the two input dimensionality definitions *timeeval.InputDimensionality.UNIVARIATE* (default) and *timeeval.InputDimensionality.MULTIVARIATE*.

### 2.2.1 Registering algorithms

```
from timeeval import TimeEval, DatasetManager, Algorithm
from timeeval.adapters import FunctionAdapter
from timeeval.constants import HPI_CLUSTER
import numpy as np

def my_algorithm(data: np.ndarray) -> np.ndarray:
    return np.zeros_like(data)

datasets = [("WebscopeS5", "A1Benchmark-1")]
algorithms = [
    # Add algorithms to evaluate...
    Algorithm(
        name="MyAlgorithm",
        main=FunctionAdapter(my_algorithm),
```

(continues on next page)

(continued from previous page)

```
        data_as_file=False,
    )
]

timeeval = TimeEval(DatasetManager(HPI_CLUSTER.akita_dataset_paths[HPI_CLUSTER.
↪BENCHMARK]), datasets, algorithms)
```

### 2.2.2 Algorithm adapters

Algorithm adapters allow you to use different algorithm types within TimeEval. The most basic adapter just wraps a python-function.

You can implement your own adapters. Example:

```
from typing import Optional
from timeeval.adapters.base import Adapter
from timeeval.data_types import AlgorithmParameter


class MyAdapter(Adapter):

    # AlgorithmParameter = Union[np.ndarray, Path]
    def _call(self, dataset: AlgorithmParameter, args: Optional[dict] = None) ->␣
↪AlgorithmParameter:
        # e.g. create another process or make a call to another language
        pass
```

#### Function adapter

The *timeeval.adapters.function.FunctionAdapter* allows you to use Python functions and methods as the algorithm main code. You can use this adapter by wrapping your function:

```
from timeeval import Algorithm
from timeeval.adapters import FunctionAdapter
from timeeval.data_types import AlgorithmParameter
import numpy as np

def your_function(data: AlgorithmParameter, args: dict) -> np.ndarray:
    if isinstance(data, np.ndarray):
        return np.zeros_like(data)
    else: # data = pathlib.Path
        return np.genfromtxt(data)[0]

Algorithm(
    name="MyPythonFunctionAlgorithm",
    main=FunctionAdapter(your_function),
    data_as_file=False
)
```

#### Docker adapter

The *timeeval.adapters.docker.DockerAdapter* allows you to run an algorithm as a Docker container. This means that the algorithm is available as a Docker image. This is the main adapter used for our evaluations. Usage example:

```
from timeeval import Algorithm
from timeeval.adapters import DockerAdapter

Algorithm(
    name="MyDockerAlgorithm",
    main=DockerAdapter(image_name="algorithm-docker-image", tag="latest"),
    data_as_file=True  # important here!
)
```

**Important:** Using a `DockerAdapter` implies that `data_as_file=True` in the `Algorithm` construction. The adapter supplies the dataset to the algorithm via bind-mounting and does not support passing the data as numpy array.

### 2.2.3 Experimental algorithm adapters

The algorithm adapters in this section are prototypical implementations and not fully tested with TimeEval. Some adapters were used in earlier versions of TimeEval and are not compatible to it anymore.

**Warning:** The following algorithm adapters should be used for educational purposes only. They are not fully tested with TimeEval!

#### Distributed adapter

The *timeeval.adapters.distributed.DistributedAdapter* allows you to execute an already distributed algorithm on multiple machines. Supply a list of `remote_hosts` and a `remote_command` to this adapter. It will use SSH to connect to the remote hosts and execute the `remote_command` on these hosts before starting the main algorithm locally.

**Important:**

- Password-less ssh to the remote machines required!
- **Do not combine with the distributed execution of TimeEval ("TimeEval.Distributed" using `TimeEval(.., distributed=True)`)!** This will affect the timing results.

#### Jar adapter

The *timeeval.adapters.jar.JarAdapter* lets you evaluate Java algorithms in TimeEval. You can supply the path to the Jar-File (executable) and any additional arguments to the Java-process call.

#### Adapter to apply univariate methods to multivariate data

The *timeeval.adapters.multivar.MultivarAdapter* allows you to apply an univariate algorithm to each dimension of a multivariate dataset individually and receive a single aggregated result. You can currently choose between three different result aggregation strategies that work on single points:

- *timeeval.adapters.multivar.AggregationMethod.MEAN*
- *timeeval.adapters.multivar.AggregationMethod.MEDIAN*
- *timeeval.adapters.multivar.AggregationMethod.MAX*

If `n_jobs > 1`, the algorithms are executed in parallel.

### 2.2.4 Algorithms provided with TimeEval

All algorithms that we provide with TimeEval use the *DockerAdapter* as adapter-implementation to allow you to use all features of TimeEval with them (such as resource restrictions, timeout, and fair runtime measurements). You can find the TimeEval algorithm implementations on Github: https://github.com/TimeEval/TimeEval-algorithms and can pull the images directly from the GitHub container registry. Using Docker images to bundle an algorithm for TimeEval also allows easy integration of new algorithms because there are no requirements regarding programming languages, frameworks, or tools. However, using Docker images to bundle algorithms makes preparing them for use with TimeEval a bit more cumbersome (cf. *How to integrate your own algorithm into TimeEval*). We use GitHub Actions to automatically build and publish the algorithm Docker images for direct use within TimeEval.

In this section, we describe some important aspects of this architecture.

#### TimeEval base Docker images

To benefit from Docker layer caching and to reduce code duplication (DRY!), we decided to put common functionality in so-called base images. The following is taken care of by base images:

- Provide system (OS and common OS tools)
- Provide language runtime (e.g. python3, java8)
- Provide common libraries / algorithm dependencies
- Define volumes for IO
- Define Docker entrypoint script (performs initial container setup before the algorithm is executed)

Currently, we provide the following root base images:

| Name/Folder | Image | Usage |
|---|---|---|
| python2-base | ghcr.io/timeeval/python2-base | Base image for TimeEval methods that use python2 (version 2.7); includes default python packages. |
| python3-base | ghcr.io/timeeval/python3-base | Base image for TimeEval methods that use python3 (version 3.7.9); includes default python packages. |
| python36-base | ghcr.io/timeeval/python36-base | Base image for TimeEval methods that use python3.6 (version 3.6.13); includes default python packages. |
| r4-base | ghcr.io/timeeval/r4-base | Base image for TimeEval methods that use R (version 4.0.5). |
| java-base | ghcr.io/timeeval/java-base | Base image for TimeEval methods that use Java (JRE 11.0.10). |
| rust-base | ghcr.io/timeeval/rust-base | Base image for TimeEVal methods that use Rust (Rust 1.58). |

In addition to the root base images, we also provide some derived base images (*intermediate images*) that add further common functionality to the language runtimes:

| Name/Folder | Image | Usage |
|---|---|---|
| tsmp | ghcr.io/timeeval/tsmp | Base image for TimeEval methods that use the matrix profile R package tsmp; is based on ghcr.io/timeeval/r4-base. |
| pyod | ghcr.io/timeeval/pyod | Base image for TimeEval methods that are based on the pyod library; is based on ghcr.io/timeeval/python3-base. |
| timeeval-test-algorithm | ghcr.io/timeeval/timeeval-test-algorithm | Test image for TimeEval tests that use docker; is based on ghcr.io/timeeval/python3-base; technically not a base images. |
| python3-torch | ghcr.io/timeeval/python3-torch | Base image for TimeEval methods that use python3 (version 3.7.9) and PyTorch (version 1.7.1); includes default python packages and torch; is based on ghcr.io/timeeval/python3-base. |

You can find all current base images in the timeeval-algorithms-repository under `0-base-images` and `1-intermediate-images`.

### TimeEval algorithm interface

TimeEval uses a common interface to execute all the algorithms that implement the *DockerAdapter*. This means that the algorithms' input, output, and parameterization is equal for all provided algorithms.

### Execution and parametrization

All algorithms are executed by creating a Docker container using their Docker image and then executing it. The base images take care of the container startup and they call the main algorithm file with a single positional parameter. This parameter contains a String-representation of the algorithm configuration as JSON. Example parameter JSON (2022-08-18):

```
{
  "executionType": 'train' | 'execute',
  "dataInput": string,   # example: "path/to/dataset.csv",
  "dataOutput": string,  # example: "path/to/results.csv",
  "modelInput": string,  # example: "/path/to/model.pkl",
  "modelOutput": string, # example: "/path/to/model.pkl",
```

(continues on next page)

(continued from previous page)

```
  "customParameters": dict
}
```

### Custom algorithm parameters

All algorithm hyperparameters described in the corresponding algorithm paper are exposed via the `customParameters` configuration option. This allows us to set those parameters from TimeEval.

**Warning:** TimeEval does **not** parse a `manifest.json` file to get the custom parameters' types and default values. We expect the users of TimeEval to be familiar with the algorithms, so that they can specify the required parameters manually. However, we require each algorithm to be executable without specifying any custom parameters (especially for testing purposes). Therefore, **please provide sensible default parameters for all custom parameters within the method's code**.

If you want to contribute your algorithm implementation to TimeEval, please add a `manifest.json`-file to your algorithm anyway to aid the integration into other tools and for user information.

If your algorithm does not use the default parameters automatically and expects them to be provided, your algorithm will fail during runtime if no parameters are provided by the TimeEval user.

### Input and output

Input and output for an algorithm is handled via bind-mounting files and folders into the Docker container.

All **input data**, such as the training dataset and the test dataset, are mounted read-only to the `/data`-folder of the container. The configuration options `dataInput` and `modelInput` reflect this with the correct path to the dataset (e.g. `{ "dataInput": "/data/dataset.test.csv" }`). The dataset format follows our *Canonical file format*.

All **output** of your algorithm should be written to the `/results`-folder. This is also reflected in the configuration options with the correct paths for `dataOutput` and `modelOutput` (e.g. `{ "dataOutput": "/results/anomaly_scores.csv" }`). The `/results`-folder is also bind-mounted to the algorithm container - but writable -, so that TimeEval can access the results after your algorithm finished. An algorithm can also use this folder to write persistent log and debug information.

Every algorithm must produce an **anomaly scoring** as output and put it at the location specified with the `dataOutput`-key in the configuration. The output file's format is CSV-based with a single column and no header. You can for example produce a correct anomaly scoring with NumPy's `numpy.savetxt`-function: `np.savetxt(<args.dataOutput>, arr, delimiter=",")`.

**Temporary files** and data of an algorithm are written to the current working directory (currently this is `/app`) or the temporary directory `/tmp` within the Docker container. All files written to those folders is lost after the algorithm container is removed.

#### Example calls

The following Docker command represents the way how the TimeEval *DockerAdapter* executes your algorithm image:

```
docker run --rm \
    -v <path/to/dataset.csv>:/data/dataset.csv:ro \
    -v <path/to/results-folder>:/results:rw \
    -e LOCAL_UID=<current user id> \
    -e LOCAL_GID=<groupid of akita group> \
    <resource restrictions> \
  ghcr.io/timeeval/<your_algorithm>:latest execute-algorithm '{
    "executionType": "execute",
    "dataInput": "/data/dataset.csv",
    "modelInput": "/results/model.pkl",
    "dataOutput": "/results/anomaly_scores.ts",
    "modelOutput": "/results/model.pkl",
    "customParameters": {}
  }'
```

This is translated to the following call within the container from the entry script of the base image:

```
docker run --rm \
    -v <path/to/dataset.csv>:/data/dataset.csv:ro \
    -v <path/to/results-folder>:/results:rw <...> \
  ghcr.io/timeeval/<your_algorithm>:latest bash
# now, within the container
<python | java -jar | Rscript> $ALGORITHM_MAIN '{
  "executionType": "execute",
  "dataInput": "/data/dataset.csv",
  "modelInput": "/results/model.pkl",
  "dataOutput": "/results/anomaly_scores.ts",
  "modelOutput": "/results/model.pkl",
  "customParameters": {}
}'
```

## 2.3 Parameter configuration and search

**Warning:** WIP

## 2.4 TimeEval configuration and resource restrictions

### 2.4.1 Experiments

**Important:** WIP

You can configure which algorithms are executed on which datasets - to some degree. Per default, TimeEval evaluates all algorithms on all datasets (cross product) skipping those combinations that are not compatible. You can control which experiments are generated using the parameters `skip_invalid_combinations`,

`force_training_type_match`, `force_dimensionality_match`, and `experiment_combinations_file`. Different values for those flags allow you to achieve different goals.

#### Avoiding conflicting combinations

Not all algorithms can be executed on all datasets. If the parameter `skip_invalid_combinations` is set to `True`, TimeEval will skip all invalid combinations of algorithms and datasets based on their input dimensionality and training type. It is automatically enabled if either `force_training_type_match` or `force_dimensionality_match` is set to `True` (see *next section*). Per default (`force_training_type_match == force_dimensionality_match == False`), the following combinations are not executed:

- supervised algorithms on semi-supervised or unsupervised datasets (datasets cannot be used to train the algorithm)
- semi-supervised algorithm on supervised or unsupervised datasets (datasets cannot be used to train the algorithm)
- univariate algorithms on multivariate datasets (algorithm cannot process the dataset)

#### Forcing property matches

force_training_type_match Narrow down the algorithm-dataset combinations further by executing an algorithm only on datasets with **the same** training type, e.g. unsupervised algorithms only on unsupervised datasets. This flag implies `skip_invalid_combinations==True`.

force_dimensionality_match Narrow down the algorithm-dataset combinations furthter by executing an algorithm only on datasets with **the same** input dimensionality, e.g. multivariate algorithms only on multivariate datasets. This flag implies `skip_invalid_combinations==True`.

#### Selecting specific combinations

You can use the parameter `experiment_combinations_file` to supply a path to an experiment combinations CSV-File. Using this file, you can specify explicitly which combinations of algorithms, datasets, and hyperparameters should be executed. The file should contain CSV data with a single header line and four columns with the following names:

1. `algorithm` - name of the algorithm
2. `collection` - name of the dataset collection
3. `dataset` - name of the dataset
4. `hyper_params_id` - ID of the hyperparameter configuration

Only experiments that are present in the TimeEval configuration **and** this file are scheduled and executed. This allows you to circumvent the cross-product that TimeEval will perform in its default configuration.

### 2.4.2 Resource restrictions

The competitive evaluation of algorithms requires that all algorithms are executed in the same (or at least very similar) execution environment. This means that no algorithm should have an unfair advantage over the other algorithms by having more time, memory, or other compute resource available.

TimeEval ensures comparable execution environments for all algorithms by executing algorithms in isolated Docker containers and controlling their resources. When configuring TimeEval, you can specify resource limits that apply to all evaluated algorithms in the same way. This includes the number of CPUs, main memory, training, and execution time limits. All those resources can be specified using a *`timeeval.ResourceConstraints`* object.

**Important:** To make use of resource restrictions, all evaluated algorithms must be registered using the *timeeval.adapters.docker.DockerAdapter*. It is the only adapter implementation that can deal with resource constraints. All other adapters ignore them.

TimeEval will raise an error if you try to use resource restrictions and non-`DockerAdapter`-based algorithms at the same time.

### Time limits

Some algorithms are not suitable for very large datasets and, thus, can take a long time until they finish either training or testing. For this reason, TimeEval uses timeouts to restrict the runtime of all (or selected) algorithms. You can change the timeout values for the training and testing phase globally using configuration options in *timeeval.ResourceConstraints*:

```
from durations import Duration
from timeeval import TimeEval, ResourceConstraints

limits = ResourceConstraints(
    train_timeout=Duration("2 hours"),
    execute_timeout=Duration("2 hours"),
)
timeeval = TimeEval(dm, datasets, algorithms,
    resource_constraints=limits
)
...
```

It's also possible to use different timeouts for specific algorithms if they run using the `DockerAdapter`. The `DockerAdapter` class can take in a `timeout` parameter that defines the maximum amount of time the algorithm is allowed to run. The parameter takes in a `durations.Duration` object as well, and overwrites the globally set timeouts. If the timeout is exceeded, a *timeeval.adapters.docker.DockerTimeoutError* is raised and the specific algorithm for the current dataset is cancelled.

### CPU and memory limits

To facilitate a fair comparison of algorithms, you can configure TimeEval to restrict the execution of algorithms to specific resources. At the moment, TimeEval supports limiting the number of CPUs and the available memory. GPUs are not supported by TimeEval. The resource constraints are enforced using explicit resource limits on the Docker containers.

In the following example, we limit each algorithm to 1 CPU core and a maximum of 3 GB of memory:

```
from timeeval import ResourceConstraints
from timeeval.resource_constraints import GB

limits = ResourceConstraints(
    task_memory_limit = 3 * GB,
    task_cpu_limit = 1.0
)
```

If TimeEval is executed on a distributed cluster, it assumes a homogenous cluster, where all nodes of the cluster have the same capabilities and resources. There are two options to configure resource limits for distributed TimeEval:

1. **automatically**: Per default, TimeEval will distribute the available resources of each node evenly to all parallel evaluation tasks. By changing the number of tasks per hosts, you can, thus, easily control the available resources to the evaluation tasks without worrying about over-provisioning.

   Because each tasks, in effect, trains or executes a time series anomaly detection algorithm, the tasks are resource-intensive, over-provisioning should be prevented. It could decrease overall performance and distort runtime measurements.

   However, if your compute cluster is not homogenous, TimeEval will assign different resource limits to algorithms depending on the node where the algorithm is executed on.

   **Example**: Non-homogenous cluster of 2 nodes with the first node $A$ having 10 cores and 30 GB of memory and the second node $B$ having 20 cores and 60 GB of memory. When setting the limits to `ResourceConstraints(tasks_per_hosts=10)`, algorithms executed on node $A$ will get 1 CPU and 3 GB of memory and algorithms executed on node $B$ will get 2 CPUs and 20 GB of memory. Therefore, always use explicit resource constraints for non-homogeneous clusters.

2. **explicitly**: To make sure that all algorithms have the same resource constraints and there is no over-provisioning, you should set the CPU und memory limits explicitly. For non-homogenous cluster take the node with the lowest overall resources and decide on how many task you want to execute in parallel. Then divide the available resources by the number of tasks and fix the resource limits for all algorithms to these numbers.

   **Example**: The same non-homogenous cluster with nodes $A$ and $B$ with 10 tasks per node (host), would result in the following constraints:

```
rcs = ResourceConstraints(
    tasks_per_host=10,
    task_memory_limit = 3 * GB,
    task_cpu_limit = 1.0,
)
```

## 2.5 TimeEval results

On configuring and executing TimeEval, TimeEval applies the algorithms with their configured hyperparameter values on all the datasets. It measures the algorithms' runtimes and checks their effectiveness using evaluation measures (*metrics*). The results are stored in a summary file called `results.csv` and a nested folder structure in the results-folder (`./results/<timestamp>` per default). The output directory has the following structure:

```
results/<timestamp>/
├── results.csv
├── <algorithm_1>/<hyper_params_id>/
|   ├── <collection_name>/<dataset_name_1>/<repetition_number>/
|   |   ├── raw_anomaly_scores.ts
|   |   ├── anomaly_scores.ts
|   |   ├── docker-algorithm-scores.csv
|   |   ├── execution.log
|   |   ├── hyper_params.json
|   |   └── metrics.csv
|   └── <collection_name>/<dataset_name_2>/<repetition_number>/
|       ├── raw_anomaly_scores.ts
|       ├── anomaly_scores.ts
|       ├── docker-algorithm-scores.csv
|       ├── execution.log
|       ├── model.pkl
```

(continues on next page)

(continued from previous page)

```
│       ├── hyper_params.json
│       └── metrics.csv
└── <algorithm_2>/<hyper_params_id>/
    └── <collection_name>/<dataset_name_1>/<repetition_number>/
        ├── raw_anomaly_scores.ts
        ├── anomaly_scores.ts
        ├── docker-algorithm-scores.csv
        ├── execution.log
        ├── hyper_params.json
        └── metrics.csv
```

We provide a description of each file below.

### 2.5.1 Summary file (`result.csv`)

For a given dataset, different algorithms with varying hyperparameters yield distinct results. The file `result.csv` provides an overview of the evaluation run and contains the following attributes:

| Column Name | Datatype | Description |
|---|---|---|
| algorithm | str | name of the algorithm as defined in *Algorithm*#name attribute |
| collection | str | name of the dataset collection. A collection contains similar datasets. |
| dataset | str | name of the dataset |
| algo_training_type | str | specifies, whether a dataset has a training time series with anomaly labels (supervised), with normal data only (semi-supervised), or no training time series at all (unsupervised) |
| algo_input_dimensionality | str | specifies if the dataset has multiple channels (multivariate) or not (univariate) |
| dataset_training_type | str | specifies, whether an algorithm requires training data with anomalies (supervised), without normal data only (semi-supervised), or does not require training data (unsupervised) |
| dataset_input_dimensionality | str | univariate or multivariate (see above) |
| train_preprocess_time | float | runtime of the preprocessing step during training in seconds |
| train_main_time | float | runtime of the training in seconds (does not include pre-processing time) |
| execute_preprocess_time | float | runtime of the preprocessing step during execution in seconds |
| execute_main_time | float | runtime of the execution of the algorithm on the test time series in seconds (does not include pre- or post-processing times) |
| execute_postprocess_time | float | runtime of the post-processing step during execution |
| status | str | specifies, whether the algorithm executed successfully (*OK*), exceeded the time limit (*TIMEOUT*), exceeded the memory limit (*OOM*), or failed (*ERROR*) |
| error_message | str | optional detailed error message |
| repetition | int | repetition number if a dataset-hyperparameter-dataset combination was executed multiple times |
| hyper_params | float | actual hyperparameter values for this execution |
| hyper_params_id | float | alphanumerical hash of the hyperparameter configuration |
| metric_1 | float | value of the first performance metric |
| … | … | … |

### 2.5.2 Directory (<algorithm_1>/<hyper_params_id>/<collection_name>/<dataset_name_1>/<repetition_number>/)

For every experiment in the configured evaluation run, TimeEval creates a new directory in the result folder. It stores all the results and temporary files for this single combination of dataset, algorithm, algorithm hyperparameter values, and repetition number. The directories are structured in nested folders named by first the algorithm name, followed by the ID of the hyperparameter settings, the dataset collection name, the dataset name, and finally the repetition number. Each experiment directory contains at least the following files:

- `raw_anomaly_scores.ts`: The raw anomaly scores produced by the algorithm after the post-processing function was executed. The file contains no header and a single floating point value in each row for each time step of the input time series. The value range depends on the algorithm.
- `anomaly_scores.ts`: Normalized anomaly scores. The value range is from 0 (normal) to 1 (most anomalous).
- `execution.log`: Unstructured log-file of the experiment execution. Contains debugging information from the Adapter, the algorithm, and the metric calculation. If an algorithm fails, its error message usually appear in this log.
- `metrics.csv`: This file lists the metric and runtime measurements for the corresponding experiment. The used metrics are defined by the user. Find more information in the API documentation: *timeeval.metrics package*
- `hyper_params.json`: Contains a JSON-object with the hyperparameter values used to execute the algorithm on the dataset. If hyperparameter heuristics were defined, the heuristic' values are already resolved.

All other files are optional and depend on the used algorithm *`Adapter`*. For example, the *`DockerAdapter`* usually produces a temporary file called `docker-algorithm-scores.csv` to pass the algorithm result from the Docker container to TimeEval, and (semi-)supervised algorithms store their trained model in `model.pkl`-files.

## 2.6 Metrics

> **Warning:** WIP
>
> Please consult the API documentation for the *available Metrics* in TimeEval.

## 2.7 Distributed TimeEval

TimeEval is able to run multiple experiments in parallel and distributedly on a cluster of multiple machines (hosts). You can enable the distributed execution of TimeEval by setting `distributed=True` when creating the *`TimeEval`* object. The cluster configuration can be managed using a *`timeeval.RemoteConfiguration`* object passed to the `remote_config` argument.

Distributed TimeEval will use your supplied configuration of algorithms, parameters, and datasets to create a list of experiments or evaluation tasks. It then schedules the execution of these tasks to the nodes in the cluster so that the full cluster can be utilized for the evaluation run. During the run, the main process monitors the execution of the tasks. At the end of the run, it collects the evaluation results, so that you can use them for further analysis.

### 2.7.1 Host roles

TimeEval uses `Dask's SSHCluster` to dynamically create a logical cluster on a list of hosts specified by IP-addresses or hostnames. According to Dask's terminology, TimeEval also distinguishes between the host roles *scheduler* and *worker*. In addition, there also is a *driver* role. The following table summarizes the host roles:

| Host Role | Description |
|---|---|
| *driver* | Host that runs the experiment script (where `python experiment-script.py` is called). |
| *scheduler* | Host that runs the `dask.Scheduler` that coordinates worker processes and distributes the tasks and jobs to the *workers*. |
| *worker* | Host that runs one or multiple `dask.Worker`s and receives tasks and jobs. The *workers* perform the actual computations. |

The *driver* host role is implicit. The host, on which you create the *TimeEval*-object, gets the *driver* role. In the distributed mode, this main Python-process does not execute any evaluation jobs. The *scheduler* and *worker* roles can be assigned to a host using the *RemoteConfiguration* object.

The *driver* host could be a local notebook or computer, while the *scheduler* and *worker* hosts are part of the cluster. A single host can have multiple roles at the same time, and for most use cases this is totally fine. Usually, a single machine of a cluster is used as *driver*, *scheduler*, and *worker*, while all the other machines just get the *worker* role. That is typically not a problem because the *driver* and *scheduler* components do not use many resources, and we can, therefore, use the computing power of the first host much more efficiently.

**Example**:

Assume that we have a cluster of 3 nodes (*node1*, *node2*, *node3*) and that we start a TimeEval experiment on *node1* with the following configuration:

```
from timeeval import RemoteConfiguration
RemoteConfiguration(
    scheduler_host="node1",
    worker_hosts=["node1", "node2", "node3"]
)
```

In this case, *node1* executes TimeEval (role *driver*), hosts the Dask scheduler (role *scheduler*), and participates in the execution of evaluation jobs (role *worker*). It has all three roles. *node2* and *node3*, however, are pure work horses and participate in the execution of evaluation jobs only (role *worker*).

### 2.7.2 Distributed execution

If TimeEval is started with `distributed=True`, it automatically starts a Dask SSHCluster on the specified *scheduler* and *worker* hosts. This is done via simple SSH-connections to the machines. It then uses the passed experiment configurations to create evaluation jobs (called `Experiments`). Each `Experiment` consists of an algorithm, its hyper-parameters, a dataset, and a repetition number. After all `Experiments` have been generated and validated, they are sent to the Dask *scheduler* and put into a task queue. The *workers* pull the tasks from the *scheduler* and perform the evaluation (i.a., executing the Docker containers of the algorithm). All results and temporary data are stored on the disk of the local node and the overall evaluation result is sent back to the scheduler. The *driver* host periodically polls the *scheduler* for the results and collects them in memory. When all tasks have been processed, the *driver* uses SSH connections again to pull all the temporary data from the *worker* nodes. This populates the local `results`-folder.

### 2.7.3 Cluster requirements

Because we use a `Dask SSHCluster` to manage the cluster hosts, there are additional requirements for every cluster node. Please ensure that your cluster setup meets the following requirements:

- Every node must have Python and Docker installed.
- The algorithm images must be present on all nodes or Docker must be able to pull them (if `skip_pull=False`).
- Every node uses the same Python environment (the path to the Python-binary must be the same) and has TimeEval installed in it.
- The whole datasets' folder must be present on all nodes at the same path. This means `DatasetManager("path/to/datasets-folder", create_if_missing=False)` must work on all nodes.
- Your Python script with the experiment configuration does not import any other **local** files (e.g., `from .util import xyz`).
- All hosts must be able to reach each other via network.
- The *driver* host must be able to open a SSH connection to all the other nodes using **passwordless SSH**. For this, please confirm that you can run `ssh <remote_host>` without any (password-)prompt; otherwise, TimeEval will not be able to reach the other nodes. (https://www.google.com/search?q=passwordless+SSH)

CHAPTER

# THREE

# API REFERENCE

This section documents the public API of TimeEval.

## 3.1 timeeval package

An Evaluation Tool for Anomaly Detection Algorithms on Time Series Data.

**How to use the documentation:**

Documentation is available in two forms: docstrings provided with the code, and a loose standing reference guide, available on timeeval.readthedocs.io.

Code snippets in the docstring examples are indicated by three greater-than signs:

```
>>> x = 42
>>> x = x + 1
```

Use the built-in `help` function to view a function's or class' docstring:

```
>>> from timeeval import TimeEval
>>> help(TimeEval)
...
```

### 3.1.1 timeeval.TimeEval

**class** timeeval.**TimeEval**(*dataset_mgr: Datasets*, *datasets: List[Tuple[str, str]]*, *algorithms: List[Algorithm]*, *results_path: Path = PosixPath('results')*, *repetitions: int = 1*, *distributed: bool = False*, *remote_config: Optional[RemoteConfiguration] = None*, *resource_constraints: Optional[ResourceConstraints] = None*, *disable_progress_bar: bool = False*, *metrics: Optional[List[Metric]] = None*, *skip_invalid_combinations: bool = True*, *force_training_type_match: bool = False*, *force_dimensionality_match: bool = False*, *n_jobs: int = - 1*, *experiment_combinations_file: Optional[Path] = None*, *module_configs: Mapping[str, Any] = {}*)

Main class of TimeEval.

This class is the main utility to configure and execute evaluation experiments. First select your algorithms and datasets and, then, pass them to TimeEval and use its constructor arguments to configure your evaluation run. Per default, TimeEval evaluates all algorithms on all datasets (cross product). You can use the parameters `skip_invalid_combinations`, `force_training_type_match`, `force_dimensionality_match`, and `experiment_combinations_file` to control which algorithm runs on which dataset. See the description of the other arguments for further configuration details.

After you have created your TimeEval object, holding the experiment run configuration, you can execute the experiments by calling *run()*. Afterward, the evaluation summary results are accessible in the `results_path` and from *get_results()*.

### Examples

Simple example experiment evaluating a single algorithm on the test datasets using the default metrics (just `ROC_AUC`):

```
>>> from timeeval import TimeEval, DefaultMetrics, Algorithm, TrainingType,␣
↪InputDimensionality, DatasetManager
>>> from timeeval.adapters import DockerAdapter
>>> from timeeval.params import FixedParameters
>>>
>>> dm = DatasetManager(Path("tests/example_data"))
>>> datasets = dm.select()
>>>
>>> algorithms = [
>>>     Algorithm(
>>>         name="COF",
>>>         main=DockerAdapter(image_name="ghcr.io/timeeval/cof"),
>>>         param_config=FixedParameters({"n_neighbors": 20, "random_state": 42}),
>>>         data_as_file=True,
>>>         training_type=TrainingType.UNSUPERVISED,
>>>         input_dimensionality=InputDimensionality.MULTIVARIATE
>>>     ),
>>> ]
>>>
>>> timeeval = TimeEval(dm, datasets, algorithms, metrics=DefaultMetrics.default_
↪list())
>>> timeeval.run()
>>> results = timeeval.get_results(aggregated=False)
>>> print(results)
```

**Parameters**

- **dataset_mgr** (*Datasets*) – The dataset manager provides the metadata about the datasets. You can either use a *DatasetManager* or a *MultiDatasetManager*.
- **datasets** (`List[Tuple[str, str]]`) – List of dataset IDs consisting of collection name and dataset name to uniquely identify each dataset. The datasets must be known by the `dataset_mgr`. You can call *select()* on the `dataset_mgr` to get a list of dataset IDs.
- **algorithms** (`List[Algorithm]`) – List of algorithms to evaluate on the datasets. The algorithm specification also contains the hyperparameter configurations that TimeEval will test.
- **results_path** (`Path`) – Use this parameter to change the path where all evaluation results are stored. If TimeEval is used in distributed mode, this path will be created on **all** nodes!
- **repetitions** (`int`) – Execute each unique combination of dataset, algorithm, and hyperparameter-setting multiple times. This allows you to use TimeEval to measure runtimes more precisely by aggregating the runtime measurements over multiple repetitions.
- **distributed** (`bool`) – Run TimeEval in distributed mode. In this case, you **should** also supply a `remote_config`.

- **remote_config** (Optional[RemoteConfiguration]) – Configuration of the Dask cluster used for distributed execution of TimeEval. See *RemoteConfiguration* for details.
- **resource_constraints** (Optional[ResourceConstraints]) – You can supply a *ResourceConstraints*-object to limit the amount of (CPU, memory, or runtime) resources available to each experiment. These options apply to each experiment to ensure a fair comparison.

  **Warning:** Resource constraints are currently only implemented by the *DockerAdapter*. If you rely on resource constraints, please make sure that all algorithms use the DockerAdapter-implementation.

- **disable_progress_bar** (bool) – Enable / disable showing the tqdm progress bars.
- **metrics** (Optional[List[Metric]]) – Supply a list of Metric to evaluate the algorithms with. TimeEval computes all supplied metrics over all experiments. If you don't specify any metric (None), the default metric list default_list() is used instead.
- **skip_invalid_combinations** (bool) – Not all algorithms can be executed on all datasets. If this flag is set to True, TimeEval will skip all invalid combinations of algorithms and datasets based on their input dimensionality and training type. It is automatically enabled if either force_training_type_match or force_dimensionality_match is set to True. Per default (force_training_type_match == force_dimensionality_match == False), the following combinations are not executed:
  - supervised algorithms on semi-supervised or unsupervised datasets (datasets cannot be used to train the algorithm)
  - semi-supervised algorithm on supervised or unsupervised datasets (datasets cannot be used to train the algorithm)
  - univariate algorithms on multivariate datasets (algorithm cannot process the dataset)
- **force_training_type_match** (bool) – Narrow down the algorithm-dataset combinations further by executing an algorithm only on datasets with **the same** training type, e.g. unsupervised algorithms only on unsupervised datasets. This flag implies skip_invalid_combinations==True.
- **force_dimensionality_match** (bool) – Narrow down the algorithm-dataset combinations furthter by executing an algorithm only on datasets with **the same** input dimensionality, e.g. multivariate algorithms only on multivariate datasets. This flag implies skip_invalid_combinations==True.
- **n_jobs** (int) – Set the number of jobs / processes used to fetch the results from the remote machine. This setting is used only in distributed mode. -1 instructs TimeEval to use all locally available cores.
- **experiment_combinations_file** (Optional[Path]) – Supply a path to an experiment combinations CSV-File. Using this file, you can specify explicitly which combinations of algorithms, datasts, and hyperparameters should be executed. The file should contain CSV data with a single header line and four columns with the following names:
  1. *algorithm* - name of the algorithm
  2. *collection* - name of the dataset collection
  3. *dataset* - name of the dataset
  4. *hyper_params_id* - ID of the hyperparameter configuration

Only experiments that are present in the TimeEval configuration **and** this file are scheduled and executed. This allows you to circumvent the cross-product that TimeEval will perform in its default configuration.

- **module_configs** (Mapping[str, Any], *optional*) – Use this parameter to pass additional configuration options for automatically loaded TimeEval modules. This is currently used only for the implementation of the Bayesian hyperparameter optimization prozedure using Optuna. See *timeeval.integration.optuna.OptunaModule* and timeeval.params.bayesian.OptunaParameterSearch for details.

  You can access loaded modules via the modules attribute (Dict[str, TimeEvalModule]) of the TimeEval instance, e.g. timeeval.modules["optuna"].

**DEFAULT_RESULT_PATH = PosixPath('results')**

Default path for the results.

If you don't specify the results_path, TimeEval will store the evaluation results in the folder results within the current working directory.

**RESULT_KEYS = ['algorithm', 'collection', 'dataset', 'algo_training_type', 'algo_input_dimensionality', 'dataset_training_type', 'dataset_input_dimensionality', 'train_preprocess_time', 'train_main_time', 'execute_preprocess_time', 'execute_main_time', 'execute_postprocess_time', 'status', 'error_message', 'repetition', 'hyper_params', 'hyper_params_id']**

This list contains all the _fixed_ result data frame's column headers. TimeEval dynamically adds the metrics and execution times depending on its configuration.

For metrics, their name() will be used as column header, and TimeEval will add the following runtime measurements depending on whether they are applicable to the algorithms in the run or not:

- train_preprocess_time: if *preprocess()* is defined
- train_main_time: if the algorithm is semi-supervised or supervised
- execute_preprocess_time: if *preprocess()* is defined
- execute_main_time: always
- execute_postprocess_time: if *postprocess()* is defined

**get_results**(*aggregated: bool = True*, *short: bool = True*) → DataFrame

Return the (aggregated) evaluation results of a previous evaluation run.

The results are returned in a Pandas DataFrame and contain the mean runtime and metrics of the algorithms for each dataset. You can tweak the output using the parameters.

**Note:** Must be called after *run()*, otherwise the returned DataFrame is empty.

**Parameters**

- **aggregated** (bool) – If True, returns the aggregated results (controled by parameter short), otherwise all collected information is returned.
- **short** (bool) – This parameter is used only in aggregation mode and controls the aggregation level and functions. If True, the aggregation is over algorithms and datasets, and the mean of the metrics, training time, and execution time is returned. If False, the aggregation is over algorithms, datasets, and parameter combinations, and the mean and standard deviation of all runtime measurements and metrics are computed.

**Return type**
DataFrame containing the evaluation results.

**rsync_results**() → None

Fetches the evaluation results of the current evaluation run from all remote machines merging the temporary data and results together on the local host. This method is automatically executed by TimeEval at the end of an evaluation run started by calling *run()*.

**See also:**

*timeeval.TimeEval.rsync_results_from*

**static rsync_results_from**(*results_path: Path*, *hosts: List[str]*, *disable_progress_bar: bool = False*, *n_jobs: int = -1*) → None

Fetches evaluation results of an independent TimeEval run from remote machines merging the temporary data and results together on the local host.

**Parameters**

- **results_path** (Path) – Path to the evaluation results. Must be the same for all hosts.
- **hosts** (List[str]) – List of hostnames or IP addresses that took part in the evaluation run.
- **disable_progress_bar** (bool) – If a progress bar should be displayed or not.
- **n_jobs** (int) – Number of parallel processes used to fetch the results. The parallelism is limited by the number of external hosts and the maximum number of available CPU cores.

**run**() → None

Starts the configured evaluation run.

Each TimeEval run consists of a number of experiments that are executed independently of each other. There are three phases: PREPARE, EVALUATION, FINALIZE.

1. _PREPARE_ phase: In the first phase, the execution environment is prepared, the result folder is created, and algorithm adapter-dependent preparation steps, such as pulling Docker images for the *DockerAdapter*, are executed.
2. _EVALUATION_ phase: In the evaluation phase, the experiments are executed and the results are recorded and stored to disk.
3. _FINALIZE_ phase: In the last phase, the execution environment is cleaned up, and algorithm adapter-dependent finalization steps, such as removing the temporary Docker containers for the *DockerAdapter*, are executed.

This method executes all three phases after each other and returns after they are finished. You can access the evaluation results either using *get_results()* programmatically or in the results folder from the file system.

**save_results**(*results_path: Optional[Path] = None*) → None

Store the evaluation results to a CSV-file in the provided *results_path*. This method is automatically executed by TimeEval at the end of an evaluation run when calling *run()*.

**Parameters**
**results_path** (Optional[Path]) – Path, where the results should be stored at. If it is not supplied, the results path of the current TimeEval run (timeeval.TimeEval.results_path) is used.

### 3.1.2 timeeval.Status

class timeeval.**Status**(*value*)

Bases: Enum

Status of an experiment.

The status of each evaluation experiment can have one of four states: ok, error, timeout, or out-of-memory (oom).

**ERROR = 1**

**OK = 0**

**OOM = 3**

**TIMEOUT = 2**

### 3.1.3 timeeval.Algorithm

class timeeval.**Algorithm**(*name: str*, *main: Adapter*, *preprocess: Optional[TSFunction] = None*, *postprocess: Optional[TSFunctionPost] = None*, *data_as_file: bool = False*, *param_schema: Dict[str, Dict[str, Any]] = <factory>*, *param_config: ParameterConfig = <timeeval.params.base.FixedParameters object>*, *training_type: TrainingType = TrainingType.UNSUPERVISED*, *input_dimensionality: InputDimensionality = InputDimensionality.UNIVARIATE*)

This class is a wrapper for any *Adapter* and an instruction plan for the TimeEval tool. It tells TimeEval what algorithm to execute, what pre- and post-steps to perform and how the parameters and data are provided to the algorithm. Moreover, it defines attributes that are necessary to help TimeEval know what kind of time series can be put into the algorithm.

**Parameters**

- **name** (str) – The name of the Algorithm shown in the results.
- **main** (*timeeval.adapters.base.Adapter*) – The adapter implementation that contains the algorithm to evaluate.
- **preprocess** (Optional[TSFunction]) – Optional function to perform before main to modify input data.
- **postprocess** (Optional[TSFunctionPost]) – Optional function to perform after main to modify output data.
- **data_as_file** (bool) – Whether the data input is a Path or a numpy.ndarray.
- **param_schema** (Dict[str, Dict[str, Any]]) – Optional schema of the algorithm's input parameters needed by timeeval_experiments.algorithm_configurator.AlgorithmConfigurator. Schema definition:

```
[
    "param_name": {
        "name": str
        "defaultValue": Any
        "description": str
        "type": str
    },
]
```

- **param_config** (*timeeval.params.ParameterConfig*) – Optional object of type ParameterConfig to define a search grid or fixed parameters.
- **training_type** (*timeeval.data_types.TrainingType*) – Definition of training type to receive the correct dataset formats (needed if TimeEval is run with `force_training_type_match` config).
- **input_dimensionality** (*timeeval.data_types.InputDimensionality*) – Definition of training type to receive the correct dataset formats (needed if TimeEval is run with `force_dimensionality_match` config option).

#### Examples

Create a baseline algorithm that always assigns a normal anomaly score:

```
>>> import numpy as np
>>> from timeeval import Algorithm
>>> from timeeval.adapters import FunctionAdapter
>>> my_fn = lambda X, args: np.zeros(len(X))
>>> Algorithm(name="Test Algorithm", main=FunctionAdapter(my_fn), data_as_
↪file=False)
```

**data_as_file:** `bool` = **False**

**input_dimensionality:** *InputDimensionality* = **'univariate'**

**main:** *Adapter*

**name:** `str`

**param_config:** *ParameterConfig* = **<timeeval.params.base.FixedParameters object>**

**param_schema:** **Dict[str, Dict[str, Any]]**

**postprocess:** **Optional[TSFunctionPost] = None**

**preprocess:** **Optional[TSFunction] = None**

**training_type:** *TrainingType* = **'unsupervised'**

### 3.1.4 timeeval.InputDimensionality

**class** timeeval.**InputDimensionality**(*value*)

Bases: Enum

Input dimensionality supported by an algorithm or of a dataset.

TimeEval distinguishes between univariate and multivariate datasets / time series.

**MULTIVARIATE = 'multivariate'**

Multivariate datasets have 2 or more features/dimensions/channels.

A multivariate algorithm can process univariate or multivariate datasets.

**UNIVARIATE = 'univariate'**

Univariate datasets consist of a single feature/dimension/channel.

An univariate algorithm can process only a dataset with a single feature/dimension/channel.

**static from_dimensions**(*n: int*) → *InputDimensionality*

Converts the feature/dimension/channel count to an Enum-object.

### 3.1.5 timeeval.TrainingType

**class** timeeval.**TrainingType**(*value*)

Bases: Enum

Training type of algorithm or dataset.

TimeEval distinguishes between unsupervised, semi-supervised, and supervised algorithms.

**SEMI_SUPERVISED = 'semi-supervised'**

A semi-supervised algorithm requires normal data for training.

A semi-supervised dataset consists of a training time series with normal data (no anomalies; all labels are 0) and a test time series.

**SUPERVISED = 'supervised'**

A supervised algorithm requires training data with anomalies.

A supervised dataset consists of a training time series with anomalies and a test time series.

**UNSUPERVISED = 'unsupervised'**

An unsupervised algorithm does not require any training data.

An unsupervised dataset consists only of a single test time series.

**static from_text**(*name: str*) → *TrainingType*

Converts the string-representation to an Enum-object.

### 3.1.6 timeeval.RemoteConfiguration

**class** timeeval.**RemoteConfiguration**(*scheduler_host: str = 'localhost'*, *scheduler_port: int = 8786*, *worker_hosts: ~typing.List[str] = <factory>*, *remote_python: str = <factory>*, *kwargs_overwrites: ~typing.Dict[str, ~typing.Any] = <factory>*, *dask_logging_file_level: str = 'INFO'*, *dask_logging_console_level: str = 'INFO'*, *dask_logging_filename: str = 'dask.log'*)

This class holds the configuration for distributed TimeEval.

TimeEval uses a dask.distributed.SSHCluster to distribute the evaluation tasks to multiple compute nodes. Please read the Dask documentation carefully and then use the constructor arguments to setup a TimeEval cluster.

**Parameters**

- **scheduler_host** (str) – IP address or hostname for the distributed.Scheduler. This node will be responsible to coordinate the cluster. The scheduler does not perform any evaluations.
- **scheduler_port** (int) – Port for the scheduler.

- **worker_hosts** (List[str]) – List of IP address or hostnames for the distributed.Worker. These nodes will execute the evaluation tasks.
- **remote_python** (str) – Path to the Python-executable. If you set up all your nodes in the same way, the default is fine.
- **kwargs_overwrites** (dict) – Use this option to overwrite any configuration options of the SSHCluster.

  **Warning:** Only use if you know what you are doing!

- **dask_logging_file_level** (str) – Logging level for the file-based Dask logger.
- **dask_logging_console_level** (str) – Logging level for the console-based Dask logger.
- **dask_logging_filename** (str) – Name of the Dask logging file without any parent paths. Each node will write its own logging file and TimeEval will automatically postfix the filenames with the hostname and place the Dask logging files into the `results_path`.

### Examples

Two-node cluster where the first node hosts the scheduler but also takes part in the evaluation:

```
>>> from timeeval import TimeEval, RemoteConfiguration
>>> config = RemoteConfiguration(scheduler_host="192.168.1.1", worker_hosts=["192.
↪168.1.1", "192.168.1.2"])
>>> TimeEval(dm=..., datasets=[], algorithms=[], distributed=True, remote_
↪config=config)
```

**dask_logging_console_level:** *str* = 'INFO'

**dask_logging_file_level:** *str* = 'INFO'

**dask_logging_filename:** *str* = 'dask.log'

**kwargs_overwrites:** *Dict[str, Any]*

**remote_python:** *str*

**scheduler_host:** *str* = 'localhost'

**scheduler_port:** *int* = 8786

**worker_hosts:** *List[str]*

## 3.1.7 timeeval.ResourceConstraints

**class** timeeval.**ResourceConstraints**(*tasks_per_host: int = 1*, *task_memory_limit: ~typing.Optional[int] = None*, *task_cpu_limit: ~typing.Optional[float] = None*, *train_timeout: ~durations.duration.Duration = <Duration 8 hours>*, *execute_timeout: ~durations.duration.Duration = <Duration 8 hours>*, *use_preliminary_model_on_train_timeout: bool = True*, *use_preliminary_scores_on_execute_timeout: bool = True*)

Use this class to configure resource constraints and how TimeEval deals with preliminary results.

> **Warning:** Resource constraints are just supported by the *DockerAdapter*!

For docker: Swap is always disabled. Resource constraints are enforced using explicit resource limits on the Docker container.

**Parameters**

- **tasks_per_host** (int) – Specify, how many evaluation tasks are executed on each host. This setting influences the default memory and CPU limits if *task_memory_limit* and *task_cpu_limit* are None: the available resources of the node are shared equally between the tasks.

  Because each tasks, in effect, trains or executes a time series anomaly detection algorithm, the tasks are resource-intensive, which means that over-provisioning is not useful and could decrease overall performance. If runtime measurements are taken, **make sure that no resources are shared between the tasks**!
- **task_memory_limit** (Optional[int]) – Specify the maximum allowed memory in Bytes. You can use *MB* and *GB* for better readability. This setting limits the available main memory per task to a fixed value.
- **task_cpu_limit** (Optional[float]) – Specify the maximum allowed CPU usage in fractions of CPUs (e.g. 0.25 means: only use 1/4 of a single CPU core). Usually, it is advisable to use whole CPU cores (e.g. 1.0 for 1 CPU core, 2.0 for 2 CPU cores, etc.).
- **train_timeout** (Duration) – Default timeout for training an algorithm. This value can be overridden for each algorithm in its *DockerAdapter*.
- **execute_timeout** (Duration) – Default timeout for executing an algorithm. This value can be overridden for each algorithm in its *DockerAdapter*.
- **use_preliminary_model_on_train_timeout** (bool) – If this option is enabled (default), then algorithms can save preliminary models (model checkpoints) to disk and TimeEval will use the last preliminary model if the training step runs into the training timeout. This is especially useful for machine learning algorithms that use an iterative training process (e.g. using SGD). As long as the algorithm implementation stores the best-so-far model after each training epoch, the training must not be limited by the number of epochs but just by the training time.
- **use_preliminary_scores_on_execute_timeout** (bool) – If this option is enabled (default) and an algorithm exceeds the execution timeout, TimeEval will look for any preliminary result. This allows the evaluation of progressive algorithms that output a rough result, refine it over time, and would otherwise run into the execution timeout.

**static default_constraints**() → *ResourceConstraints*

Creates a configuration object with the default resource constraints.

**execute_timeout: Duration = <Duration 8 hours>**

**get_compute_resource_limits**(*memory_overwrite: Optional[int] = None*, *cpu_overwrite: Optional[float] = None*) → Tuple[int, float]

Calculates the resource constraints for a single task.

There are three sources for resource limits (in decreasing priority):

1. Overwrites (passed to this function as arguments)

2. Explicitly set resource limits (on this object using *task_memory_limit* and *task_cpu_limit*)
3. Default resource constraints

**Overall default**:

1 task per node using all available cores and RAM (except small margin for OS).

When multiple tasks are specified, the resources are equally shared between all concurrent tasks. This means that CPU limit is set based on node CPU count divided by the number of tasks and memory limit is set based on total memory of node minus 1 GB (for OS) divided by the number of tasks.

> **Attention:** Must be called on the node that will execute the task!

**Parameters**

- **memory_overwrite** (int) – If this is set, it will overwrite the memory limit.
- **cpu_overwrite** (float) – If this is set, it will overwrite the CPU limit.

**Returns**

**memory_limit, cpu_limit** – Tuple of memory and CPU limit. Memory limit is expressed in Bytes and CPU limit is expressed in fractions of CPUs (e.g. 0.25 means: only use 1/4 of a single CPU core).

**Return type**

Tuple[int,float]

**get_execute_timeout**(*timeout_overwrite: Optional[Duration] = None*) → Duration

Returns the maximum runtime of an execution task in seconds.

**Parameters**

**timeout_overwrite** (Duration) – If this is set, it will overwrite the global timeout.

**Returns**

**execute_timeout** – The execution timeout with the highest precedence (method overwrite then global configuration).

**Return type**

Duration

**get_train_timeout**(*timeout_overwrite: Optional[Duration] = None*) → Duration

Returns the maximum runtime of a training task in seconds.

**Parameters**

**timeout_overwrite** (Duration) – If this is set, it will overwrite the global timeout.

**Returns**

**train_timeout** – The training timeout with the highest precedence (method overwrite then global configuration).

**Return type**

Duration

**task_cpu_limit: Optional[float] = None**

**task_memory_limit: Optional[int] = None**

**tasks_per_host: int = 1**

**train_timeout: Duration = <Duration 8 hours>**

**use_preliminary_model_on_train_timeout: bool = True**

**use_preliminary_scores_on_execute_timeout: bool = True**

timeeval.resource_constraints.**GB = 1073741824**

$1GB = 2^{30}$Bytes

Can be used to set the memory limit.

#### Examples

```
>>> from timeeval.resource_constraints import ResourceConstraints, GB
>>> ResourceConstraints(task_memory_limit=1 * GB)
```

timeeval.resource_constraints.**MB = 1048576**

$1MB = 2^{20}$Bytes

Can be used to set the memory limit.

#### Examples

```
>>> from timeeval.resource_constraints import ResourceConstraints, MB
>>> ResourceConstraints(task_memory_limit=500 * MB)
```

### 3.1.8 timeeval.constants

**class** timeeval.constants.**HPI_CLUSTER**

Cluster constant for the HPI cluster.

These constants are applicable only for the HPI infrastructure and might not be useful for you.

**BENCHMARK = 'benchmark'**

**CORRELATION_ANOMALIES = 'correlation-anomalies'**

**MULTIVARIATE_ANOMALY_TEST_CASES = 'multivariate-anomaly-test-cases'**

**MULTIVARIATE_TEST_CASES = 'multivariate-test-cases'**

**UNIVARIATE_ANOMALY_TEST_CASES = 'univariate-anomaly-test-cases'**

**VARIABLE_LENGTH_TEST_CASES = 'variable-length'**

**akita_dataset_paths: Dict[str, Path] = {'benchmark':**
**PosixPath('/home/projects/akita/data/benchmark-data/data-processed'),**
**'correlation-anomalies':**
**PosixPath('/home/projects/akita/data/correlation-anomalies'),**
**'multivariate-anomaly-test-cases':**
**PosixPath('/home/projects/akita/data/multivariate-anomaly-test-cases'),**
**'multivariate-test-cases':**
**PosixPath('/home/projects/akita/data/multivariate-test-cases'),**
**'univariate-anomaly-test-cases':**
**PosixPath('/home/projects/akita/data/univariate-anomaly-test-cases'),**
**'variable-length': PosixPath('/home/projects/akita/data/variable-length')}**

This dictionary contains the paths to the dataset collection folders.

**nodes: List[str] = ['odin01', 'odin02', 'odin03', 'odin04', 'odin05', 'odin06', 'odin07', 'odin08', 'odin09', 'odin10', 'odin11', 'odin12', 'odin13', 'odin14']**

All nodes of the homogenous HPI cluster.

**nodes_ip: List[str] = ['172.20.11.101', '172.20.11.102', '172.20.11.103', '172.20.11.104', '172.20.11.105', '172.20.11.106', '172.20.11.107', '172.20.11.108', '172.20.11.109', '172.20.11.110', '172.20.11.111', '172.20.11.112', '172.20.11.113', '172.20.11.114']**

All IP addresses of the nodes in the homogenous HPI cluster.

**odin01: str = 'odin01'**

**odin01_ip: str = '172.20.11.101'**

**odin02: str = 'odin02'**

**odin02_ip: str = '172.20.11.102'**

**odin03: str = 'odin03'**

**odin03_ip: str = '172.20.11.103'**

**odin04: str = 'odin04'**

**odin04_ip: str = '172.20.11.104'**

**odin05: str = 'odin05'**

**odin05_ip: str = '172.20.11.105'**

**odin06: str = 'odin06'**

**odin06_ip: str = '172.20.11.106'**

**odin07: str = 'odin07'**

**odin07_ip: str = '172.20.11.107'**

**odin08: str = 'odin08'**

**odin08_ip: str = '172.20.11.108'**

**odin09: str = 'odin09'**

**odin09_ip: str = '172.20.11.109'**

**odin10: str = 'odin10'**

**odin10_ip: str = '172.20.11.110'**

**odin11: str = 'odin11'**

**odin11_ip: str = '172.20.11.111'**

**odin12: str = 'odin12'**

**odin12_ip: str = '172.20.11.112'**

**odin13:** str = 'odin13'

**odin13_ip:** str = '172.20.11.113'

**odin14:** str = 'odin14'

**odin14_ip:** str = '172.20.11.114'

### 3.1.9 timeeval.data_types.ExecutionType

**class** timeeval.data_types.**ExecutionType**(*value*)

Bases: Enum

Enum used to indicate the execution type of algorithms.

TimeEval calls each algorithm up to two times with two different execution types and passes the current execution type as an object of this class to the algorithm adapter implementation.

Depending on the algorithm's *timeeval.TrainingType*, it requires a training step. TimeEval will call these algorithms first with the execution type set to `TRAIN`. Then, for all algorithms, the algorithm is called with execution type `EXECUTE`.

**EXECUTE = 'execute'**

**TRAIN = 'train'**

## 3.2 timeeval.adapters package

### 3.2.1 timeeval.adapters.base module

**class** timeeval.adapters.base.**Adapter**

Bases: ABC

The base class for all adapters. An adapter is a wrapper around an anomaly detection algorithm that allows to execute it in a standardized way with the TimeEval framework. A subclass of Adapter must implement the _call method that executes the algorithm and returns the results. Optionally, it can also implement the get_prepare_fn and get_finalize_fn methods that are called before and after the execution of the algorithm, respectively.

**get_finalize_fn**() → Optional[Callable[[], None]]

This method is executed after all algorithms are run.

**get_prepare_fn**() → Optional[Callable[[], None]]

This method is executed before all algorithms are run.

### 3.2.2 timeeval.adapters.distributed module

**class** timeeval.adapters.distributed.**DistributedAdapter**(*algorithm: Callable[[Union[ndarray, Path], dict], Union[ndarray, Path]]*, *remote_command: str*, *remote_user: str*, *remote_hosts: List[str]*)

Bases: *Adapter*

An adapter that allows to run a function as an anomaly detector on multiple remote machines. So far, this adapter only supports TSFunctions as algorithms. Please, be aware that you need password-less ssh to the remote machines!

> **Warning:** This adapter is deprecated and will be removed in a future version of TimeEval.

**Parameters**

- **algorithm** (TSFunction) – The function to run.
- **remote_command** (str) – The command to run on the remote machines.
- **remote_user** (str) – The user to use for the ssh connection.
- **remote_hosts** (List[str]) – The hosts to connect to.

### 3.2.3 timeeval.adapters.docker module

class timeeval.adapters.docker.**AlgorithmInterface**(*dataInput: pathlib.PurePath*, *dataOutput: pathlib.PurePath*, *modelInput: pathlib.PurePath*, *modelOutput: pathlib.PurePath*, *executionType: timeeval.data_types.ExecutionType*, *customParameters: Dict[str, Any] = <factory>*)

Bases: object

**customParameters:** Dict[str, Any]

**dataInput:** PurePath

**dataOutput:** PurePath

**executionType:** *ExecutionType*

**modelInput:** PurePath

**modelOutput:** PurePath

**to_json_string**() → str

class timeeval.adapters.docker.**DockerAdapter**(*image_name: str*, *tag: str = 'latest'*, *group_privileges: str = 'akita'*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*, *memory_limit_overwrite: Optional[int] = None*, *cpu_limit_overwrite: Optional[float] = None*)

Bases: *Adapter*

An adapter that allows to run a Docker image as an anomaly detector. You can find a list of available Docker images on GitHub.

**Parameters**

- **image_name** (str) – The name of the Docker image to run.
- **tag** (str) – The tag of the Docker image to run. Defaults to "latest".

- **group_privileges** (str) – The group privileges to use for the Docker container. Defaults to "akita".
- **skip_pull** (bool) – Whether to skip pulling the Docker image. Defaults to False.
- **timeout** (Optional[Duration]) – The timeout for the Docker container. If not set, the timeout is taken from the ResourceConstraints.
- **memory_limit_overwrite** (Optional[int]) – The memory limit for the Docker container. If not set, the memory limit is taken from the ResourceConstraints.
- **cpu_limit_overwrite** (Optional[float]) – The CPU limit for the Docker container. If not set, the CPU limit is taken from the ResourceConstraints.

**get_finalize_fn**() → Optional[Callable[[], None]]

This method is executed after all algorithms are run.

**get_prepare_fn**() → Optional[Callable[[], None]]

This method is executed before all algorithms are run.

**exception** timeeval.adapters.docker.**DockerAdapterInternalError**

Bases: Exception

**exception** timeeval.adapters.docker.**DockerAlgorithmFailedError**

Bases: Exception

**class** timeeval.adapters.docker.**DockerJSONEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: NumpyEncoder

**default**(*o: Any*) → Any

Implement this method in a subclass such that it returns a serializable object for o, or calls the base implementation (to raise a TypeError).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

**exception** timeeval.adapters.docker.**DockerMemoryError**

Bases: Exception

**exception** timeeval.adapters.docker.**DockerTimeoutError**

Bases: Exception

### 3.2.4 timeeval.adapters.function module

class timeeval.adapters.function.**FunctionAdapter**(*fn: Callable[[Union[ndarray, Path], dict], Union[ndarray, Path]]*)

Bases: *Adapter*

An adapter that allows to run a function as an anomaly detector.

**Parameters**
**fn** (TSFunction) – The function to run.

**static identity**() → *FunctionAdapter*

### 3.2.5 timeeval.adapters.jar module

class timeeval.adapters.jar.**JarAdapter**(*jar_file: str*, *output_file: str*, *args: List[Any]*, *kwargs: Dict[str, Any]*, *verbose: bool = False*)

Bases: *Adapter*

An adapter that allows to run a jar file as an anomaly detector.

**Warning:** This adapter is deprecated and will be removed in a future version of TimeEval.

**Parameters**

- **jar_file** (str) – The path to the jar file to run.
- **output_file** (str) – The path to the file to which the jar file writes its output.
- **args** (List[Any]) – The arguments to pass to the jar file.
- **kwargs** (Dict[str, Any]) – The keyword arguments to pass to the jar file.
- **verbose** (bool) – Whether to print the output of the jar file to the console.

### 3.2.6 timeeval.adapters.multivar module

class timeeval.adapters.multivar.**AggregationMethod**(*value*)

Bases: Enum

An enum that specifies how to aggregate the anomaly scores of the channels.

**MAX = 2**

aggregates channel scores using the element-wise max.

**MEAN = 0**

aggregates channel scores using the element-wise mean.

**MEDIAN = 1**

aggregates channel scores using the element-wise median.

**SUM_BEFORE = 3**

sums the channels before running the anomaly detector.

class timeeval.adapters.multivar.**MultivarAdapter**(*adapter:* Adapter, *aggregation:* AggregationMethod = *AggregationMethod.MEAN*)

Bases: *Adapter*

An adapter that allows to apply univariate anomaly detectors to multiple dimensions of a timeseries. In one case, the adapter runs the anomaly detector on each dimension separately and aggregates the results using the specified aggregation method. In the other case, the adapter combines the dimensions into a single timeseries and runs the anomaly detector on the combined timeseries.

**Parameters**

- **adapter** (Adapter) – The Adapter that runs the anomaly detector on each dimension.
- **aggregation** (*AggregationMethod*) – The *AggregationMethod* to use to combine the anomaly scores of the dimensions.

**get_finalize_fn**() → Optional[Callable[[], None]]

This method is executed after all algorithms are run.

**get_prepare_fn**() → Optional[Callable[[], None]]

This method is executed before all algorithms are run.

## 3.3 timeeval.algorithms package

### 3.3.1 timeeval.algorithms.arima

timeeval.algorithms.**arima**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

ARIMA

Anomoly detector using ARIMA estimation and (default: euclidean) distance function to calculate prediction error as anomaly score

> **Warning:** The implementation of this algorithm is not publicly available (closed source). Thus, TimeEval will fail to download the Docker image and the algorithm will not be available. Please contact the authors of the algorithm for the implementation and build the algorithm Docker image yourself.

**Algorithm Parameters:**

**window_size: int**
Size of sliding window (also used as prediction window size) (default: 20)

**max_lag: int**
Number of points, after which the ARIMA model is re-fitted to the data to deal with trends and shifts (default: 30000)

**p_start: int**
Minimum AR-order for the auto-ARIMA process (default: 1)

**q_start: int**
Minimum MA-order for the auto-ARIMA process (default: 1)

**max_p: int**
Maximum AR-order for the auto-ARIMA process (default: 5)

**max_q: int**
Maximum MA-order for the auto-ARIMA process (default: `5`)

**differencing_degree: int**
Differencing degree for the auto-ARIMA process (default: `0`)

**distance_metric: enum[Euclidean,Mahalanobis,Garch,SSA,Fourier,DTW,EDRS,TWED]**
Distance measure used to calculate the prediction error = anomaly score (default: `Euclidean`)

**random_state: int**
Seed for the random number generator (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the ARIMA algorithm.

**Return type**
*Algorithm*

### 3.3.2 timeeval.algorithms.autoencoder

timeeval.algorithms.**autoencoder**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

AutoEncoder (AE)

Implementation of https://dl.acm.org/doi/10.1145/2689746.2689747

**Algorithm Parameters:**

**latent_size: int**
Dimensionality of the latent space (default: `32`)

**epochs: int**
Number of training epochs (default: `10`)

**learning_rate: float**
Learning rate (default: `0.005`)

**split: float**
Fraction to split training data by for validation (default: `0.8`)

**early_stopping_delta: float**
If loss is *delta* or less smaller for *patience* epochs, stop (default: `0.5`)

**early_stopping_patience: int**
If loss is *delta* or less smaller for *patience* epochs, stop (default: `10`)

**random_state: int**
Seed for the random number generator (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm

- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the AutoEncoder (AE) algorithm.

**Return type**
*Algorithm*

### 3.3.3 timeeval.algorithms.bagel

timeeval.algorithms.**bagel**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Bagel

Implementation of https://doi.org/10.1109/PCCC.2018.8710885

**Algorithm Parameters:**

**window_size: int**
Size of sliding windows (default: 120)

**latent_size: int**
Dimensionality of encoding (default: 8)

**hidden_layer_shape: List[int]**
NN hidden layers structure (default: [100, 100])

**dropout: float**
Rate of conditional dropout used (default: 0.1)

**cuda: boolean**
Use GPU for training (default: False)

**epochs: int**
Number of passes over the entire dataset (default: 50)

**batch_size: int**
Batch size for input data (default: 128)

**split: float**
Fraction to split training data by for validation (default: 0.8)

**early_stopping_delta: float**
If loss is *delta* or less smaller for *patience* epochs, stop (default: 0.5)

**early_stopping_patience: int**
If loss is *delta* or less smaller for *patience* epochs, stop (default: 10)

**random_state: int**
Seed for the random number generator (default: 42)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.

- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Bagel algorithm.

**Return type**
*Algorithm*

### 3.3.4 timeeval.algorithms.baseline_increasing

timeeval.algorithms.**baseline_increasing**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Increasing Baseline

Baseline that returns a score that steadily increases from 0 to 1

**Algorithm Parameters:**

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Increasing Baseline algorithm.

**Return type**
*Algorithm*

### 3.3.5 timeeval.algorithms.baseline_normal

timeeval.algorithms.**baseline_normal**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Normal Baseline

Baseline that returns a score of all zeros

**Algorithm Parameters:**

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Normal Baseline algorithm.

**Return type**
*Algorithm*

### 3.3.6 timeeval.algorithms.baseline_random

timeeval.algorithms.**baseline_random**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Random Baseline

Baseline that returns a random score between 0 and 1

**Algorithm Parameters:**

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Random Baseline algorithm.

**Return type**
*Algorithm*

### 3.3.7 timeeval.algorithms.cblof

timeeval.algorithms.**cblof**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

CBLOF

Implementation of https://doi.org/10.1016/S0167-8655(03)00003-5.

**Algorithm Parameters:**

**n_clusters: int**
The number of clusters to form as well as the number of centroids to generate. (default: `8`)

**alpha: float**
Coefficient for deciding small and large clusters. The ratio of the number of samples in large clusters to the number of samples in small clusters. (0.5 < alpha < 1) (default: `0.9`)

**beta: float**
Coefficient for deciding small and large clusters. For a list sorted clusters by size |*C1*|, |*C2*|, . . . , |*Cn*|, *beta* = |*Ck*|/|*Ck-1*|. (1.0 < beta ) (default: `5`)

**use_weights: boolean**
If set to True, the size of clusters are used as weights in outlier score calculation. (default: `false`)

**random_state: int**
Seed for random number generation. (default: `42`)

**n_jobs: int**
The number of parallel jobs to run for neighbors search. If *-1*, then the number of jobs is set to the number of CPU cores. Affects only kneighbors and kneighbors_graph methods. (default: `1`)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the CBLOF algorithm.

**Return type**
*Algorithm*

### 3.3.8 timeeval.algorithms.cof

timeeval.algorithms.**cof**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

COF

Implementation of https://doi.org/10.1007/3-540-47887-6_53.

**Algorithm Parameters:**

**n_neighbors: int**
Number of neighbors to use by default for k neighbors queries. Note that n_neighbors should be less than the number of samples. If n_neighbors is larger than the number of samples provided, all samples will be used. (default: 20)

**random_state: int**
Seed for random number generation. (default: 42)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the COF algorithm.

**Return type**
*Algorithm*

### 3.3.9 timeeval.algorithms.copod

timeeval.algorithms.**copod**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

COPOD

Implementation of https://publications.pik-potsdam.de/pubman/faces/ViewItemOverviewPage.jsp?itemId=item_24536.

**Algorithm Parameters:**

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the COPOD algorithm.

**Return type**
*Algorithm*

### 3.3.10 timeeval.algorithms.dae

timeeval.algorithms.**dae**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

DenoisingAutoEncoder (DAE)

Implementation of https://dl.acm.org/doi/10.1145/2689746.2689747

**Algorithm Parameters:**

**latent_size: int**
Dimensionality of latent space (default: `32`)

**epochs: int**
Number of training epochs (default: `10`)

**learning_rate: float**
Learning rate (default: `0.005`)

**noise_ratio: float**
Percentage of points that are converted to noise (0) during training (default: `0.1`)

**split: float**
Fraction to split training data by for validation (default: `0.8`)

**early_stopping_delta: float**
If loss is *delta* or less smaller for *patience* epochs, stop (default: `0.5`)

**early_stopping_patience: int**
If loss is *delta* or less smaller for *patience* epochs, stop (default: `10`)

**random_state: int**
Seed for the random number generator (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the DenoisingAutoEncoder (DAE) algorithm.

**Return type**
*Algorithm*

### 3.3.11 timeeval.algorithms.damp

timeeval.algorithms.**damp**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

DAMP

Implementation of https://www.cs.ucr.edu/~eamonn/DAMP_long_version.pdf

**Algorithm Parameters:**

**anomaly_window_size: int**
Size of the sliding windows (default: `50`)

**n_init_train: int**
Fraction of data used to warmup streaming. (default: `100`)

**max_lag: int**
Maximum size to look back in time. (default: `None`)

**lookahead: int**
Amount of steps to look into the future for deciding which future windows to skip analyzing. (default: `None`)

**random_state: int**
Seed for the random number generator (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the DAMP algorithm.

**Return type**
*Algorithm*

### 3.3.12 timeeval.algorithms.dbstream

timeeval.algorithms.**dbstream**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

DBStream

A simple density-based clustering algorithm that assigns data points to micro-clusters with a given radius and implements shared-density-based reclustering.

**Algorithm Parameters:**

**window_size: int**
The length of the subsequences the dataset should be splitted in. (default: `20`)

**radius: float**
The radius of micro-clusters. (default: `0.1`)

**lambda: float**
The lambda used in the fading function. (default: `0.001`)

**distance_metric: enum[Euclidean,Manhattan,Maximum]**
The metric used to calculate distances. If shared_density is TRUE this has to be Euclidian. (default: `Euclidean`)

**shared_density: boolean**
Record shared density information. If set to TRUE then shared density is used for reclustering, otherwise reachability is used (overlapping clusters with less than r(1alpha) distance are clustered together) (default: `True`)

**n_clusters: int**
The number of macro clusters to be returned if macro is true. (default: `0`)

**alpha: float**
For shared density: The minimum proportion of shared points between to clus-ters to warrant combining them (a suitable value for 2D data is .3). For reacha-bility clustering it is a distance factor (default: `0.1`)

**min_weight: float**
The proportion of the total weight a macro-cluster needs to have not to be noise(between 0 and 1). (default: `0.0`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the DBStream algorithm.

**Return type**
*Algorithm*

### 3.3.13 timeeval.algorithms.deepant

timeeval.algorithms.**deepant**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

DeepAnT

Adapted community implementation (https://github.com/dev-aadarsh/DeepAnT)

**Algorithm Parameters:**

**epochs: int**
Number of training epochs (default: `50`)

**window_size: int**
History window: Number of time stamps in history, which are taken into account (default: `45`)

**prediction_window_size: int**
Prediction window: Number of data points that will be predicted from each window (default: `1`)

**learning_rate: float**
Learning rate (default: `1e-05`)

**batch_size: int**
Batch size for input data (default: `45`)

**random_state: int**
Seed for the random number generator (default: `42`)

**split: float**
Train-validation split for early stopping (default: `0.8`)

**early_stopping_delta: float**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `0.05`)

**early_stopping_patience: int**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `10`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the DeepAnT algorithm.

**Return type**
*Algorithm*

### 3.3.14 timeeval.algorithms.deepnap

timeeval.algorithms.**deepnap**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

DeepNAP

Implementation of https://doi.org/10.1016/j.ins.2018.05.020

**Algorithm Parameters:**

**anomaly_window_size: int**
Size of the sliding windows (default: `15`)

**partial_sequence_length: int**
Number of points taken from the beginning of the predicted window used to build a partial sequence (with neighboring points) that is passed through another linear network. (default: `3`)

**lstm_layers: int**
Number of LSTM layers within encoder and decoder (default: `2`)

**rnn_hidden_size: int**
Number of neurons in LSTM hidden layer (default: `200`)

**dropout: float**
Probability for a neuron to be zeroed for regularization (default: `0.5`)

**linear_hidden_size: int**
Number of neurons in linear hidden layer (default: `100`)

**batch_size: int**
Number of instances trained at the same time (default: `32`)

**validation_batch_size: int**
Number of instances used for validation at the same time (default: `256`)

**epochs: int**
Number of training iterations over entire dataset; recommended value: 256 (default: `1`)

**learning_rate: float**
Learning rate for Adam optimizer (default: `0.001`)

**split: float**
Train-validation split for early stopping (default: `0.8`)

**early_stopping_delta: float**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `0.05`)

**early_stopping_patience: int**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `10`)

**random_state: int**
Seed for the random number generator (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the DeepNAP algorithm.

**Return type**
*Algorithm*

### 3.3.15 timeeval.algorithms.donut

timeeval.algorithms.**donut**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Donut

Implementation of https://doi.org/10.1145/3178876.3185996

**Algorithm Parameters:**

**window_size: int**
Size of sliding windows (default: `120`)

**latent_size: int**
Dimensionality of encoding (default: `5`)

**regularization: float**
Factor for regularization in loss (default: `0.001`)

**linear_hidden_size: int**
Size of linear hidden layer (default: `100`)

**epochs: int**
Number of training passes over entire dataset (default: `256`)

**random_state: int**
Seed for random number generation. (default: `42`)

**use_column_index: int**
The column index to use as input for the univariate algorithm for multivariate datasets. The selected single channel of the multivariate time series is analyzed by the algorithms. The index is 0-based and does not include the index-column ('timestamp'). The single channel of an univariate dataset, therefore, has index 0. (default: `0`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Donut algorithm.

**Return type**
*Algorithm*

### 3.3.16 timeeval.algorithms.dspot

timeeval.algorithms.**dspot**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

DSPOT

Implementation of https://doi.org/10.1145/3097983.3098144.

**Algorithm Parameters:**

**q: float**
Main parameter: maximum probability of an abnormal event (default: `0.001`)

**n_init: int**
Calibration: number of data used to calibrate algorithm. The user must ensure that n_init * (1 - level) > 10 (default: `1000`)

**level: float**
Calibration: proportion of initial data (n_init) not involved in the tail distribution fit during initialization. The user must ensure that n_init * (1 - level) > 10 (default: `0.99`)

**up: boolean**
Compute upper thresholds (default: `true`)

**down: boolean**
Compute lower thresholds (default: `true`)

**alert: boolean**
Enable alert triggering, if false, even out-of-bounds-data will be taken into account for tail fit (default: `true`)

**bounded: boolean**
Performance: enable memory bounding (also improves performance) (default: `true`)

**max_excess: int**
Performance: maximum number of data stored to perform the tail fit when memory bounding is enabled (default: `200`)

**random_state: int**
Seed for the random number generator (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the DSPOT algorithm.

**Return type**
*Algorithm*

### 3.3.17 timeeval.algorithms.dwt_mlead

timeeval.algorithms.**dwt_mlead**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

DWT-MLEAD

Implementation of http://blogs.gm.fh-koeln.de/ciop/files/2019/01/thillwavelet.pdf.

**Algorithm Parameters:**

**start_level: int**
First discrete wavelet decomposition level to consider (default: `3`)

**quantile_epsilon: float**
Percentage of windows to flag as anomalous within each decomposition level's coefficients (default: `0.01`)

**random_state: int**
Seed for the random number generator (default: `42`)

**use_column_index: int**
The column index to use as input for the univariate algorithm for multivariate datasets. The selected single channel of the multivariate time series is analyzed by the algorithms. The index is 0-based and does not include the index-column ('timestamp'). The single channel of an univariate dataset, therefore, has index 0. (default: `0`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the DWT-MLEAD algorithm.

**Return type**
*Algorithm*

### 3.3.18 timeeval.algorithms.eif

timeeval.algorithms.**eif**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Extended Isolation Forest (EIF)

Extenstion to the basic isolation forest. Implementation of https://doi.org/10.1109/TKDE.2019.2947676. Code from https://github.com/sahandha/eif

**Algorithm Parameters:**

**n_trees: int**
The number of decision trees (base estimators) in the forest (ensemble). (default: `200`)

**max_samples: float**
The number of samples to draw from X to train each base estimator: *max_samples * X.shape[0]*. If unspecified (*null*), then *max_samples=min(256, X.shape[0])*. (default: `None`)

**extension_level: int**
Extension level 0 resembles standard isolation forest. If unspecified (*null*), then *extension_level=X.shape[1] - 1*. (default: `None`)

**limit: int**
The maximum allowed tree depth. This is by default set to average length of unsucessful search in a binary tree. (default: `None`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Extended Isolation Forest (EIF) algorithm.

**Return type**
*Algorithm*

### 3.3.19 timeeval.algorithms.encdec_ad

`timeeval.algorithms.`**encdec_ad**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

EncDec-AD

Implementation of https://arxiv.org/pdf/1607.00148.pdf

**Algorithm Parameters:**

**lstm_layers: int**
Number of LSTM layers within encoder and decoder (default: `1`)

**anomaly_window_size: int**
Size of the sliding windows (default: `30`)

**latent_size: int**
Size of the autoencoder's latent space (embedding size) (default: `40`)

**batch_size: int**
Number of instances trained at the same time (default: `32`)

**validation_batch_size: int**
Number of instances used for validation at the same time (default: `128`)

**epochs: int**
Number of training iterations over entire dataset (default: `50`)

**split: float**
Train-validation split for early stopping (default: `0.9`)

**early_stopping_delta: float**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `0.05`)

**early_stopping_patience: int**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `10`)

**learning_rate: float**
Learning rate for Adam optimizer (default: `0.001`)

**random_state: int**
Seed for the random number generator (default: `42`)

**window_size: int**
Size of the sliding windows (default: `30`)

**test_batch_size: int**
Number of instances used for testing at the same time (default: `128`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the EncDec-AD algorithm.

**Return type**
*Algorithm*

### 3.3.20 timeeval.algorithms.ensemble_gi

timeeval.algorithms.**ensemble_gi**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Ensemble GI

Implementation of https://doi.org/10.5441/002/edbt.2020.09

**Algorithm Parameters:**

**anomaly_window_size: int**
The size of the sliding window, in which *w* regions are made discrete. (default: `50`)

**n_estimators: int**
The number of models in the ensemble. (default: `10`)

**max_paa_transform_size: int**
Maximum size of the embedding space used by PAA (SAX word size *w*) (default: `20`)

**max_alphabet_size: int**
Maximum number of symbols used for discretization by SAX (*lpha*) (default: `10`)

**selectivity: float**
The fraction of models in the ensemble included in the end result. (default: `0.8`)

**random_state: int**
Seed for the random number generator (default: `42`)

**n_jobs: int**
The number of parallel jobs to use for executing the models. If *-1*, then the number of jobs is set to the number of CPU cores. (default: `1`)

**window_method: enum[sliding,tumbling,orig]**
Windowing method used to create subsequences. The original implementation had a strange method (*orig*) that is similar to *tumbling*, the paper uses a *sliding* window. However, *sliding* is significantly slower than *tumbling* while producing better results (higher anomaly score resolution). *orig* should not be used! (default: `sliding`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Ensemble GI algorithm.

**Return type**
*Algorithm*

### 3.3.21 timeeval.algorithms.fast_mcd

timeeval.algorithms.**fast_mcd**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Fast-MCD

Implementation of https://doi.org/10.2307/1270566

**Algorithm Parameters:**

**store_precision: boolean**
Specify if the estimated precision is stored (default: `True`)

**support_fraction: float**
The proportion of points to be included in the support of the raw MCD estimate. Default is None, which implies that the minimum value of support_fraction will be used within the algorithm: *(n_sample + n_features + 1) / 2*. The parameter must be in the range (0, 1). (default: `None`)

**random_state: int**
Determines the pseudo random number generator for shuffling the data. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Fast-MCD algorithm.

**Return type**
*Algorithm*

### 3.3.22 timeeval.algorithms.fft

timeeval.algorithms.**fft**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

FFT

Implementation of https://dl.acm.org/doi/10.5555/1789574.1789615 proudly provided by members of the HPI AKITA project.

**Algorithm Parameters:**

**fft_parameters: int**
Number of parameters to be used in IFFT for creating the fit. (default: `5`)

**context_window_size: int**
Centered window of neighbors to consider for the calculation of local outliers' z_scores (default: `21`)

**local_outlier_threshold: float**
Outlier threshold in multiples of sigma for local outliers (default: `0.6`)

**max_anomaly_window_size: int**
Maximum size of outlier regions. (default: `50`)

**max_sign_change_distance: int**
Maximum gap between two closed oppositely signed local outliers to detect a sign change for outlier region grouping. (default: `10`)

**random_state: int**
Seed for the random number generator (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the FFT algorithm.

**Return type**
*Algorithm*

### 3.3.23 timeeval.algorithms.generic_rf

timeeval.algorithms.**generic_rf**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Random Forest Regressor (RR)

A generic windowed forecasting method using random forest regression (requested by RollsRoyce). The forecasting error is used as anomaly score.

**Algorithm Parameters:**

**train_window_size: int**
Size of the training windows. Always predicts a single point! (default: `50`)

**n_trees: int**
The number of trees in the forest. (default: `100`)

**max_features_method: enum[auto,sqrt,log2]**
The number of features to consider when looking for the best split between trees: 'auto': max_features=n_features, 'sqrt': max_features=sqrt(n_features), 'log2': max_features=log2(n_features). (default: `auto`)

**bootstrap: boolean**
Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree. (default: `True`)

**max_samples: float**
If bootstrap is True, the number of samples to draw from X to train each base estimator. (default: `None`)

**random_state: int**
Seeds the randomness of the bootstrapping and the sampling of the features. (default: `42`)

**verbose: int**
Controls logging verbosity. (default: `0`)

**n_jobs: int**
The number of jobs to run in parallel. *-1* means using all processors (default: `1`)

**max_depth: int**
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. (default: `None`)

**min_samples_split: int**
The minimum number of samples required to split an internal node. (default: `2`)

**min_samples_leaf: int**
The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least *min_samples_leaf* training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression. (default: `1`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Random Forest Regressor (RR) algorithm.

**Return type**
*Algorithm*

### 3.3.24 timeeval.algorithms.generic_xgb

timeeval.algorithms.**generic_xgb**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

XGBoosting (RR)

A generic windowed forecasting method using XGBoost regression (requested by RollsRoyce). The forecasting error is used as anomaly score.

**Algorithm Parameters:**

**train_window_size: int**
Size of the training windows. Always predicts a single point! (default: `50`)

**n_estimators: int**
Number of gradient boosted trees. Equivalent to number of boosting rounds. (default: `100`)

**learning_rate: float**
Boosting learning rate (xgb's *eta*) (default: `0.1`)

**booster: enum[gbtree,gblinear,dart]**
Booster to use (default: `gbtree`)

**tree_method: enum[auto,exact,approx,hist]**
Tree method to use. Default to auto. If this parameter is set to default, XGBoost will choose the most conservative option available. *exact* is slowest, *hist* is fastest. Prefer *hist* and *approx* over *exact*, because for most datasets they have comparative quality, but are significantly faster. (default: `auto`)

**n_trees: int**
If >1, then boosting random forests with *n_trees* trees. (default: `1`)

**max_depth: int**
Maximum tree depth for base learners. (default: `None`)

**max_samples: float**
Subsample ratio of the training instance. (default: `None`)

**colsample_bytree: float**
Subsample ratio of columns when constructing each tree. (default: `None`)

**colsample_bylevel: float**
Subsample ratio of columns for each level. (default: `None`)

**colsample_bynode: float**
Subsample ratio of columns for each split. (default: `None`)

**random_state: int**
Seeds the randomness of the bootstrapping and the sampling of the features. (default: `42`)

**verbose: int**
Controls logging verbosity. (default: `0`)

**n_jobs: int**
The number of jobs to run in parallel. *-1* means using all processors. (default: `1`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.

- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the XGBoosting (RR) algorithm.

**Return type**
*Algorithm*

### 3.3.25 timeeval.algorithms.grammarviz3

timeeval.algorithms.**grammarviz3**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

GrammarViz

Implementation of https://doi.org/10.1145/3051126.

**Algorithm Parameters:**

**anomaly_window_size: int**
Size of the sliding window. Equal to the discord length! (default: 170)

**paa_transform_size: int**
Size of the embedding space used by PAA (paper calls it number of frames or SAX word size *w*) (performance parameter) (default: 4)

**alphabet_size: int**
Number of symbols used for discretization by SAX (paper uses *lpha*) (performance parameter) (default: 4)

**normalization_threshold: float**
Threshold for Z-normalization of subsequences (windows). If variance of a window is higher than this threshold, it is normalized. (default: 0.01)

**random_state: int**
Seed for the random number generator (default: 42)

**use_column_index: int**
The column index to use as input for the univariate algorithm for multivariate datasets. The selected single channel of the multivariate time series is analyzed by the algorithms. The index is 0-based and does not include the index-column ('timestamp'). The single channel of an univariate dataset, therefore, has index 0. (default: 0)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the GrammarViz algorithm.

**Return type**
*Algorithm*

### 3.3.26 timeeval.algorithms.grammarviz3_multi

timeeval.algorithms.**grammarviz3_multi**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

GrammarViz-Multivariate

Multivariate extension of the GrammarViz3 algorithm.

**Algorithm Parameters:**

**anomaly_window_size: int**
Size of the sliding window. Equal to the discord length! (default: `100`)

**output_mode: int**
Algorithm to use for output [Density, Discord, Full] (default: `2`)

**multi_strategy: int**
Strategy to handle multivariate output [Merge all, Merge clustered, All separate] (default: `1`)

**paa_transform_size: int**
Size of the embedding space used by PAA (paper calls it number of frames or SAX word size *w*) (performance parameter) (default: `5`)

**alphabet_size: int**
Number of symbols used for discretization by SAX (paper uses *lpha*) (performance parameter) (default: `6`)

**normalization_threshold: float**
Threshold for Z-normalization of subsequences (windows). If variance of a window is higher than this threshold, it is normalized. (default: `0.01`)

**random_state: int**
Seed for the random number generator (default: `42`)

**n_discords: int**
Number of discords to report when using discord output strategy (default: `10`)

**numerosity_reduction: boolean**
Disables / enables numerosity reduction strategy (default: `True`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the GrammarViz-Multivariate algorithm.

**Return type**
*Algorithm*

### 3.3.27 timeeval.algorithms.hbos

timeeval.algorithms.**hbos**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

HBOS

Implementation of https://citeseerx.ist.psu.edu/viewdoc/citations;jsessionid=2B4E3FB2BB07448253B4D45C3DAC2E95?doi=10.1.1.401.5686.

**Algorithm Parameters:**

**n_bins: int**
The number of bins. (default: `10`)

**alpha: float**
Regulizing alpha to prevent overflows. (default: `0.1`)

**bin_tol: float**
Parameter to decide the flexibility while dealing with the samples falling outside the bins. (default: `0.5`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the HBOS algorithm.

**Return type**
*Algorithm*

### 3.3.28 timeeval.algorithms.health_esn

timeeval.algorithms.**health_esn**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

HealthESN

Implementation of https://doi.org/10.1007/s00521-018-3747-z

**Algorithm Parameters:**

**linear_hidden_size: int**
Hidden units in ESN reservoir. (default: `500`)

**prediction_window_size: int**
Window of predicted points in the future. (default: `20`)

**connectivity: float**
How dense the units in the reservoir are connected (= percentage of non-zero weights) (default: `0.25`)

**spectral_radius: float**
Factor used for random initialization of ESN neural connections. (default: `0.6`)

**activation: enum[tanh,sigmoid]**
Activation function used for the ESN. (default: `tanh`)

**random_state: int**
Seed for the random number generator (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the HealthESN algorithm.

**Return type**
*Algorithm*

### 3.3.29 timeeval.algorithms.hif

`timeeval.algorithms.hif`(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Hybrid Isolation Forest (HIF)

Implementation of https://arxiv.org/abs/1705.03800

**Algorithm Parameters:**

**n_trees: int**
The number of decision trees (base estimators) in the forest (ensemble). (default: `1024`)

**max_samples: float**
The number of samples to draw from X to train each base estimator: *max_samples * X.shape[0]*. If unspecified (*null*), then *max_samples=min(256, X.shape[0])*. (default: `None`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Hybrid Isolation Forest (HIF) algorithm.

**Return type**
*Algorithm*

### 3.3.30 timeeval.algorithms.hotsax

timeeval.algorithms.**hotsax**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

HOT SAX

Implementation of https://doi.org/10.1109/ICDM.2005.79.

**Algorithm Parameters:**

**num_discords: int**
The number of anomalies (discords) to search for in the time series. If not set, the scores for all discords are searched. (default: `None`)

**anomaly_window_size: int**
Size of the sliding window. Equal to the discord length! (default: `100`)

**paa_transform_size: int**
Size of the embedding space used by PAA (paper calls it number of frames or SAX word size *w*) (performance parameter) (default: `3`)

**alphabet_size: int**
Number of symbols used for discretization by SAX (paper uses *lpha*) (performance parameter) (default: `3`)

**normalization_threshold: float**
Threshold for Z-normalization of subsequences (windows). If variance of a window is higher than this threshold, it is normalized. (default: `0.01`)

**random_state: int**
Seed for the random number generator (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the HOT SAX algorithm.

**Return type**
*Algorithm*

### 3.3.31 timeeval.algorithms.hybrid_knn

timeeval.algorithms.**hybrid_knn**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Hybrid KNN

Implementation of https://www.hindawi.com/journals/cin/2017/8501683/

**Algorithm Parameters:**

**linear_layer_shape: List[int]**
NN structure with embedding dim as last value (default: `[100, 10]`)

**split: float**
train-validation split (default: `0.8`)

**anomaly_window_size: int**
windowing size for time series (default: `20`)

**batch_size: int**
number of simultaneously trained data instances (default: `64`)

**test_batch_size: int**
number of simultaneously tested data instances (default: `256`)

**epochs: int**
number of training iterations over entire dataset (default: `1`)

**early_stopping_delta: float**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `0.05`)

**early_stopping_patience: int**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `10`)

**learning_rate: float**
Gradient factor for backpropagation (default: `0.001`)

**n_neighbors: int**
Defines which neighbour's distance to use (default: `12`)

**n_estimators: int**
Defines number of ensembles (default: `3`)

**random_state: int**
Seed for the random number generator (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Hybrid KNN algorithm.

**Return type**
*Algorithm*

### 3.3.32 timeeval.algorithms.if_lof

`timeeval.algorithms.if_lof(`*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*`)` → *Algorithm*

IF-LOF

Isolation Forest - Local Outlier Factor: Uses a 3 step process - Building an isolation forest, pruning the forest with a computed treshhold, and applies local outlier factor to the resulting dataset

**Algorithm Parameters:**

**n_trees: int**
Number of trees in isolation forest (default: `200`)

**max_samples: float**
The number of samples to draw from X to train each tree: *max_samples * X.shape[0]*. If unspecified (*null*), then *max_samples=min(256, X.shape[0])*. (default: `None`)

**n_neighbors: int**
Number neighbors to look at in local outlier factor calculation (default: `10`)

**alpha: float**
Scalar that depends on consideration of the dataset and controls the amount of data to be pruned (default: `0.5`)

**m: int**
m features with highest scores will be used for pruning (default: `None`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the IF-LOF algorithm.

**Return type**
*Algorithm*

### 3.3.33 timeeval.algorithms.iforest

timeeval.algorithms.**iforest**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Isolation Forest (iForest)

Implementation of https://doi.org/10.1145/2133360.2133363.

**Algorithm Parameters:**

**n_trees: int**
The number of decision trees (base estimators) in the forest (ensemble). (default: `100`)

**max_samples: float**
The number of samples to draw from X to train each base estimator: *max_samples * X.shape[0]*. If unspecified (*null*), then *max_samples=min(256, n_samples)*. (default: `None`)

**max_features: float**
The number of features to draw from X to train each base estimator: *max_features * X.shape[1]*. (default: `1.0`)

**bootstrap: boolean**
If True, individual trees are fit on random subsets of the training data sampled with replacement. If False, sampling without replacement is performed. (default: `false`)

**random_state: int**
Seed for random number generation. (default: `42`)

**verbose: int**
Controls the verbosity of the tree building process logs. (default: `0`)

**n_jobs: int**
The number of jobs to run in parallel. If -1, then the number of jobs is set to the number of cores. (default: 1)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Isolation Forest (iForest) algorithm.

**Return type**
*Algorithm*

### 3.3.34 timeeval.algorithms.img_embedding_cae

timeeval.algorithms.**img_embedding_cae**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

ImageEmbeddingCAE

Implementation of http://arxiv.org/abs/2009.02040

**Algorithm Parameters:**

**anomaly_window_size: int**
length of one time series chunk (tumbling window) (default: `512`)

**kernel_size: int**
width, height of each convolution kernel (stride is equal to this value) (default: `2`)

**num_kernels: int**
number of convolution kernels used in each layer (default: `64`)

**latent_size: int**
number of neurons used in the embedding layer (default: `100`)

**leaky_relu_alpha: float**
alpha value used for leaky relu activation function (default: `0.03`)

**batch_size: int**
number of simultaneously trained data instances (default: `32`)

**test_batch_size: int**
number of simultaneously trained data instances (default: `128`)

**learning_rate: float**
Gradient factor for backpropagation (default: `0.001`)

**epochs: int**
number of training iterations over entire dataset (default: `30`)

**split: float**
train-validation split (default: `0.8`)

**early_stopping_delta: float**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `0.05`)

**early_stopping_patience: int**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `10`)

**random_state: int**
Seed for the random number generator (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the ImageEmbeddingCAE algorithm.

**Return type**
*Algorithm*

### 3.3.35 timeeval.algorithms.kmeans

timeeval.algorithms.**kmeans**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

k-Means

Implementation of http://robotics.estec.esa.int/i-SAIRAS/isairas2001/papers/Paper_AS012.pdf

**Algorithm Parameters:**

**n_clusters: int**
The number of clusters to form as well as the number of centroids to generate. The bigger *n_clusters* (*k*) is, the less noisy the anomaly scores are. (default: `20`)

**anomaly_window_size: int**
Size of sliding windows. The bigger *window_size* is, the bigger the anomaly context is. If it's to big, things seem anomalous that are not. If it's too small, the algorithm is not able to find anomalous windows and looses its time context. (default: `20`)

**stride: int**
Stride of sliding windows. It is the step size between windows. The larger *stride* is, the noisier the scores get. If *stride* == *window_size*, they are tumbling windows. (default: `1`)

**n_jobs: int**
Internal parallelism used (sample-wise in the main loop which assigns each sample to its closest center). If *-1* or *None*, all available CPUs are used. (default: `1`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.

- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the k-Means algorithm.

**Return type**
*Algorithm*

### 3.3.36 timeeval.algorithms.knn

timeeval.algorithms.**knn**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

KNN

Implementation of https://doi.org/10.1145/342009.335437.

**Algorithm Parameters:**

**n_neighbors: int**
Number of neighbors to use by default for *kneighbors* queries. (default: 5)

**leaf_size: int**
Leaf size passed to *BallTree*. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem. (default: 30)

**method: enum[largest,mean,median]**
'largest': use the distance to the kth neighbor as the outlier score, 'mean': use the average of all k neighbors as the outlier score, 'median': use the median of the distance to k neighbors as the outlier score. (default: largest)

**radius: float**
Range of parameter space to use by default for *radius_neighbors* queries. (default: 1.0)

**distance_metric_order: int**
Parameter for the Minkowski metric from sklearn.metrics.pairwise.pairwise_distances. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used. See http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.pairwise_distances. (default: 2)

**n_jobs: int**
The number of parallel jobs to run for neighbors search. If -1, then the number of jobs is set to the number of CPU cores. Affects only kneighbors and kneighbors_graph methods. (default: 1)

**random_state: int**
Seed for random number generation. (default: 42)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the KNN algorithm.

**Return type**
*Algorithm*

### 3.3.37 timeeval.algorithms.laser_dbn

timeeval.algorithms.**laser_dbn**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

LaserDBN

Implementation of https://doi.org/10.1007/978-3-662-53806-7_3

**Algorithm Parameters:**

**timesteps: int**
Number of time steps the DBN builds probabilities for (min: 2) (default: `2`)

**n_bins: int**
Number of bins used for discretization. (default: `10`)

**random_state: int**
Seed for the random number generator (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the LaserDBN algorithm.

**Return type**
*Algorithm*

### 3.3.38 timeeval.algorithms.left_stampi

timeeval.algorithms.**left_stampi**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Left STAMPi

Implementation of https://www.cs.ucr.edu/~eamonn/PID4481997_extend_Matrix%20Profile_I.pdf

**Algorithm Parameters:**

**anomaly_window_size: int**
Size of the sliding windows (default: `50`)

**n_init_train: int**
Fraction of data used to warmup streaming. (default: `100`)

**random_state: int**
Seed for the random number generator (default: `42`)

**use_column_index: int**
The column index to use as input for the univariate algorithm for multivariate datasets. The selected single channel of the multivariate time series is analyzed by the algorithms. The index is 0-based and does not include the index-column ('timestamp'). The single channel of an univariate dataset, therefore, has index 0. (default: `0`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Left STAMPi algorithm.

**Return type**
*Algorithm*

### 3.3.39 timeeval.algorithms.lof

`timeeval.algorithms.lof`(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

LOF

Implementation of https://doi.org/10.1145/342009.335388.

**Algorithm Parameters:**

**n_neighbors: int**
Number of neighbors to use by default for *kneighbors* queries. If n_neighbors is larger than the number of samples provided, all samples will be used. (default: `20`)

**leaf_size: int**
Leaf size passed to *BallTree* or *KDTree*. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem. (default: `30`)

**distance_metric_order: int**
Parameter for the Minkowski metric from sklearn.metrics.pairwise.pairwise_distances. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used. See http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.pairwise_distances. (default: `2`)

**n_jobs: int**
The number of parallel jobs to run for neighbors search. If `-1`, then the number of jobs is set to the number of CPU cores. Affects only kneighbors and kneighbors_graph methods. (default: `1`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.

- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the LOF algorithm.

**Return type**
*Algorithm*

### 3.3.40 timeeval.algorithms.lstm_ad

timeeval.algorithms.**lstm_ad**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

LSTM-AD

Implementation of https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2015-56.pdf

**Algorithm Parameters:**

**lstm_layers: int**
Number of stacked LSTM layers (default: 2)

**split: float**
Train-validation split for early stopping (default: 0.9)

**window_size: int**
(default: 30)

**prediction_window_size: int**
Number of points predicted (default: 1)

**batch_size: int**
Number of instances trained at the same time (default: 32)

**validation_batch_size: int**
Number of instances used for validation at the same time (default: 128)

**epochs: int**
Number of training iterations over entire dataset (default: 50)

**early_stopping_delta: float**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: 0.05)

**early_stopping_patience: int**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: 10)

**learning_rate: float**
Learning rate for Adam optimizer (default: 0.001)

**random_state: int**
Seed for the random number generator (default: 42)

**test_batch_size: int**
Number of instances used for testing at the same time (default: 128)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.

- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the LSTM-AD algorithm.

**Return type**
*Algorithm*

### 3.3.41 timeeval.algorithms.lstm_vae

timeeval.algorithms.**lstm_vae**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

LSTM-VAE

self implementation of: https://ieeexplore.ieee.org/document/8279425

**Algorithm Parameters:**

**rnn_hidden_size: int**
LTSM cells hidden dimension (default: 5)

**latent_size: int**
dimension of latent space (default: 5)

**learning_rate: float**
rate at which the gradients are updated (default: 0.001)

**batch_size: int**
size of batch given for each iteration (default: 32)

**epochs: int**
number of iterations we train the model (default: 10)

**window_size: int**
number of datapoints that the model takes once (default: 10)

**lstm_layers: int**
number of layers in lstm (default: 10)

**early_stopping_delta: float**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: 0.05)

**early_stopping_patience: int**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: 10)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the LSTM-VAE algorithm.

**Return type**
*Algorithm*

### 3.3.42 timeeval.algorithms.median_method

timeeval.algorithms.**median_method**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

MedianMethod

Implementation of https://doi.org/10.1007/s10115-006-0026-6

**Algorithm Parameters:**

**neighbourhood_size: int**
Specifies the number of time steps to look forward and backward for each data point. (default: `100`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the MedianMethod algorithm.

**Return type**
*Algorithm*

### 3.3.43 timeeval.algorithms.mscred

timeeval.algorithms.**mscred**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

MSCRED

Implementation of https://doi.org/10.1609/aaai.v33i01.33011409

**Algorithm Parameters:**

**windows: List[int]**
Number and size of different signature matrices (correlation matrices) to compute as a preprocessing step (default: `[10, 30, 60]`)

**gap_time: int**
Number of points to skip over between the generation of signature matrices (default: `10`)

**window_size: int**
Size of the sliding windows (default: `5`)

**batch_size: int**
Number of instances trained at the same time (default: `32`)

**learning_rate: float**
Learning rate for Adam optimizer (default: `0.001`)

**epochs: int**
Number of training iterations over entire dataset (default: `1`)

**early_stopping_patience: int**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `10`)

**early_stopping_delta: float**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `0.05`)

**split: float**
Train-validation split for early stopping (default: `0.8`)

**test_batch_size: int**
Number of instances used for validation and testing at the same time (default: `256`)

**random_state: int**
Seed for the random number generator (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the MSCRED algorithm.

**Return type**
*Algorithm*

### 3.3.44 timeeval.algorithms.mstamp

timeeval.algorithms.**mstamp**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

mSTAMP

Implementation of http://www.cs.ucr.edu/%7Eeamonn/Motif_Discovery_ICDM.pdf

**Algorithm Parameters:**

**anomaly_window_size: int**
Size of the sliding windows (default: `50`)

**random_state: int**
Seed for the random number generator (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the mSTAMP algorithm.

**Return type**
*Algorithm*

### 3.3.45 timeeval.algorithms.mtad_gat

timeeval.algorithms.**mtad_gat**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

MTAD-GAT

Implementation of http://arxiv.org/abs/2009.02040

**Algorithm Parameters:**

**mag_window_size: int**
Window size for sliding window average calculation (default: `3`)

**score_window_size: int**
Window size for anomaly scoring (default: `40`)

**threshold: float**
Threshold for SR cleaning (default: `3`)

**context_window_size: int**
Window for mean in SR cleaning (default: `5`)

**kernel_size: int**
Kernel size for 1D-convolution (default: `7`)

**learning_rate: float**
Learning rate for training (default: `0.001`)

**epochs: int**
Number of times the algorithm trains on the dataset (default: `1`)

**batch_size: int**
Number of data points propagated in parallel (default: `64`)

**window_size: int**
Window size for windowing of Time Series (default: `20`)

**gamma: float**
Importance factor for posterior in scoring (default: `0.8`)

**latent_size: int**
Embedding size in VAE (default: `300`)

**linear_layer_shape: List[int]**
Architecture of FC-NN (default: `[300, 300, 300]`)

**early_stopping_patience: int**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `10`)

**early_stopping_delta: float**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `0.05`)

**split: float**
Train-validation split for early stopping (default: `0.8`)

**random_state: int**
Seed for the random number generator (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm

- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the MTAD-GAT algorithm.

**Return type**
*Algorithm*

### 3.3.46 timeeval.algorithms.multi_hmm

timeeval.algorithms.**multi_hmm**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

MultiHMM

Implementation of https://doi.org/10.1016/j.asoc.2017.06.035

**Algorithm Parameters:**

**discretizer: enum[sugeno,choquet,fcm]**
Available discretizers are "sugeno", "choquet", and "fcm". If only 1 feature in time series, K-Bins discretizer is used. (default: fcm)

**n_bins: int**
Number of bins used for discretization. (default: 10)

**random_state: int**
Seed for random number generation. (default: 42)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the MultiHMM algorithm.

**Return type**
*Algorithm*

### 3.3.47 timeeval.algorithms.multi_norma

timeeval.algorithms.**multi_norma**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

MultiNormA

Improved algorithm based on NorM (https://doi.org/10.1109/ICDE48307.2020.00182).

> **Warning:** The implementation of this algorithm is not publicly available (closed source). Thus, TimeEval will fail to download the Docker image and the algorithm will not be available. Please contact the authors of the algorithm for the implementation and build the algorithm Docker image yourself.

**Algorithm Parameters:**

**anomaly_window_size: int**
Sliding window size used to create subsequences (equal to desired anomaly length) (default: `20`)

**normal_model_percentage: float**
Percentage of (random) subsequences used to build the normal model. (default: `0.5`)

**max_motifs: int**
Maximum number of used motifs. Important to avoid OOM errors. (default: `4096`)

**random_state: int**
Seed for random number generation. (default: `42`)

**motif_detection: Enum[stomp,random,mixed]**
Algorithm to use for motif detection [random, stomp, mixed]. (default: `mixed`)

**sum_dims: boolean**
Sum all dimensions up before computing dists, otherwise each dim is handled seperately. (default: `False`)

**normalize_join: boolean**
Apply join normalization heuristic. [false = no normalization, true = normalize] (default: `True`)

**join_combine_method: int**
how to combine the join values from all dimensions.[0=sum, 1=max, 2=score dims (based on std, mean, range), 3=weight higher vals, 4=vals**channels] (default: `1`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the MultiNormA algorithm.

**Return type**
*Algorithm*

### 3.3.48 timeeval.algorithms.multi_subsequence_lof

`timeeval.algorithms.`**`multi_subsequence_lof`**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Multi-Sub-LOF

LOF on sliding windows of multivariate time series to detect subsequence anomalies.

**Algorithm Parameters:**

**window_size: int**
Size of the sliding windows to extract subsequences as input to LOF. (default: `100`)

**n_neighbors: int**
Number of neighbors to use by default for *kneighbors* queries. If n_neighbors is larger than the number of samples provided, all samples will be used. (default: `20`)

**leaf_size: int**
Leaf size passed to *BallTree* or *KDTree*. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem. (default: `30`)

**distance_metric_order: int**
Parameter for the Minkowski metric from sklearn.metrics.pairwise.pairwise_distances. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used. See http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.pairwise_distances. (default: `2`)

**dim_aggregation_method: enum[concat,sum]**
Method used to aggregate multiple dimensions, so that LOF can process the subsequence. When 'concat', the 2D-matrix is flattened into a 1D-vector; when 'sum' the 2D-matrix is aggregated over the channels to get a 1D sliding window. (default: `concat`)

**n_jobs: int**
The number of parallel jobs to run for neighbors search. If `-1`, then the number of jobs is set to the number of CPU cores. Affects only kneighbors and kneighbors_graph methods. (default: `1`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Multi-Sub-LOF algorithm.

**Return type**
*Algorithm*

### 3.3.49 timeeval.algorithms.mvalmod

`timeeval.algorithms.`**`mvalmod`**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

mVALMOD

Implementation of https://doi.org/10.1007/s10618-020-00685-w summed up for every channel.

**Algorithm Parameters:**

**min_anomaly_window_size: Int**
Minimum sliding window size (default: `30`)

**max_anomaly_window_size: Int**
Maximum sliding window size (default: `40`)

**heap_size: Int**
Size of the distance profile heap buffer (default: `50`)

**exclusion_zone: Float**
Size of the exclusion zone as a factor of the window_size. This prevents self-matches. (default: `0.5`)

**verbose: Int**
Controls logging verbosity. (default: `1`)

**random_state: Int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the mVALMOD algorithm.

**Return type**
*Algorithm*

### 3.3.50 timeeval.algorithms.norma

timeeval.algorithms.**norma**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

NormA

Improved algorithm based on NorM (https://doi.org/10.1109/ICDE48307.2020.00182).

> **Warning:** The implementation of this algorithm is not publicly available (closed source). Thus, TimeEval will fail to download the Docker image and the algorithm will not be available. Please contact the authors of the algorithm for the implementation and build the algorithm Docker image yourself.

**Algorithm Parameters:**

**anomaly_window_size: int**
Sliding window size used to create subsequences (equal to desired anomaly length) (default: `20`)

**normal_model_percentage: float**
Percentage of (random) subsequences used to build the normal model. (default: `0.5`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.

- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the NormA algorithm.

**Return type**
*Algorithm*

### 3.3.51 timeeval.algorithms.normalizing_flows

timeeval.algorithms.**normalizing_flows**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Normalizing Flows

Implementation of https://arxiv.org/abs/1912.09323

**Algorithm Parameters:**

**n_hidden_features_factor: float**
Factor deciding how many hidden features for NFs are used based on number of features (default: `1.0`)

**hidden_layer_shape: List[int]**
NN hidden layers structure (default: `[100, 100]`)

**window_size: int**
Window size of sliding window over time series (default: `20`)

**split: float**
Train-validation split (default: `0.9`)

**epochs: int**
Number of training epochs (default: `1`)

**batch_size: int**
How many data instances are trained at the same time. (default: `64`)

**test_batch_size: int**
How many data instances are tested at the same time. (default: `128`)

**teacher_epochs: int**
Number of epochs for teacher NF training (default: `1`)

**distillation_iterations: int**
Number of training steps for distillation (default: `1`)

**percentile: float**
Percentile defining the tails for anomaly sampling. (default: `0.05`)

**early_stopping_patience: int**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `10`)

**early_stopping_delta: float**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `0.05`)

**random_state: int**
Seed for the random number generator (default: `42`)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm

- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Normalizing Flows algorithm.

**Return type**
*Algorithm*

### 3.3.52 timeeval.algorithms.novelty_svr

timeeval.algorithms.**novelty_svr**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

NoveltySVR

Implementation of https://doi.org/10.1145/956750.956828.

**Algorithm Parameters:**

**n_init_train: int**
Number of initial points to fit regression model on. For those points no score is calculated. (default: 500)

**forgetting_time: int**
If this is set, points older than forgetting_time are removed from the model (forgotten) (paper: W) (default: None)

**train_window_size: int**
Size of training windows, also called embedding dimensions, used as context to predict the next point (paper: D) (default: 16)

**anomaly_window_size: int**
Size of event windows, also called event duration, for which suprising occurences are aggregated. Should not be chosen too large! (paper: n) (default: 6)

**lower_suprise_bound: int**
Number of suprising occurences that must be present within an event (see window_size) to regard the event as novel/anomalous (paper: h). Range: 0 < lower_suprise_bound < window_size. If not supplied 'h = window_size / 2' is used as default. (default: None)

**scaling: enum[null,standard,robust,power]**
If the data should be scaled/normalized before regression using StandardScaler, RobustScaler, or PowerTransformer (Yeo-Johnson + standard scaling). See https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#sphx-glr-auto-examples-preprocessing-plot-all-scaling-py. (default: standard)

**epsilon: float**
Specifies epsilon-tube to find suprising occurences in the prediction residuals (resid !> 2eps). Reused as Online SVR parameter: Epsilon in the epsilon-SVR model. It specifies the epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value. (default: 0.1)

**verbose: int**
Controls verbose output. Higher values mean more detailled output [0; 5]. Verbose output of the Online SVR appears not until >=3. (default: 0)

**C: float**
Online SVR parameter: Penalty parameter C of the error term. (default: 1.0)

**kernel: enum[linear,poly,rbf,sigmoid,rbf-gaussian,rbf-exp]**
Online SVR parameter: Specifies the kernel type to be used in the algorithm. (default: `rbf`)

**degree: int**
Online SVR parameter: Degree of the polynomial kernel function ('poly'). Ignored by all other kernels. (default: `3`)

**gamma: float**
Online SVR parameter: Kernel coefficient for 'poly', 'sigmoid', and 'rbf'-kernels. If gamma is None then 1/n_features will be used instead. (default: `None`)

**coef0: float**
Online SVR parameter: Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'. (default: `0.0`)

**tol: float**
Online SVR parameter: Tolerance for stopping criterion. (default: `0.001`)

**stabilized: boolean**
Online SVR parameter: If stabilization should be used. (default: `true`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the NoveltySVR algorithm.

**Return type**
*Algorithm*

### 3.3.53 timeeval.algorithms.numenta_htm

timeeval.algorithms.**numenta_htm**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

NumentaHTM

Implementation of https://doi.org/10.1016/j.neucom.2017.04.070

**Algorithm Parameters:**

**encoding_input_width: int**
(default: `21`)

**encoding_output_width: int**
(default: `50`)

**autoDetectWaitRecords: int**
(default: `50`)

**columnCount: int**
Number of cell columns in the cortical region (same number for SP and TM) (default: `2048`)

**numActiveColumnsPerInhArea: int**
Maximum number of active columns in the SP region's output (when there are more, the weaker ones are suppressed) (default: `40`)

**potentialPct: float**
What percent of the columns's receptive field is available for potential synapses. At initialization time, we will choose potentialPct * (2*potentialRadius+1)^2 (default: `0.5`)

**synPermConnected: float**
The default connected threshold. Any synapse whose permanence value is above the connected threshold is a "connected synapse", meaning it can contribute to the cell's firing. Typical value is 0.10. Cells whose activity level before inhibition falls below minDutyCycleBeforeInh will have their own internal synPermConnectedCell threshold set below this default value. (default: `0.1`)

**synPermActiveInc: float**
(default: `0.1`)

**synPermInactiveDec: float**
(default: `0.005`)

**cellsPerColumn: int**
The number of cells (i.e., states), allocated per column. (default: `32`)

**inputWidth: int**
(default: `2048`)

**newSynapseCount: int**
New Synapse formation count (default: `20`)

**maxSynapsesPerSegment: int**
Maximum number of synapses per segment (default: `32`)

**maxSegmentsPerCell: int**
Maximum number of segments per cell (default: `128`)

**initialPerm: float**
Initial Permanence (default: `0.21`)

**permanenceInc: float**
Permanence Increment (default: `0.1`)

**permanenceDec: float**
Permanence Decrement (default: `0.1`)

**globalDecay: float**
(default: `0.0`)

**maxAge: int**
(default: `0`)

**minThreshold: int**
Minimum number of active synapses for a segment to be considered during search for the best-matching segments. (default: `9`)

**activationThreshold: int**
Segment activation threshold. A segment is active if it has >= tpSegmentActivationThreshold connected synapses that are active due to infActiveState (default: `12`)

**pamLength: int**
"Pay Attention Mode" length. This tells the TM how many new elements to append to the end of a learned sequence at a time. Smaller values are better for datasets with short sequences, higher values are better for datasets with long sequences. (default: `1`)

**alpha: float**
This controls how fast the classifier learns/forgets. Higher values make it adapt faster and forget older patterns faster (default: `0.5`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the NumentaHTM algorithm.

**Return type**
*Algorithm*

### 3.3.54 timeeval.algorithms.ocean_wnn

timeeval.algorithms.**ocean_wnn**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

OceanWNN

Implementation of https://doi.org/10.1016/j.oceaneng.2019.106129

**Algorithm Parameters:**

**train_window_size: int**
Window size used for forecasting the next point (default: `20`)

**hidden_size: int**
Number of neurons in hidden layer (default: `20`)

**batch_size: int**
Number of instances trained at the same time (default: `64`)

**test_batch_size: int**
Batch size over test and validation dataset (default: `256`)

**epochs: int**
Number of training iterations over entire dataset; recommended value: 1000 (default: `1`)

**split: float**
Train-validation split for early stopping (default: `0.8`)

**early_stopping_delta: float**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `0.05`)

**early_stopping_patience: int**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `10`)

**learning_rate: float**
Learning rate for Adam optimizer (default: `0.01`)

**wavelet_a: float**
WBF scale parameter; recommended range: [-2.5, 2.5] (default: `-2.5`)

**wavelet_k: float**
WBF shift parameter; recommended range: [-1.5, 1.5] (default: `-1.5`)

**wavelet_wbf: enum[mexican_hat,central_symmetric,morlet]**
Mother WBF; allowed values: "mexican_hat", "central_symmetric", "morlet" (default: `mexican_hat`)

**wavelet_cs_C: float**
Cosine factor for central-symmetric WBF. (default: `1.75`)

**threshold_percentile: float**
Upper percentile of training residual distribution used for detection replacement. (default: `0.99`)

**random_state: int**
Seed for the random number generator (default: `42`)

**with_threshold: boolean**
If true, values whose forecasting error exceeds the threshold are not included in next window, but are replaced by the prediction. (default: `true`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the OceanWNN algorithm.

**Return type**
*Algorithm*

### 3.3.55 timeeval.algorithms.omnianomaly

timeeval.algorithms.**omnianomaly**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

OmniAnomaly

Implementation of https://doi.org/10.1145/3292500.3330672

**Algorithm Parameters:**

**latent_size: int**
Reduced dimension size (default: `3`)

**rnn_hidden_size: int**
Size of RNN hidden layer (default: `500`)

**window_size: int**
Sliding window size (default: `100`)

**linear_hidden_size: int**
Dense layer size (default: `500`)

**nf_layers: int**
NF layer size (default: `20`)

**epochs: int**
Number of training passes over entire dataset (default: `10`)

**split: float**
Train-validation split (default: `0.8`)

**batch_size: int**
Number of datapoints fitted parallel (default: `50`)

**l2_reg: float**
Regularization factor (default: `0.0001`)

**learning_rate: float**
Learning Rate for Adam Optimizer (default: `0.001`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the OmniAnomaly algorithm.

**Return type**
*Algorithm*

### 3.3.56 timeeval.algorithms.pcc

`timeeval.algorithms.pcc`(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

PCC

Implementation of http://citeseerx.ist.psu.edu/viewdoc/summary;jsessionid=003008C2CF2373B9C332D4A1DB035515?doi=10.1.1.66.299.

**Algorithm Parameters:**

**n_components: int**
Number of components to keep. If *n_components* is not set all components are kept: *n_components == min(n_samples, n_features)*. (default: `None`)

**n_selected_components: int**
Number of selected principal components for calculating the outlier scores. It is not necessarily equal to the total number of the principal components. If not set, use all principal components. (default: `None`)

**whiten: boolean**
When True the *components_* vectors are multiplied by the square root of *n_samples* and then divided by the singular values to ensure uncorrelated outputs with unit component-wise variances. Whitening will remove some information from the transformed signal (the relative variance scales of the components) but can sometime improve the predictive accuracy of the downstream estimators by making their data respect some hard-wired assumptions. (default: `false`)

**svd_solver: enum[auto,full,arpack,randomized]**
'auto': the solver is selected by a default policy based on *X.shape* and *n_components*. If the input data is larger than 500x500 and the number of components to extract is lower than 80% of the smallest dimension

of the data, then the more efficient 'randomized' method is enabled. Otherwise the exact full SVD is computed and optionally truncated afterwards. 'full': run exact full SVD calling the standard LAPACK solver via *scipy.linalg.svd* and select the components by postprocessing. 'arpack': run SVD truncated to n_components calling ARPACK solver via *scipy.sparse.linalg.svds*. It requires strictly *0 < n_components < X.shape[1]*. 'randomized': run randomized SVD by the method of Halko et al. (default: `auto`)

**tol: float**
Tolerance for singular values computed by *svd_solver == 'arpack'*. (default: `0.0`)

**max_iter: int**
Number of iterations for the power method computed by *svd_solver == 'randomized'*. (default: `None`)

**random_state: int**
Used when *svd_solver == 'arpack' or svd_solver == 'randomized'* to seed random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using `ResourceConstraints`.

**Returns**
A correctly configured `Algorithm` object for the PCC algorithm.

**Return type**
`Algorithm`

### 3.3.57 timeeval.algorithms.pci

`timeeval.algorithms.pci`(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

PCI

Implementation of https://doi.org/10.1155/2014/879736

**Algorithm Parameters:**

**window_size: int**
The algorithm uses windows around the current points to predict that point (*k* points before and *k* after, where *k = window_size // 2*). The difference between real and predicted value is used as anomaly score. The parameter *window_size* acts as a kind of smoothing factor. The bigger the *window_size*, the smoother the predictions, the more values have big errors. If *window_size* is too small, anomalies might not be found. *window_size* should correlate with anomaly window sizes. (default: `20`)

**thresholding_p: float**
This parameter is only needed if the algorithm should decide itself whether a point is an anomaly. It treats *p* as a confidence coefficient. It's the t-statistics confidence coefficient. The smaller *p* is, the bigger is the confidence interval. If *p* is too small, anomalies might not be found. If *p* is too big, too many points might be labeled anomalous. (default: `0.05`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the PCI algorithm.

**Return type**
*Algorithm*

### 3.3.58 timeeval.algorithms.phasespace_svm

timeeval.algorithms.**phasespace_svm**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

PhaseSpace-SVM

Implementation of https://doi.org/10.1109/IJCNN.2003.1223670.

**Algorithm Parameters:**

**embed_dim_range: List[int]**
List of phase space dimensions (sliding window sizes). For each dimension a OC-SVM is fitted to calculate outlier scores. The final result is the point-wise aggregation of the anomaly scores. (default: `[50, 100, 150]`)

**project_phasespace: boolean**
Whether to use phasespace projection or just work on the phasespace values. (default: `False`)

**nu: float**
Main parameter of OC-SVM. An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors. Should be in the interval (0, 1]. (default: `0.5`)

**kernel: enum[linear,poly,rbf,sigmoid]**
Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', or 'sigmoid'. (default: `rbf`)

**gamma: float**
Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. If gamma is not set (*null*) then it uses 1 / (n_features * X.var()) as value of gamma (default: `None`)

**degree: int**
Degree of the polynomial kernel function ('poly'). Ignored by all other kernels. (default: `3`)

**coef0: float**
Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'. (default: `0.0`)

**tol: float**
Tolerance for stopping criterion. (default: `0.001`)

**random_state: int**
Seed for random number generation. (default: `42`)

**use_column_index: int**
The column index to use as input for the univariate algorithm for multivariate datasets. The selected single channel of the multivariate time series is analyzed by the algorithms. The index is 0-based and does not include the index-column ('timestamp'). The single channel of an univariate dataset, therefore, has index 0. (default: `0`)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the PhaseSpace-SVM algorithm.

**Return type**
*Algorithm*

### 3.3.59 timeeval.algorithms.pst

timeeval.algorithms.**pst**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

PST

Implementation of a modified version (with preceding discretization) of https://doi.org/10.1137/1.9781611972764.9.

**Algorithm Parameters:**

**window_size: int**
Length of the subsequences in which the time series should be splitted into (sliding window). (default: 5)

**max_depth: int**
Maximal depth of the PST. Default to maximum length of the sequence(s) in object minus 1. (default: 4)

**n_min: int**
Minimum number of occurences of a string to add it in the tree. (default: 1)

**y_min: float**
Smoothing parameter for conditional probabilities, assuring that nosymbol, and hence no sequence, is predicted to have a null probability. The parameter $ymin$ sets a lower bound for a symbol's probability. (default: None)

**n_bins: int**
Number of Bags (bins) in which the time-series should be splitted by frequency. (default: 5)

**sim: enum[SIMo,SIMn]**
The similarity measure to use when computing the similarity between a sequence and the pst. SIMn is supposed to yield better results. (default: SIMn)

**random_state: int**
Seed for random number generation. (default: 42)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the PST algorithm.

**Return type**
*Algorithm*

### 3.3.60 timeeval.algorithms.random_black_forest

timeeval.algorithms.**random_black_forest**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Random Black Forest (RR)

An ensemble of multiple multi-output random forest regressors based on different feature subsets (requested by RollsRoyce). The forecasting error is used as anomaly score.

**Algorithm Parameters:**

**train_window_size: int**
Size of the training windows. Always predicts a single point! (default: `50`)

**n_estimators: int**
The number of forests. Each forest is trained on *max_features* features. (default: `2`)

**max_features_per_estimator: float**
Each forest is trained on randomly selected *int(max_features * n_features)* features. (default: `0.5`)

**n_trees: int**
The number of trees in the forest. (default: `100`)

**max_features_method: enum[auto,sqrt,log2]**
The number of features to consider when looking for the best split between trees: 'auto': max_features=n_features, 'sqrt': max_features=sqrt(n_features), 'log2': max_features=log2(n_features). (default: `auto`)

**bootstrap: boolean**
Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree. (default: `True`)

**max_samples: float**
If bootstrap is True, the number of samples to draw from X to train each base estimator. (default: `None`)

**random_state: int**
Seeds the randomness of the bootstrapping and the sampling of the features. (default: `42`)

**verbose: int**
Controls logging verbosity. (default: `0`)

**n_jobs: int**
The number of jobs to run in parallel. *-1* means using all processors (default: `1`)

**max_depth: int**
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. (default: `None`)

**min_samples_split: int**
The minimum number of samples required to split an internal node. (default: `2`)

**min_samples_leaf: int**
The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least *min_samples_leaf* training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression. (default: `1`)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Random Black Forest (RR) algorithm.

**Return type**
*Algorithm*

### 3.3.61 timeeval.algorithms.robust_pca

timeeval.algorithms.**robust_pca**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

RobustPCA

Implementation of https://arxiv.org/pdf/1801.01571.pdf

**Algorithm Parameters:**

**max_iter: int**
Defines the number of maximum robust PCA iterations for solving matrix decomposition. (default: 1000)

**random_state: int**
Seed for random number generation. (default: 42)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the RobustPCA algorithm.

**Return type**
*Algorithm*

### 3.3.62 timeeval.algorithms.s_h_esd

timeeval.algorithms.**s_h_esd**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

S-H-ESD (Twitter)

Implementation of http://citeseerx.ist.psu.edu/viewdoc/summary;jsessionid=003008C2CF2373B9C332D4A1DB035515?doi=10.1.1.66.299

**Algorithm Parameters:**

**max_anomalies: float**
expected maximum relative frequency of anomalies in the dataset (default: `0.05`)

**timestamp_unit: enum[m,h,d]**
If the index column ('timestamp') is of type integer, this gives the unit for date conversion. A unit less than seconds is not supported by S-H-ESD! (default: `m`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the S-H-ESD (Twitter) algorithm.

**Return type**
*Algorithm*

### 3.3.63 timeeval.algorithms.sand

`timeeval.algorithms.`**sand**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

SAND

Implementation of SAND described in http://www.vldb.org/pvldb/vol14/p1717-boniol.pdf.

> **Warning:** The implementation of this algorithm is not publicly available (closed source). Thus, TimeEval will fail to download the Docker image and the algorithm will not be available. Please contact the authors of the algorithm for the implementation and build the algorithm Docker image yourself.

**Algorithm Parameters:**

**anomaly_window_size: int**
Size of the anomalous pattern; sliding windows for clustering and preprocessing are of size 3*anomaly_window_size. (default: `75`)

**n_clusters: int**
Number of clusters used in Kshape that are maintained iteratively as a normal model (default: `6`)

**n_init_train: int**
Number of points to build the initial model (may contain anomalies) (default: `2000`)

**iter_batch_size: int**
Number of points for each batch. Mostly impacts performance (not too small). (default: `500`)

**alpha: float**
Weight decay / forgetting factor. Quite robust (default: `0.5`)

**random_state: int**
Seed for random number generation. (default: `42`)

**use_column_index: int**
The column index to use as input for the univariate algorithm for multivariate datasets. The selected single channel of the multivariate time series is analyzed by the algorithms. The index is 0-based and does not include the index-column ('timestamp'). The single channel of an univariate dataset, therefore, has index 0. (default: `0`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the SAND algorithm.

**Return type**
*Algorithm*

### 3.3.64 timeeval.algorithms.sarima

timeeval.algorithms.**sarima**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

SARIMA

Implementation of SARIMA method described in https://milets18.github.io/papers/milets18_paper_19.pdf.

**Algorithm Parameters:**

**train_window_size: int**
Number of points from the beginning of the series to build model on. (default: `500`)

**prediction_window_size: int**
Number of points to forecast in one go; smaller = slower, but more accurate. (default: `10`)

**max_lag: int**
Refit SARIMA model after that number of points (only helpful if fixed_orders=None) (default: `None`)

**period: int**
Periodicity (number of periods in season), often it is 4 for quarterly data or 12 for monthly data. Default is no seasonal effect (==1). Must be >= 1. (default: `1`)

**max_iter: int**
The maximum number of function evaluations. smaller = faster, but might not converge. (default: `20`)

**exhaustive_search: boolean**
Performs full grid search to find optimal SARIMA-model without considering statistical tests on the data –> SLOW! but finds the optimal model. (default: `false`)

**n_jobs: int**
The number of parallel jobs to run for grid search. If `-1`, then the number of jobs is set to the number of CPU cores. (default: `1`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the SARIMA algorithm.

**Return type**
*Algorithm*

### 3.3.65 timeeval.algorithms.series2graph

timeeval.algorithms.**series2graph**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Series2Graph

Implementation of https://doi.org/10.14778/3407790.3407792.

**Warning:** The implementation of this algorithm is not publicly available (closed source). Thus, TimeEval will fail to download the Docker image and the algorithm will not be available. Please contact the authors of the algorithm for the implementation and build the algorithm Docker image yourself.

**Algorithm Parameters:**

**window_size: Int**
Size of the sliding window (paper: *l*), independent of anomaly length, but should in the best case be larger. (default: 50)

**query_window_size: Int**
Size of the sliding windows used to find anomalies (query subsequences). query_window_size must be >= window_size! (paper: *l_q*) (default: 75)

**rate: Int**
Number of angles used to extract pattern nodes. A higher value will lead to high precision, but at the cost of increased computation time. (paper: *r* performance parameter) (default: 30)

**random_state: Int**
Seed for random number generation. (default: 42)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Series2Graph algorithm.

**Return type**
*Algorithm*

### 3.3.66 timeeval.algorithms.sr

timeeval.algorithms.**sr**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Spectral Residual (SR)

Implementation of https://doi.org/10.1145/3292500.3330680

**Algorithm Parameters:**

**mag_window_size: int**
Window size for sliding window average calculation (default: `3`)

**score_window_size: int**
Window size for anomaly scoring (default: `40`)

**window_size: int**
Sliding window size (default: `50`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Spectral Residual (SR) algorithm.

**Return type**
*Algorithm*

### 3.3.67 timeeval.algorithms.sr_cnn

timeeval.algorithms.**sr_cnn**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

SR-CNN

Implementation of https://doi.org/10.1145/3292500.3330680

**Algorithm Parameters:**

**window_size: int**
Sliding window size (default: `128`)

**random_state: int**
Seed for random number generators (default: `42`)

**step: int**
stride size for training data generation (default: `64`)

**num: int**
Max value for generated data (default: `10`)

**learning_rate: float**
Gradient factor during SGD training (default: `1e-06`)

**epochs: int**
Number of training passes over entire dataset (default: `1`)

**batch_size: int**
Number of data points trained in parallel (default: `256`)

**n_jobs: int**
Number of processes used during training (default: `1`)

**split: float**
Train-validation split for early stopping (default: `0.9`)

**early_stopping_delta: float**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `0.05`)

**early_stopping_patience: int**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `10`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using `ResourceConstraints`.

**Returns**
A correctly configured `Algorithm` object for the SR-CNN algorithm.

**Return type**
`Algorithm`

### 3.3.68 timeeval.algorithms.ssa

`timeeval.algorithms.ssa`(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

SSA

Segmented Sequence Analysis calculates two piecewise linear models, aligns them and then computes the similarity between them. Finally a treshhold based approach is used to classify data as anomalous.

> **Warning:** The implementation of this algorithm is not publicly available (closed source). Thus, TimeEval will fail to download the Docker image and the algorithm will not be available. Please contact the authors of the algorithm for the implementation and build the algorithm Docker image yourself.

**Algorithm Parameters:**

**ep: int**
Score normalization value (default: `3`)

**window_size: int**
Size of sliding window. (default: `20`)

**rf_method: Enum[all,alpha]**
*all*: Directly calculate reference timeseries from all points. *alpha*: Create weighted reference timeseries with help of parameter 'a' (default: `alpha`)

**alpha: float**
Describes weights that are used for reference time series creation. Can be a single weight(float) or an array of weights. So far only supporting a single value (default: `0.2`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the SSA algorithm.

**Return type**
*Algorithm*

### 3.3.69 timeeval.algorithms.stamp

timeeval.algorithms.**stamp**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

STAMP

Implementation of https://doi.org/10.1109/ICDM.2016.0179.

**Algorithm Parameters:**

**anomaly_window_size: Int**
Size of the sliding window. (default: `30`)

**exclusion_zone: Float**
Size of the exclusion zone as a factor of the window_size. This prevents self-matches. (default: `0.5`)

**verbose: Int**
Controls logging verbosity. (default: `1`)

**n_jobs: Int**
The number of jobs to run in parallel. *-1* is not supported, defaults back to serial implementation. (default: `1`)

**random_state: Int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.

- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the STAMP algorithm.

**Return type**
*Algorithm*

### 3.3.70 timeeval.algorithms.stomp

timeeval.algorithms.**stomp**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

STOMP

Implementation of https://doi.org/10.1109/ICDM.2016.0085.

**Algorithm Parameters:**

**anomaly_window_size: Int**
Size of the sliding window. (default: 30)

**exclusion_zone: Float**
Size of the exclusion zone as a factor of the window_size. This prevents self-matches. (default: 0.5)

**verbose: Int**
Controls logging verbosity. (default: 1)

**n_jobs: Int**
The number of jobs to run in parallel. *-1* is not supported, defaults back to serial implementation. (default: 1)

**random_state: Int**
Seed for random number generation. (default: 42)

**use_column_index: int**
The column index to use as input for the univariate algorithm for multivariate datasets. The selected single channel of the multivariate time series is analyzed by the algorithms. The index is 0-based and does not include the index-column ('timestamp'). The single channel of an univariate dataset, therefore, has index 0. (default: 0)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the STOMP algorithm.

**Return type**
*Algorithm*

### 3.3.71 timeeval.algorithms.subsequence_fast_mcd

timeeval.algorithms.**subsequence_fast_mcd**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Subsequence Fast-MCD

Implementation of https://doi.org/10.2307/1270566 with sliding windows as input

**Algorithm Parameters:**

**store_precision: boolean**
Specify if the estimated precision is stored (default: `True`)

**support_fraction: float**
The proportion of points to be included in the support of the raw MCD estimate. Default is None, which implies that the minimum value of support_fraction will be used within the algorithm: *(n_sample + n_features + 1) / 2*. The parameter must be in the range (0, 1). (default: `None`)

**random_state: int**
Determines the pseudo random number generator for shuffling the data. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using `ResourceConstraints`.

**Returns**
A correctly configured `Algorithm` object for the Subsequence Fast-MCD algorithm.

**Return type**
`Algorithm`

### 3.3.72 timeeval.algorithms.subsequence_if

timeeval.algorithms.**subsequence_if**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Subsequence IF

Isolation Forest on sliding windows to detect subsequence anomalies.

**Algorithm Parameters:**

**window_size: int**
Size of the sliding windows to extract subsequences as input to LOF. (default: `100`)

**n_trees: int**
The number of decision trees (base estimators) in the forest (ensemble). (default: `100`)

**max_samples: float**
The number of samples to draw from X to train each base estimator: *max_samples * X.shape[0]*. If unspecified (*null*), then *max_samples=min(256, n_samples)*. (default: `None`)

**max_features: float**
The number of features to draw from X to train each base estimator: *max_features * X.shape[1]*. (default: `1.0`)

**bootstrap: boolean**
If True, individual trees are fit on random subsets of the training data sampled with replacement. If False, sampling without replacement is performed. (default: `false`)

**random_state: int**
Seed for random number generation. (default: `42`)

**verbose: int**
Controls the verbosity of the tree building process logs. (default: `0`)

**n_jobs: int**
The number of jobs to run in parallel. If -1, then the number of jobs is set to the number of cores. (default: 1)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Subsequence IF algorithm.

**Return type**
*Algorithm*

### 3.3.73 timeeval.algorithms.subsequence_knn

timeeval.algorithms.**subsequence_knn**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Sub-KNN

KNN on sliding windows to detect subsequence anomalies.

**Algorithm Parameters:**

**window_size: int**
Size of the sliding windows to extract subsequences as input to LOF. (default: `100`)

**n_neighbors: int**
Number of neighbors to use by default for *kneighbors* queries. (default: `5`)

**leaf_size: int**
Leaf size passed to *BallTree*. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem. (default: `30`)

**method: enum[largest,mean,median]**
'largest': use the distance to the kth neighbor as the outlier score, 'mean': use the average of all k neighbors as the outlier score, 'median': use the median of the distance to k neighbors as the outlier score. (default: `largest`)

**radius: float**
Range of parameter space to use by default for *radius_neighbors* queries. (default: `1.0`)

**distance_metric_order: int**
Parameter for the Minkowski metric from sklearn.metrics.pairwise.pairwise_distances. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p,

minkowski_distance (l_p) is used. See http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.pairwise_distances. (default: 2)

**n_jobs: int**
The number of parallel jobs to run for neighbors search. If -1, then the number of jobs is set to the number of CPU cores. Affects only kneighbors and kneighbors_graph methods. (default: 1)

**random_state: int**
Seed for random number generation. (default: 42)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Sub-KNN algorithm.

**Return type**
*Algorithm*

### 3.3.74 timeeval.algorithms.subsequence_lof

timeeval.algorithms.**subsequence_lof**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Subsequence LOF

LOF on sliding windows to detect subsequence anomalies.

**Algorithm Parameters:**

**window_size: int**
Size of the sliding windows to extract subsequences as input to LOF. (default: 100)

**n_neighbors: int**
Number of neighbors to use by default for *kneighbors* queries. If n_neighbors is larger than the number of samples provided, all samples will be used. (default: 20)

**leaf_size: int**
Leaf size passed to *BallTree* or *KDTree*. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem. (default: 30)

**distance_metric_order: int**
Parameter for the Minkowski metric from sklearn.metrics.pairwise.pairwise_distances. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used. See http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.pairwise_distances. (default: 2)

**n_jobs: int**
The number of parallel jobs to run for neighbors search. If -1, then the number of jobs is set to the number of CPU cores. Affects only kneighbors and kneighbors_graph methods. (default: 1)

**random_state: int**
Seed for random number generation. (default: 42)

**use_column_index: int**
The column index to use as input for the univariate algorithm for multivariate datasets. The selected single channel of the multivariate time series is analyzed by the algorithms. The index is 0-based and does not include the index-column ('timestamp'). The single channel of an univariate dataset, therefore, has index 0. (default: `0`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Subsequence LOF algorithm.

**Return type**
*Algorithm*

### 3.3.75 timeeval.algorithms.tanogan

timeeval.algorithms.**tanogan**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

TAnoGan

Implementation of http://arxiv.org/abs/2008.09567

**Algorithm Parameters:**

**epochs: int**
Number of training iterations over entire dataset (default: `1`)

**cuda: boolean**
Set to *true*, if the GPU-backend (using CUDA) should be used. Otherwise, the algorithm is executed on the CPU. (default: `false`)

**window_size: int**
Size of the sliding windows (default: `30`)

**learning_rate: float**
Learning rate for Adam optimizer (default: `0.0002`)

**batch_size: int**
Number of instances trained at the same time (default: `32`)

**n_jobs: int**
Number of workers (processes) used to load and preprocess the data (default: `1`)

**random_state: int**
Seed for random number generation. (default: `42`)

**early_stopping_patience: int**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `10`)

**early_stopping_delta: float**
If 1 - (loss / last_loss) is less than *delta* for *patience* epochs, stop (default: `0.05`)

**split: float**
Train-validation split for early stopping (default: `0.8`)

**iterations: int**
Number of test iterations per window (default: `25`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the TAnoGan algorithm.

**Return type**
*Algorithm*

### 3.3.76 timeeval.algorithms.tarzan

timeeval.algorithms.**tarzan**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

TARZAN

Implementation of https://dl.acm.org/doi/10.1145/775047.775128

**Algorithm Parameters:**

**random_state: int**
Seed for random number generation. (default: `42`)

**anomaly_window_size: int**
Size of the sliding window. Equal to the discord length! (default: `20`)

**alphabet_size: int**
Number of symbols used for discretization by SAX (performance parameter) (default: `4`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the TARZAN algorithm.

**Return type**
*Algorithm*

### 3.3.77 timeeval.algorithms.telemanom

timeeval.algorithms.**telemanom**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Telemanom

Implementation of https://doi.org/10.1145/3219819.3219845.

**Algorithm Parameters:**

**batch_size: Int**
number of values to evaluate in each batch (default: `70`)

**smoothing_window_size: Int**
number of trailing batches to use in error calculation (default: `30`)

**smoothing_perc: Float**
etermines window size used in EWMA smoothing (percentage of total values for channel) (default: `0.05`)

**error_buffer: Int**
number of values surrounding an error that are brought into the sequence (promotes grouping on nearby sequences) (default: `100`)

**dropout: Float**
LSTM dropout probability (default: `0.3`)

**lstm_batch_size: Int**
number of vlaues to evaluate in one batch for the LSTM (default: `64`)

**epochs: Int**
Number of training iterations over entire dataset (default: `35`)

**split: Float**
Train-validation split for early stopping (default: `0.8`)

**early_stopping_patience: Int**
If loss is *delta* or less smaller for *patience* epochs, stop (default: `10`)

**early_stopping_delta: Float**
If loss is *delta* or less smaller for *patience* epochs, stop (default: `0.0003`)

**window_size: Int**
num previous timesteps provided to model to predict future values (default: `250`)

**prediction_window_size: Int**
number of steps to predict ahead (default: `10`)

**p: Float**
minimum percent decrease between max errors in anomalous sequences (used for pruning) (default: `0.13`)

**random_state: int**
Seed for the random number generator (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Telemanom algorithm.

**Return type**
*Algorithm*

### 3.3.78 timeeval.algorithms.torsk

timeeval.algorithms.**torsk**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Torsk

Implementation of http://arxiv.org/abs/1909.01709

**Algorithm Parameters:**

**input_map_size: int**
Size of the random weight preprocessing latent space. *input_map_size* must be larger than or equal to *context_window_size*! (default: `100`)

**input_map_scale: float**
Feature scaling of the random weight preprocessing. (default: `0.125`)

**context_window_size: int**
Size of a tumbling window used to encode the time series into a 2D (image-based) representation, called slices (default: `10`)

**train_window_size: int**
Torsk creates the input subsequences by sliding a window of size *train_window_size + prediction_window_size + 1* over the slices with shape (context_window_size, dim). *train_window_size* represents the size of the input windows for training and prediction (default: `50`)

**prediction_window_size: int**
Torsk creates the input subsequences by sliding a window of size *train_window_size + prediction_window_size + 1* over the slices with shape (context_window_size, dim). *prediction_window_size* represents the size of the ESN predictions, should be *min_anomaly_length < prediction_window_size < 10 * min_anomaly_length* (default: `20`)

**transient_window_size: int**
Just a part of the training window, the first *transient_window_size* slices, are used for the ESN optimization. (default: `10`)

**spectral_radius: float**
ESN hyperparameter that determines the influence of previous internal ESN state on the next one. *spectral_radius > 1.0* increases non-linearity, but decreases short-term-memory capacity (maximized at 1.0) (default: `2.0`)

**density: float**
Density of the ESN cell, where approx. *density* percent of elements being non-zero (default: `0.01`)

**reservoir_representation: enum[sparse,dense]**
Representation of the ESN reservoirs. *sparse* is significantly faster than *dense* (default: `sparse`)

**imed_loss: boolean**
Calculate loss on spatially aware (image-based) data representation instead of flat arrays (default: `False`)

**train_method: enum[pinv_lstsq,pinv_svd,tikhonov]**
Solver used to train the ESN. *tikhonov* - linear solver with tikhonov regularization, *pinv_lstsq* - exact least-squares-solver that may lead to a numerical blowup, *pinv_svd* - SVD-based least-squares-solver that is highly numerically stable, but approximate (default: `pinv_svd`)

**tikhonov_beta: float**
Parameter of the Tikhonov regularization term when *train_method* = *tikhonov* is used. (default: `None`)

**verbose: int**
Controls the logging output (default: `2`)

**scoring_small_window_size: int**
Size of the smaller of two windows slid over the prediction errors to calculate the final anomaly scores. (default: `10`)

**scoring_large_window_size: int**
Size of the larger of two windows slid over the prediction errors to calculate the final anomaly scores. (default: `100`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Torsk algorithm.

**Return type**
*Algorithm*

### 3.3.79 timeeval.algorithms.triple_es

timeeval.algorithms.**triple_es**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

Triple ES (Holt-Winter's)

Implementation of http://www.diva-portal.org/smash/get/diva2:1198551/FULLTEXT02.pdf

**Algorithm Parameters:**

**train_window_size: int**
size of each TripleES model to predict the next timestep (default: `200`)

**period: int**
number of time units at which events happen regularly/periodically (default: `100`)

**trend: enum[add, mul]**
type of trend component (default: `add`)

**seasonal: enum[add, mul]**
type of seasonal component (default: `add`)

**random_state: int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm

- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the Triple ES (Holt-Winter's) algorithm.

**Return type**
*Algorithm*

### 3.3.80 timeeval.algorithms.ts_bitmap

timeeval.algorithms.**ts_bitmap**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

TSBitmap

Implementation of https://dl.acm.org/doi/abs/10.5555/1116877.1116907

**Algorithm Parameters:**

**feature_window_size: int**
Size of the tumbling windows used for SAX discretization. (default: 100)

**lead_window_size: int**
How far to look ahead to create lead bitmap. (default: 200)

**lag_window_size: int**
How far to look back to create the lag bitmap. (default: 300)

**alphabet_size: int**
Number of bins for SAX discretization. (default: 5)

**level_size: int**
Desired level of recursion of the bitmap. (default: 3)

**compression_ratio: int**
How much to compress the timeseries in the PAA step. If *compression_ration == 1*, no compression. (default: 2)

**random_state: int**
Seed for random number generation. (default: 42)

**Parameters**

- **params** (Optional[ParameterConfig]) – Parameter configuration for the algorithm
- **skip_pull** (bool) – Set to True to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (Optional[Duration]) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the TSBitmap algorithm.

**Return type**
*Algorithm*

### 3.3.81 timeeval.algorithms.valmod

timeeval.algorithms.**valmod**(*params: Optional[ParameterConfig] = None*, *skip_pull: bool = False*, *timeout: Optional[Duration] = None*) → *Algorithm*

VALMOD

Implementation of https://doi.org/10.1007/s10618-020-00685-w.

**Algorithm Parameters:**

**min_anomaly_window_size: Int**
Minimum sliding window size (default: `30`)

**max_anomaly_window_size: Int**
Maximum sliding window size (default: `40`)

**heap_size: Int**
Size of the distance profile heap buffer (default: `50`)

**exclusion_zone: Float**
Size of the exclusion zone as a factor of the window_size. This prevents self-matches. (default: `0.5`)

**verbose: Int**
Controls logging verbosity. (default: `1`)

**random_state: Int**
Seed for random number generation. (default: `42`)

**Parameters**

- **params** (`Optional[ParameterConfig]`) – Parameter configuration for the algorithm
- **skip_pull** (`bool`) – Set to `True` to skip pulling the Docker image and use a local image instead. If the image is not present locally, this will raise an error.
- **timeout** (`Optional[Duration]`) – Set an individual execution and training timeout for this algorithm. This will overwrite the global timeouts set using *ResourceConstraints*.

**Returns**
A correctly configured *Algorithm* object for the VALMOD algorithm.

**Return type**
*Algorithm*

## 3.4 timeeval.datasets package

### 3.4.1 timeeval.datasets.analyzer

**class** timeeval.datasets.analyzer.**DatasetAnalyzer**(*dataset_id: Tuple[str, str]*, *is_train: bool*, *df: Optional[DataFrame] = None*, *dataset_path: Optional[Path] = None*, *dmgr: Optional[Datasets] = None*, *ignore_stationarity: bool = False*, *ignore_trend: bool = False*)

Utility class to analyze a dataset and infer metadata about the dataset.

Use this class to compute necessary metadata from a time series. The computation is started directly when instantiating this class. You can access the results using the property `metadata`. There are multiple ways to instantiate this class, but you always have to specify the dataset ID, because it is part of the metadata:

1. Use an existing pandas data frame object. Supply a value to the parameter *df*.
2. Use a path to a time series. Supply a value to the parameter *dataset_path*.
3. Use a dataset ID and a reference to the dataset manager. Supply a value to the parameter *dmgr*.

This class computes simple metadata, such as number of anomalies, mean, and standard deviation, as well as advanced metadata, such as trends or stationarity information for all time series channels. The simple metadata is exact. But the advanced metadata is estimated based on the observed time series data. The trend is computed by fitting linear regression models of different order to the time series. If the regression has a high enough correlation with the observed values, the trends and their confidence are recorded. The stationarity of the time series is estimated using two statistical tests, the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) and the Augmented Dickey Fuller (ADF) test.

The metadata of a dataset can be stored to disk. This class provides utility functions to create a JSON-file per dataset, containing the metadata about the test time series and the optional training time series.

**Parameters**

- **dataset_id** (tuple of str, str) – ID of the dataset consisting of collection and dataset name.
- **is_train** (bool) – If the analyzed time series is the testing or training time series of the dataset.
- **df** (data frame, *optional*) – Time series data frame. If *df* is supplied, you can omit *dataset_path* and *dmgr*.
- **dataset_path** (path, *optional*) – Path to the time series. If *dataset_path* is supplied, you can omit *df* and *dmgr*.
- **dmgr** (Datasets object, *optional*) – Dataset manager instance that is used to load the time series if *df* and *dataset_path* are not specified.
- **ignore_stationarity** (bool, *optional*) – Don't estimate the time series' channels stationarity. This might be necessary for large datasets, because this step takes a lot of time.
- **ignore_trend** (bool, *optional*) – Don't estimate the time series' channels trend type. This might be necessary for large datasets, because this step takes a lot of time.

**See also:**

statsmodels.tsa.stattools.adfuller, statsmodels.tsa.stattools.kpss

**static load_from_json**(*filename: Union[str, Path]*, *train: bool = False*) → *DatasetMetadata*

Loads existing time series metadata from disk.

If there are multiple metadata entries with the same dataset ID and training/testing-label, the first entry is used.

**Parameters**

- **filename** (path) – Path to the JSON-file containing the dataset metadata. Can be written using *timeeval.datasets.analyzer.DatasetAnalyzer.save_to_json()*.
- **train** (bool) – Whether the training or testing time series' metadata should be loaded from the file.

**Returns**

**metadata** – Metadata of the training or testing time series.

**Return type**

time series metadata object

**property metadata:** *DatasetMetadata*

Returns the computed metadata about the time series.

**save_to_json**(*filename: Union[str, Path]*, *overwrite: bool = False*) → None

Save the computed metadata for a dataset to disk.

This method writes a dataset's metadata to a JSON-formatted file to disk. The file contains a list of metadata specifications. One specification for the test time series and potentially another one for the train time series. Since the DatasetAnalyzer just analyzes a single time series at a time, this method appends the current metadata to the existing list per default. If you want to overwrite the existing content of the file, you can use the parameter *overwrite*.

**Parameters**

- **filename** (path) – Path to the file, where the metadata should be written to. Might already exist.
- **overwrite** (bool) – If existing data in the file should be overwritten or the current metadata should just be added to it.

### 3.4.2 timeeval.datasets.custom

**class** timeeval.datasets.custom.**CDEntry**(*test_path*, *train_path*, *details*)

Bases: NamedTuple

**details:** *Dataset*

Alias for field number 2

**test_path:** Path

Alias for field number 0

**train_path:** Optional[Path]

Alias for field number 1

**class** timeeval.datasets.custom.**CustomDatasets**(*dataset_config: Union[str, Path]*)

Bases: *CustomDatasetsBase*

Implementation of the custom datasets API.

Internal API! You should **not need to use or modify** this class.

This class behaves similar to the *timeeval.datasets.datasets.Datasets*-API while using a different internal representation for the dataset index.

**get**(*dataset_name: str*) → *Dataset*

**get_collection_names**() → List[str]

**get_dataset_names**() → List[str]

**get_path**(*dataset_name: str*, *train: bool*) → Path

**select**(*collection: Optional[str] = None*, *dataset: Optional[str] = None*, *dataset_type: Optional[str] = None*, *datetime_index: Optional[bool] = None*, *training_type: Optional[TrainingType] = None*, *train_is_normal: Optional[bool] = None*, *input_dimensionality: Optional[InputDimensionality] = None*, *min_anomalies: Optional[int] = None*, *max_anomalies: Optional[int] = None*, *max_contamination: Optional[float] = None*) → List[Tuple[str, str]]

### 3.4.3 timeeval.datasets.custom_base

**class** timeeval.datasets.custom_base.**CustomDatasetsBase**

Bases: ABC

API definition for custom datasets.

Internal API! You should **not need to use or modify** this class.

**abstract get**(*dataset_name: str*) → *Dataset*

**abstract get_collection_names**() → List[str]

**abstract get_dataset_names**() → List[str]

**abstract get_path**(*dataset_name: str*, *train: bool*) → Path

**abstract select**(*collection: Optional[str] = None*, *dataset: Optional[str] = None*, *dataset_type: Optional[str] = None*, *datetime_index: Optional[bool] = None*, *training_type: Optional[TrainingType] = None*, *train_is_normal: Optional[bool] = None*, *input_dimensionality: Optional[InputDimensionality] = None*, *min_anomalies: Optional[int] = None*, *max_anomalies: Optional[int] = None*, *max_contamination: Optional[float] = None*) → List[Tuple[str, str]]

### 3.4.4 timeeval.datasets.custom_noop

**class** timeeval.datasets.custom_noop.**NoOpCustomDatasets**

Bases: *CustomDatasetsBase*

Dummy implementation of the CustomDatasets interface.

Internal API! You should **not need to use or modify** this class.

This dummy implementation does nothing and improves readability of the *timeeval.datasets.datasets.Datasets*-implementation by removing the need for None-checks.

**get**(*dataset_name: str*) → *Dataset*

**get_collection_names**() → List[str]

**get_dataset_names**() → List[str]

**get_path**(*dataset_name: str*, *train: bool*) → Path

**select**(*collection: Optional[str] = None*, *dataset: Optional[str] = None*, *dataset_type: Optional[str] = None*, *datetime_index: Optional[bool] = None*, *training_type: Optional[TrainingType] = None*, *train_is_normal: Optional[bool] = None*, *input_dimensionality: Optional[InputDimensionality] = None*, *min_anomalies: Optional[int] = None*, *max_anomalies: Optional[int] = None*, *max_contamination: Optional[float] = None*) → List[Tuple[str, str]]

### 3.4.5 timeeval.datasets.dataset

class timeeval.datasets.dataset.**Dataset**(*datasetId: Tuple[str, str]*, *dataset_type: str*, *training_type: TrainingType*, *length: int*, *dimensions: int*, *contamination: float*, *min_anomaly_length: int*, *median_anomaly_length: int*, *max_anomaly_length: int*, *period_size: Optional[int] = None*, *num_anomalies: Optional[int] = None*)

Bases: object

Dataset information containing basic metadata about the dataset.

This class is used within TimeEval heuristics to determine the heuristic values based on the dataset properties.

**property collection_name:** str

**contamination:** float

**datasetId:** Tuple[str, str]

**dataset_type:** str

**dimensions:** int

**property has_anomalies:** Optional[bool]

**property input_dimensionality:** *InputDimensionality*

**length:** int

**max_anomaly_length:** int

**median_anomaly_length:** int

**min_anomaly_length:** int

**property name:** str

**num_anomalies:** Optional[int] = None

**period_size:** Optional[int] = None

**training_type:** *TrainingType*

### 3.4.6 timeeval.datasets.dataset_manager

class timeeval.datasets.dataset_manager.**DatasetManager**(*data_folder: Union[str, Path]*, *custom_datasets_file: Optional[Union[str, Path]] = None*, *create_if_missing: bool = True*)

Bases: ContextManager[*DatasetManager*], *Datasets*

Manages benchmark datasets and their meta-information.

Manages dataset collections and their meta-information that are stored in a single folder with an index file. You can also use this class to create a new TimeEval dataset collection.

**Warning:** ATTENTION: Not multi-processing-safe! There is no check for changes to the underlying *dataset.csv* file while this class is loaded.

Read-only access is fine with multiple processes.

**Parameters**

- **data_folder** (path) – Path to the folder, where the benchmark data is stored. This folder consists of the file *datasets.csv* and the datasets in a hierarchical storage layout.
- **custom_datasets_file** (path) – Path to a file listing additional custom datasets.
- **create_if_missing** (bool) – Create an index-file in the data_folder if none could be found. Set this to False if an exception should be raised if the folder is wrong or does not exist.

**Raises**

FileNotFoundError – If create_if_missing is set to False and no *datasets.csv*-file was found in the data_folder.

**See also:**

*timeeval.datasets.datasets.Datasets*, *timeeval.datasets.multi_dataset_manager.MultiDatasetManager*

**INDEX_FILENAME: str = 'datasets.csv'**

**METADATA_FILENAME_SUFFIX: str = 'metadata.json'**

**add_dataset**(*dataset:* DatasetRecord) → None

Adds a new dataset to the benchmark dataset collection (in-memory).

The provided dataset metadata is added to this dataset collection (to the in-memory index). You can save the in-memory index to disk using the timeeval.datasets.DatasetManager.save()-method. The referenced time series files (training and testing paths) are not touched. If the same dataset ID (collection_name, dataset_name) than an existing dataset is specified, its entries are overwritten!

**Parameters**

**dataset** (DatasetRecord object) – The dataset information to add to the benchmark collection.

**add_datasets**(*datasets: List[*DatasetRecord*]*) → None

Add a list of datasets to the dataset collection.

Add a list of new datasets to the benchmark dataset collection (in-memory). Already existing keys are overwritten!

**Parameters**

**datasets** (list of DatasetRecord objects) – List of dataset metdata to add to this dataset collection.

**See also:**

*timeeval.datasets.dataset_manager.DatasetManager.add_dataset()*

**df**() → DataFrame

Returns a copy of the internal dataset metadata collection.

The DataFrame has the following schema:

**Index:**
dataset_name, collection_name

**Columns:**
train_path, test_path, dataset_type, datetime_index, split_at, train_type, train_is_normal, input_type, length, dimensions, contamination, num_anomalies, min_anomaly_length, median_anomaly_length, max_anomaly_length, mean, stddev, trend, stationarity, period_size

**Returns**
**df** – All custom and benchmark datasets and their metadata.

**Return type**
`data frame`

**get**(*collection_name: Union[str, Tuple[str, str]]*, *dataset_name: Optional[str] = None*) → *Dataset*

Returns dataset metadata.

### Examples

```
>>> from timeeval.datasets import DatasetManager
>>> dm = DatasetManager("path/to/datasets")
>>> dataset_id = ("custom", "dataset1")
```

Access using the dataset ID:

```
>>> dm.get(dataset_id)
Dataset(datsetId=("custom", "dataset1"), ...)
```

Access using collection and dataset name:

```
>>> dm.get("custom", "dataset1")
Dataset(datsetId=("custom", "dataset1"), ...)
```

**Parameters**

- **collection_name** (`str` or `tuple` of `str` and `str`) – Name of the dataset collection or the dataset ID (collection, and dataset name).
- **dataset_name** (`str`, *optional*) – Name of the dataset or empty.

**Returns**
**dataset** – The dataset metadata

**Return type**
`a dataset object`

**get_collection_names**() → List[str]

Returns the unique dataset collection names (includes custom datasets if present).

**get_dataset_df**(*dataset_id: Tuple[str, str]*, *train: bool = False*) → DataFrame

Loads the training/testing time series as a data frame.

**Parameters**

- **dataset_id** (`tuple` of `str`, `str`) – Dataset ID (collection and dataset name).
- **train** (`bool`) – Whether the training (`True`) or testing (`False`, default) should be loaded.

**Returns**
**df** – The training or testing time series as a `pandas.DataFrame`.

**Return type**
`data frame`

`get_dataset_names()` → List[str]

Returns the unique dataset names (includes custom datasets if present).

`get_dataset_ndarray(dataset_id: Tuple[str, str], train: bool = False)` → ndarray

Loads the training/testing time series as an multi-dimensional array.

**Parameters**

- **dataset_id** (`tuple` of `str`, `str`) – Dataset ID (collection and dataset name).
- **train** (`bool`) – Whether the training (`True`) or testing (`False`, default) should be loaded.

**Returns**
**values** – The training or testing time series as a multi-dimensional array.

**Return type**
`ndarray`

`get_dataset_path(dataset_id: Tuple[str, str], train: bool = False)` → Path

Returns the path to the training/testing time series of the dataset.

**Parameters**

- **dataset_id** (`tuple` of `str`, `str`) – Dataset ID (collection and dataset name)
- **train** (`bool`) – Whether the training (`True`) or testing (`False`, default) should be returned.

**Returns**
**dataset_path** – The path to the training or testing time series.

**Return type**
`path`

`get_detailed_metadata(dataset_id: Tuple[str, str], train: bool = False)` → *DatasetMetadata*

Computes detailed metadata about the training or testing time series of a dataset.

For most of the benchmark datasets, the detailed metadata is pre-computed and just has to be loaded from disk. For all other datasets, the time series is analyzed on the fly using `timeeval.datasets.DatasetAnalyzer` and the result is saved back to disk for later reuse. The metadata about custom datasets is not cached on disk! The following additional metadata is provided:

- Information about the training time series, if `train=True` is specified.
- Mean, variance, trend, and stationarity information for each channel of the time series individually.

**Parameters**

- **dataset_id** (`tuple` of `str`, `str`) – Dataset ID (collection and dataset name).
- **train** (`bool`) – Whether the training (`True`) or testing (`False`, default) should be loaded.

**Returns**
**metadata** – Detailed metadata about the training or testing time series.

**Return type**
`dataset metadata object`

**See also:**

**timeeval.datasets.DatasetAnalyzer**
Utility class used for the extraction of metadata.

**timeeval.datasets.DatasetMetadata**
Data class of the returned result.

**get_training_type**(*dataset_id: Tuple[str, str]*) → *TrainingType*

Returns the training type of a specific dataset.

**Parameters**
**dataset_id** (tuple of str, str) – Dataset ID (collection and dataset name)

**Returns**
**training_type** – Either unsupervised, semi-supervised, or supervised.

**Return type**
TrainingType enum

**See also:**

*timeeval.TrainingType*
Enumeration of training types that could be returned by this method.

**load_custom_datasets**(*file_path: Union[str, Path]*) → None

Reads a configuration file that contains additional datasets and adds them to the current dataset index.

You can add custom datasets to the dataset manager either using a constructor argument or using this method. The datasets from the configuration file are added to the internal dataset index and are then available for querying. The configuration file uses the JSON schema and supports this structure:

```
{
    "dataset_name": {
        "test_path": "./path/to/test.csv",
        "train_path": "./path/to/train.csv",
        "type": "synthetic",
        "period": 10
    }
}
```

The properties train_path, type, and period are optional. Dataset names must be unqiue within the configuration file. The datasets are automatically assigned to the custom dataset collection.

**Warning:** Repeated calls to this method overwrite the existing custom dataset list.

**Parameters**
**file_path** (path) – Path to the custom dataset configuration file.

**refresh**(*force: bool = False*) → None

Re-read the benchmark dataset collection information from disk.

**save**() → None

Saves the in-memory dataset index to disk.

Persists newly added benchmark datasets from memory to the benchmark dataset collection file datasets.csv. Custom datasets are excluded from persistence and cannot be saved to disk; use *add_dataset()* or *add_datasets()* to add datasets to the benchmark dataset collection.

**select**(*collection: Optional[str] = None*, *dataset: Optional[str] = None*, *dataset_type: Optional[str] = None*, *datetime_index: Optional[bool] = None*, *training_type: Optional[TrainingType] = None*, *train_is_normal: Optional[bool] = None*, *input_dimensionality: Optional[InputDimensionality] = None*, *min_anomalies: Optional[int] = None*, *max_anomalies: Optional[int] = None*, *max_contamination: Optional[float] = None*) → List[Tuple[str, str]]

Returns a list of dataset identifiers from the dataset collection whose datasets match **all** of the given conditions.

**Parameters**

- **collection** (str) – restrict datasets to a specific collection
- **dataset** (str) – restrict datasets to a specific name
- **dataset_type** (str) – restrict dataset type (e.g. *real* or *synthetic*)
- **datetime_index** (bool) – only select datasets for which a datetime index exists; if True: "timestamp"-column has datetime values; if False: "timestamp"-column has monotonically increasing integer values; this condition is ignored by custom datasets.
- **training_type** (*timeeval.TrainingType*) – select datasets for specific training needs: * *SUPERVISED*, * *SEMI_SUPERVISED*, or * *UNSUPERVISED*
- **train_is_normal** (bool) – if True: only return datasets for which the training dataset does not contain anomalies; if False: only return datasets for which the training dataset contains anomalies
- **input_dimensionality** (*timeeval.InputDimensionality*) – restrict dataset to input type, either univariate or multivariate
- **min_anomalies** (int) – restrict datasets to those with a minimum number of min_anomalies anomalous subsequences
- **max_anomalies** (int) – restrict datasets to those with a maximum number of max_anomalies anomalous subsequences
- **max_contamination** (int) – restrict datasets to those having a contamination smaller or equal to max_contamination

**Returns**

**dataset_names** – A list of dataset identifiers (tuple of collection name and dataset name).

**Return type**

List[Tuple[str,str]]

**class** timeeval.datasets.dataset_manager.**DatasetRecord**(*collection_name*, *dataset_name*, *train_path*, *test_path*, *dataset_type*, *datetime_index*, *split_at*, *train_type*, *train_is_normal*, *input_type*, *length*, *dimensions*, *contamination*, *num_anomalies*, *min_anomaly_length*, *median_anomaly_length*, *max_anomaly_length*, *mean*, *stddev*, *trend*, *stationarity*, *period_size*)

Bases: NamedTuple

**collection_name: str**

Alias for field number 0

**contamination: float**

Alias for field number 12

**count**(*value*, */*)

Return number of occurrences of value.

**dataset_name:** str

Alias for field number 1

**dataset_type:** str

Alias for field number 4

**datetime_index:** bool

Alias for field number 5

**dimensions:** int

Alias for field number 11

**index**(*value*, *start=0*, *stop=9223372036854775807*, */*)

Return first index of value.

Raises ValueError if the value is not present.

**input_type:** str

Alias for field number 9

**length:** int

Alias for field number 10

**max_anomaly_length:** int

Alias for field number 16

**mean:** float

Alias for field number 17

**median_anomaly_length:** int

Alias for field number 15

**min_anomaly_length:** int

Alias for field number 14

**num_anomalies:** int

Alias for field number 13

**period_size:** Optional[int]

Alias for field number 21

**split_at:** int

Alias for field number 6

**stationarity:** str

Alias for field number 20

**stddev:** float

Alias for field number 18

**test_path:** str

Alias for field number 3

**train_is_normal:** bool

Alias for field number 8

**train_path:** Optional[str]

Alias for field number 2

**train_type:** str

Alias for field number 7

**trend:** str

Alias for field number 19

### 3.4.7 timeeval.datasets.datasets

**class** timeeval.datasets.datasets.**Datasets**(*df: DataFrame*, *custom_datasets_file: Optional[Union[str, Path]] = None*)

Bases: ABC

Provides read-only access to benchmark datasets and their metadata.

This is an abstract class (interface). Please use *timeeval.datasets.dataset_manager.DatasetManager* or *timeeval.datasets.multi_dataset_manager.MultiDatasetManager* instead. The constructor arguments are filled in by the respective implementation.

**Parameters**

- **df** (pandas DataFrame) – Metadata of all loaded datasets.
- **custom_datasets_file** (pathlib.Path or str) – Path to a file listing additional custom datasets.

**INDEX_FILENAME:** str = 'datasets.csv'

**METADATA_FILENAME_SUFFIX:** str = 'metadata.json'

**df**() → DataFrame

Returns a copy of the internal dataset metadata collection.

The DataFrame has the following schema:

**Index:**
dataset_name, collection_name

**Columns:**
train_path, test_path, dataset_type, datetime_index, split_at, train_type, train_is_normal, input_type, length, dimensions, contamination, num_anomalies, min_anomaly_length, median_anomaly_length, max_anomaly_length, mean, stddev, trend, stationarity, period_size

**Returns**
**df** – All custom and benchmark datasets and their metadata.

**Return type**
data frame

**get**(*collection_name: Union[str, Tuple[str, str]]*, *dataset_name: Optional[str] = None*) → *Dataset*

Returns dataset metadata.

### Examples

```
>>> from timeeval.datasets import DatasetManager
>>> dm = DatasetManager("path/to/datasets")
>>> dataset_id = ("custom", "dataset1")
```

Access using the dataset ID:

```
>>> dm.get(dataset_id)
Dataset(datsetId=("custom", "dataset1"), ...)
```

Access using collection and dataset name:

```
>>> dm.get("custom", "dataset1")
Dataset(datsetId=("custom", "dataset1"), ...)
```

**Parameters**

- **collection_name** (str or tuple of str and str) – Name of the dataset collection or the dataset ID (collection, and dataset name).
- **dataset_name** (str, *optional*) – Name of the dataset or empty.

**Returns**
**dataset** – The dataset metadata

**Return type**
a dataset object

**get_collection_names**() → List[str]

Returns the unique dataset collection names (includes custom datasets if present).

**get_dataset_df**(*dataset_id: Tuple[str, str]*, *train: bool = False*) → DataFrame

Loads the training/testing time series as a data frame.

**Parameters**

- **dataset_id** (tuple of str, str) – Dataset ID (collection and dataset name).
- **train** (bool) – Whether the training (True) or testing (False, default) should be loaded.

**Returns**
**df** – The training or testing time series as a pandas.DataFrame.

**Return type**
data frame

**get_dataset_names**() → List[str]

Returns the unique dataset names (includes custom datasets if present).

**get_dataset_ndarray**(*dataset_id: Tuple[str, str]*, *train: bool = False*) → ndarray

Loads the training/testing time series as an multi-dimensional array.

**Parameters**

- **dataset_id** (tuple of str, str) – Dataset ID (collection and dataset name).
- **train** (bool) – Whether the training (True) or testing (False, default) should be loaded.

**Returns**
**values** – The training or testing time series as a multi-dimensional array.

**Return type**
ndarray

get_dataset_path(*dataset_id: Tuple[str, str]*, *train: bool = False*) → Path

Returns the path to the training/testing time series of the dataset.

**Parameters**

- **dataset_id** (tuple of str, str) – Dataset ID (collection and dataset name)
- **train** (bool) – Whether the training (True) or testing (False, default) should be returned.

**Returns**
**dataset_path** – The path to the training or testing time series.

**Return type**
path

get_detailed_metadata(*dataset_id: Tuple[str, str]*, *train: bool = False*) → *DatasetMetadata*

Computes detailed metadata about the training or testing time series of a dataset.

For most of the benchmark datasets, the detailed metadata is pre-computed and just has to be loaded from disk. For all other datasets, the time series is analyzed on the fly using timeeval.datasets.DatasetAnalyzer and the result is saved back to disk for later reuse. The metadata about custom datasets is not cached on disk! The following additional metadata is provided:

- Information about the training time series, if train=True is specified.
- Mean, variance, trend, and stationarity information for each channel of the time series individually.

**Parameters**

- **dataset_id** (tuple of str, str) – Dataset ID (collection and dataset name).
- **train** (bool) – Whether the training (True) or testing (False, default) should be loaded.

**Returns**
**metadata** – Detailed metadata about the training or testing time series.

**Return type**
dataset metadata object

**See also:**

**timeeval.datasets.DatasetAnalyzer**
Utility class used for the extraction of metadata.

**timeeval.datasets.DatasetMetadata**
Data class of the returned result.

get_training_type(*dataset_id: Tuple[str, str]*) → *TrainingType*

Returns the training type of a specific dataset.

**Parameters**
**dataset_id** (tuple of str, str) – Dataset ID (collection and dataset name)

**Returns**
**training_type** – Either unsupervised, semi-supervised, or supervised.

**Return type**
TrainingType enum

**See also:**

*timeeval.TrainingType*
Enumeration of training types that could be returned by this method.

**load_custom_datasets**(*file_path: Union[str, Path]*) → None

Reads a configuration file that contains additional datasets and adds them to the current dataset index.

You can add custom datasets to the dataset manager either using a constructor argument or using this method. The datasets from the configuration file are added to the internal dataset index and are then available for querying. The configuration file uses the JSON schema and supports this structure:

```
{
    "dataset_name": {
        "test_path": "./path/to/test.csv",
        "train_path": "./path/to/train.csv",
        "type": "synthetic",
        "period": 10
    }
}
```

The properties `train_path`, `type`, and `period` are optional. Dataset names must be unqiue within the configuration file. The datasets are automatically assigned to the `custom` dataset collection.

**Warning:** Repeated calls to this method overwrite the existing custom dataset list.

**Parameters**
**file_path** (path) – Path to the custom dataset configuration file.

**abstract refresh**(*force: bool = False*) → None

Re-read the benchmark dataset collection information from disk.

**select**(*collection: Optional[str] = None, dataset: Optional[str] = None, dataset_type: Optional[str] = None, datetime_index: Optional[bool] = None, training_type: Optional[TrainingType] = None, train_is_normal: Optional[bool] = None, input_dimensionality: Optional[InputDimensionality] = None, min_anomalies: Optional[int] = None, max_anomalies: Optional[int] = None, max_contamination: Optional[float] = None*) → List[Tuple[str, str]]

Returns a list of dataset identifiers from the dataset collection whose datasets match **all** of the given conditions.

**Parameters**

- **collection** (str) – restrict datasets to a specific collection
- **dataset** (str) – restrict datasets to a specific name
- **dataset_type** (str) – restrict dataset type (e.g. *real* or *synthetic*)
- **datetime_index** (bool) – only select datasets for which a datetime index exists; if `True`: "timestamp"-column has datetime values; if `False`: "timestamp"-column has monotonically increasing integer values; this condition is ignored by custom datasets.
- **training_type** (*timeeval.TrainingType*) – select datasets for specific training needs: * *SUPERVISED*, * *SEMI_SUPERVISED*, or * *UNSUPERVISED*
- **train_is_normal** (bool) – if `True`: only return datasets for which the training dataset does not contain anomalies; if `False`: only return datasets for which the training dataset contains anomalies

- **input_dimensionality** (*timeeval.InputDimensionality*) – restrict dataset to input type, either univariate or multivariate
- **min_anomalies** (int) – restrict datasets to those with a minimum number of min_anomalies anomalous subsequences
- **max_anomalies** (int) – restrict datasets to those with a maximum number of max_anomalies anomalous subsequences
- **max_contamination** (int) – restrict datasets to those having a contamination smaller or equal to max_contamination

**Returns**
**dataset_names** – A list of dataset identifiers (tuple of collection name and dataset name).

**Return type**
List[Tuple[str,str]]

### 3.4.8 timeeval.datasets.metadata

class timeeval.datasets.metadata.**AnomalyLength**(*min: int*, *median: int*, *max: int*)

Bases: object

**max:** int

**median:** int

**min:** int

class timeeval.datasets.metadata.**DatasetMetadata**(*dataset_id: Tuple[str, str]*, *is_train: bool*, *length: int*, *dimensions: int*, *contamination: float*, *num_anomalies: int*, *anomaly_length: AnomalyLength*, *means: Dict[str, float]*, *stddevs: Dict[str, float]*, *trends: Dict[str, List[Trend]]*, *stationarities: Dict[str, Stationarity]*)

Bases: object

Represents the metadata of a single time series of a dataset (for each channel).

**anomaly_length:** *AnomalyLength*

**property channels:** int

**contamination:** float

**dataset_id:** Tuple[str, str]

**dimensions:** int

**static from_json**(*s: str*) → *DatasetMetadata*

**get_stationarity_name**() → str

**is_train:** bool

**length:** int

**property mean:** float

**means:** Dict[str, float]

**num_anomalies:** int

**property shape:** Tuple[int, int]

**stationarities:** Dict[str, *Stationarity*]

**property stationarity:** *Stationarity*

**property stddev:** float

**stddevs:** Dict[str, float]

**to_json**(*pretty: bool = False*) → str

**property trend:** str

**trends:** Dict[str, List[*Trend*]]

**class** timeeval.datasets.metadata.**DatasetMetadataEncoder**(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, default=None*)

Bases: JSONEncoder

**default**(*o: Any*) → Any

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

**static object_hook**(*dct: Dict[str, Any]*) → Any

**class** timeeval.datasets.metadata.**Stationarity**(*value*)

Bases: Enum

An enumeration.

**DIFFERENCE_STATIONARY = 1**

**NOT_STATIONARY = 3**

**STATIONARY = 0**

**TREND_STATIONARY = 2**

**static from_name**(*s: int*) → *Stationarity*

class timeeval.datasets.metadata.**Trend**(*tpe:* timeeval.datasets.metadata.TrendType, *coef: float*, *confidence_r2: float*)

Bases: object

**coef:** float

**confidence_r2:** float

**property name:** str

**property order:** int

**tpe:** *TrendType*

class timeeval.datasets.metadata.**TrendType**(*value*)

Bases: Enum

An enumeration.

**KUBIC = 3**

**LINEAR = 1**

**QUADRATIC = 2**

**static from_order**(*order: int*) → *TrendType*

### 3.4.9 timeeval.datasets.multi_dataset_manager

class timeeval.datasets.multi_dataset_manager.**MultiDatasetManager**(*data_folders: List[Union[str, Path]]*, *custom_datasets_file: Optional[Union[str, Path]] = None*)

Bases: *Datasets*

Provides read-only access to multiple benchmark datasets collections and their meta-information.

Manages dataset collections and their meta-information that are stored in multiple folders. The entries in all index files must be unique and are NOT allowed to overlap! This would lead to information loss!

**Parameters**

- **data_folders** (list of paths) – List of data paths that hold the datasets and the index files.
- **custom_datasets_file** (path) – Path to a file listing additional custom datasets.

**Raises**

FileNotFoundError – If the *datasets.csv*-file was not found in any of the *data_folders*.

**See also:**

timeeval.datasets.Datasets, timeeval.datasets.DatasetManager

**INDEX_FILENAME: str = 'datasets.csv'**

**METADATA_FILENAME_SUFFIX: str = 'metadata.json'**

**df**() → DataFrame

Returns a copy of the internal dataset metadata collection.

The DataFrame has the following schema:

**Index:**
dataset_name, collection_name

**Columns:**
train_path, test_path, dataset_type, datetime_index, split_at, train_type, train_is_normal, input_type, length, dimensions, contamination, num_anomalies, min_anomaly_length, median_anomaly_length, max_anomaly_length, mean, stddev, trend, stationarity, period_size

**Returns**
**df** – All custom and benchmark datasets and their metadata.

**Return type**
data frame

**get**(*collection_name: Union[str, Tuple[str, str]]*, *dataset_name: Optional[str] = None*) → *Dataset*

Returns dataset metadata.

### Examples

```
>>> from timeeval.datasets import DatasetManager
>>> dm = DatasetManager("path/to/datasets")
>>> dataset_id = ("custom", "dataset1")
```

Access using the dataset ID:

```
>>> dm.get(dataset_id)
Dataset(datsetId=("custom", "dataset1"), ...)
```

Access using collection and dataset name:

```
>>> dm.get("custom", "dataset1")
Dataset(datsetId=("custom", "dataset1"), ...)
```

**Parameters**

- **collection_name** (str or tuple of str and str) – Name of the dataset collection or the dataset ID (collection, and dataset name).
- **dataset_name** (str, *optional*) – Name of the dataset or empty.

**Returns**
**dataset** – The dataset metadata

**Return type**
a dataset object

**get_collection_names**() → List[str]

Returns the unique dataset collection names (includes custom datasets if present).

**get_dataset_df**(*dataset_id: Tuple[str, str]*, *train: bool = False*) → DataFrame

Loads the training/testing time series as a data frame.

**Parameters**

- **dataset_id** (tuple of str, str) – Dataset ID (collection and dataset name).
- **train** (bool) – Whether the training (True) or testing (False, default) should be loaded.

**Returns**
**df** – The training or testing time series as a pandas.DataFrame.

**Return type**
data frame

**get_dataset_names**() → List[str]

Returns the unique dataset names (includes custom datasets if present).

**get_dataset_ndarray**(*dataset_id: Tuple[str, str]*, *train: bool = False*) → ndarray

Loads the training/testing time series as an multi-dimensional array.

**Parameters**

- **dataset_id** (tuple of str, str) – Dataset ID (collection and dataset name).
- **train** (bool) – Whether the training (True) or testing (False, default) should be loaded.

**Returns**
**values** – The training or testing time series as a multi-dimensional array.

**Return type**
ndarray

**get_dataset_path**(*dataset_id: Tuple[str, str]*, *train: bool = False*) → Path

Returns the path to the training/testing time series of the dataset.

**Parameters**

- **dataset_id** (tuple of str, str) – Dataset ID (collection and dataset name)
- **train** (bool) – Whether the training (True) or testing (False, default) should be returned.

**Returns**
**dataset_path** – The path to the training or testing time series.

**Return type**
path

**get_detailed_metadata**(*dataset_id: Tuple[str, str]*, *train: bool = False*) → *DatasetMetadata*

Computes detailed metadata about the training or testing time series of a dataset.

For most of the benchmark datasets, the detailed metadata is pre-computed and just has to be loaded from disk. For all other datasets, the time series is analyzed on the fly using timeeval.datasets.DatasetAnalyzer and the result is saved back to disk for later reuse. The metadata about custom datasets is not cached on disk! The following additional metadata is provided:

- Information about the training time series, if train=True is specified.
- Mean, variance, trend, and stationarity information for each channel of the time series individually.

**Parameters**

- **dataset_id** (tuple of str, str) – Dataset ID (collection and dataset name).
- **train** (bool) – Whether the training (True) or testing (False, default) should be loaded.

**Returns**
**metadata** – Detailed metadata about the training or testing time series.

**Return type**
`dataset metadata object`

**See also:**

`timeeval.datasets.DatasetAnalyzer`
Utility class used for the extraction of metadata.

`timeeval.datasets.DatasetMetadata`
Data class of the returned result.

`get_training_type`(*dataset_id: Tuple[str, str]*) → *TrainingType*

Returns the training type of a specific dataset.

**Parameters**
**dataset_id** (`tuple` of `str`, `str`) – Dataset ID (collection and dataset name)

**Returns**
**training_type** – Either unsupervised, semi-supervised, or supervised.

**Return type**
`TrainingType enum`

**See also:**

*timeeval.TrainingType*
Enumeration of training types that could be returned by this method.

`load_custom_datasets`(*file_path: Union[str, Path]*) → None

Reads a configuration file that contains additional datasets and adds them to the current dataset index.

You can add custom datasets to the dataset manager either using a constructor argument or using this method. The datasets from the configuration file are added to the internal dataset index and are then available for querying. The configuration file uses the JSON schema and supports this structure:

```
{
    "dataset_name": {
        "test_path": "./path/to/test.csv",
        "train_path": "./path/to/train.csv",
        "type": "synthetic",
        "period": 10
    }
}
```

The properties `train_path`, `type`, and `period` are optional. Dataset names must be unqiue within the configuration file. The datasets are automatically assigned to the `custom` dataset collection.

**Warning:** Repeated calls to this method overwrite the existing custom dataset list.

**Parameters**
**file_path** (`path`) – Path to the custom dataset configuration file.

**refresh**(*force: bool = False*) → None

Re-read the benchmark dataset collection information from disk.

**select**(*collection: Optional[str] = None*, *dataset: Optional[str] = None*, *dataset_type: Optional[str] = None*, *datetime_index: Optional[bool] = None*, *training_type: Optional[TrainingType] = None*, *train_is_normal: Optional[bool] = None*, *input_dimensionality: Optional[InputDimensionality] = None*, *min_anomalies: Optional[int] = None*, *max_anomalies: Optional[int] = None*, *max_contamination: Optional[float] = None*) → List[Tuple[str, str]]

Returns a list of dataset identifiers from the dataset collection whose datasets match **all** of the given conditions.

**Parameters**

- **collection** (str) – restrict datasets to a specific collection
- **dataset** (str) – restrict datasets to a specific name
- **dataset_type** (str) – restrict dataset type (e.g. *real* or *synthetic*)
- **datetime_index** (bool) – only select datasets for which a datetime index exists; if `True`: "timestamp"-column has datetime values; if `False`: "timestamp"-column has monotonically increasing integer values; this condition is ignored by custom datasets.
- **training_type** (*timeeval.TrainingType*) – select datasets for specific training needs: * *SUPERVISED*, * *SEMI_SUPERVISED*, or * *UNSUPERVISED*
- **train_is_normal** (bool) – if `True`: only return datasets for which the training dataset does not contain anomalies; if `False`: only return datasets for which the training dataset contains anomalies
- **input_dimensionality** (*timeeval.InputDimensionality*) – restrict dataset to input type, either univariate or multivariate
- **min_anomalies** (int) – restrict datasets to those with a minimum number of `min_anomalies` anomalous subsequences
- **max_anomalies** (int) – restrict datasets to those with a maximum number of `max_anomalies` anomalous subsequences
- **max_contamination** (int) – restrict datasets to those having a contamination smaller or equal to `max_contamination`

**Returns**

**dataset_names** – A list of dataset identifiers (tuple of collection name and dataset name).

**Return type**

`List[Tuple[str,str]]`

## 3.5 timeeval.heuristics package

timeeval.heuristics.**TimeEvalHeuristic**(*signature: str*) → *TimeEvalParameterHeuristic*

Factory function for TimeEval heuristics based on their string-representation.

This wrapper allows using the heuristics by name without the need for imports. It is primarily used in the *timeeval.heuristics.inject_heuristic_values()* function. The following heuristics are currently supported:

- *RelativeDatasetSizeHeuristic*
- *AnomalyLengthHeuristic*

- *CleanStartSequenceSizeHeuristic*
- *ParameterDependenceHeuristic*
- *PeriodSizeHeuristic*
- *EmbedDimRangeHeuristic*
- *ContaminationHeuristic*
- *DefaultFactorHeuristic*
- *DefaultExponentialFactorHeuristic*
- *DatasetIdHeuristic*

**Parameters**
**signature** (str) – String representation of the heuristic to be created. Must be of the form `<heuristic_name>(<heuristic_parameters>)`

**Returns**
**heuristic** – The created heuristic object.

**Return type**
*TimeEvalParameterHeuristic*

`timeeval.heuristics.inject_heuristic_values`(*params: T*, *algorithm:* Algorithm, *dataset_details:* Dataset, *dataset_path: Path*) → T

This function parses the supplied parameter mapping in `params` and replaces all heuristic definitions with their actual values.

The heuristics are generally evaluated in the order they are defined in the parameter mapping. However, `The order within multiple ``ParameterDependenceHeuristic`s` is not defined. If a heuristic returns `None`, the corresponding parameter is removed from the parameter mapping. Heuristics can be defined by using the following syntax as the parameter value:

```
"heuristic:<heuristic_name>(<heuristic_parameters>)"
```

Heuristics can use the following information to compute their values:

- properties of the algorithm
- properties of the dataset
- the full dataset (supplied as a path to the dataset)
- the (current) parameter mapping (later evaluated heuristics can see the changes of previous heuristics)

**Parameters**

- **params** (T) – The current parameter mapping, whose values should be updated by the heuristics. If a immutable mapping is passed, no changes will be made.
- **algorithm** (Algorithm) – The algorithm (*Algorithm*) for which the parameter mapping is valid.
- **dataset_details** (Dataset) – The dataset for which the parameter mapping is supposed to be used.
- **dataset_path** (Path) – The path to the dataset.

**Returns**
**params** – The updated parameter mapping.

**Return type**
T

### 3.5.1 timeeval.heuristics.TimeEvalParameterHeuristic

**class** timeeval.heuristics.**TimeEvalParameterHeuristic**

Bases: ABC

Base class for TimeEval parameter heuristics.

Heuristics are used to calculate parameter values for algorithms based on information about the algorithm, the dataset, or other parameters. They are evaluated in the driver process when TimeEval is configured. This means that the datasets must be available on the node executing the driver process. The calculated parameter values are then injected into the algorithm configuration and the algorithm is executed on the cluster.

**See also:**

*timeeval.heuristics.inject_heuristic_values()*
Function that uses the heuristics to calculate parameter values for algorithms.

**classmethod get_param_names**() → List[str]

Get parameter names (arguments) for the heuristic.

Adopted from https://github.com/scikit-learn/scikit-learn/blob/2beed5584/sklearn/base.py.

**property name: str**

Name of this parameter heuristic (corresponds to the class name).

**parameters**() → Dict[str, Any]

Get the heuristic's parameters (arguments to the heuristic) as a dictionary.

### 3.5.2 timeeval.heuristics.AnomalyLengthHeuristic

**class** timeeval.heuristics.**AnomalyLengthHeuristic**(*agg_type: str = 'median'*)

Bases: *TimeEvalParameterHeuristic*

Heuristic to use the anomaly length of the dataset as parameter value. **Uses ground-truth labels,** and should therefore only be used for testing purposes.

#### Examples

```
>>> from timeeval.params import FixedParameters
>>> params = FixedParameters({"window_size": "heuristic:AnomalyLengthHeuristic(agg_
↪type='max')"})
```

**Parameters**
**agg_type** (str) – Type of aggregation to use for calculating the anomaly length when multiple anomalies are present in the time series. Must be one of min, median, or max. (default: median)

### 3.5.3 timeeval.heuristics.CleanStartSequenceSizeHeuristic

**class** timeeval.heuristics.**CleanStartSequenceSizeHeuristic**(*max_factor: float = 0.1*)

Bases: *TimeEvalParameterHeuristic*

Heuristic to compute the number of time steps until the first anomaly occurs, and use it as parameter value. **Uses ground-truth labels.** Allows to specify a maximum fraction of the entire time series length. The minimum of the computed value and the maximum fraction is used as parameter value.

#### Examples

```
>>> from timeeval.params import FixedParameters
>>> params = FixedParameters({"n_init":
↪"heuristic:CleanStartSequenceSizeHeuristic(max_factor=0.1)"})
```

**Parameters**

**max_factor** (float) – Maximum fraction of the entire time series length to use as parameter value. This limits the parameter value. (default: 0.1)

### 3.5.4 timeeval.heuristics.ContaminationHeuristic

**class** timeeval.heuristics.**ContaminationHeuristic**

Bases: *TimeEvalParameterHeuristic*

Heuristic to use the time series' contamination as parameter value. The contamination is defined as the fraction of anomalous points to all points in the time series.

#### Examples

```
>>> from timeeval.params import FixedParameters
>>> params = FixedParameters({"fraction": "heuristic:ContaminationHeuristic()"})
```

### 3.5.5 timeeval.heuristics.DatasetIdHeuristic

**class** timeeval.heuristics.**DatasetIdHeuristic**

Bases: *TimeEvalParameterHeuristic*

Heuristic to pass the dataset ID as a parameter value.

The dataset ID is a tuple of the collection name and the dataset name, such as `("KDD-TSAD", "022_UCR_Anomaly_DISTORTEDGP711MarkerLFM5z4")`.

#### Examples

```
>>> from timeeval.params import FixedParameters
>>> params = FixedParameters({"dataset_id": "heuristic:DatasetIdHeuristic()"})
```

### 3.5.6 timeeval.heuristics.DefaultExponentialFactorHeuristic

**class** timeeval.heuristics.**DefaultExponentialFactorHeuristic**(*exponent: int = 0*, *zero_fb: float = 1.0*)

Bases: *TimeEvalParameterHeuristic*

Heuristic to use the default value multiplied by a factor of $10^{exponent}$ as parameter value.

This allows easier specification of exponential parameter search spaces based on the default value. E.g. if we consider a learning rate parameter with default value 0.01, we can use this heuristic to specify a search space of [0.0001, 0.001, 0.01, 0.1, 1] by using the following parameter values:

- `"heuristic:DefaultExponentialFactorHeuristic(exponent=-2)"`
- `"heuristic:DefaultExponentialFactorHeuristic(exponent=-1)"`
- `"heuristic:DefaultExponentialFactorHeuristic()"`
- `"heuristic:DefaultExponentialFactorHeuristic(exponent=1)"`
- `"heuristic:DefaultExponentialFactorHeuristic(exponent=2)"`

But if the default parameter value is 0.5, the search space would be [0.005, 0.05, 0.5, 5, 50].

#### Examples

```
>>> from timeeval.params import FixedParameters
>>> params = FixedParameters({"window_size":
↪"heuristic:DefaultExponentialFactorHeuristic(exponent=1, zero_fb=200)"})
```

**Parameters**

- **exponent** (int) – Exponent to use for the factor. (default: 0)
- **zero_fb** (float) – Value to use for the default value if it is 0. (default: 1.0)

### 3.5.7 timeeval.heuristics.DefaultFactorHeuristic

**class** timeeval.heuristics.**DefaultFactorHeuristic**(*factor: float = 1.0*, *zero_fb: float = 1.0*)

Bases: *TimeEvalParameterHeuristic*

Heuristic to use the default value multiplied by a factor as parameter value.

This allows easier specification of parameter search spaces based on the default value. E.g. if we consider a n_clusters parameter with default value 50, we can use this heuristic to specify a search space of [10, 25, 50, 75, 100] by using the following parameter values:

- `"heuristic:DefaultExponentialFactorHeuristic(factor=0.2)"`
- `"heuristic:DefaultExponentialFactorHeuristic(factor=0.5)"`
- `"heuristic:DefaultExponentialFactorHeuristic()"`

- "heuristic:DefaultExponentialFactorHeuristic(factor=1.5)"
- "heuristic:DefaultExponentialFactorHeuristic(factor=2.0)"

But if the default parameter value is 100, the search space would be [20, 50, 100, 150, 200].

#### Examples

```
>>> from timeeval.params import FixedParameters
>>> params = FixedParameters({"window_size":
↪"heuristic:DefaultFactorHeuristic(factor=1, zero_fb=200)"})
```

**Parameters**

- **factor** (float) – Factor to use for the default value. (default: 1.0)
- **zero_fb** (float) – Value to use for the default value if it is 0. (default: 1.0)

### 3.5.8 timeeval.heuristics.EmbedDimRangeHeuristic

**class** timeeval.heuristics.**EmbedDimRangeHeuristic**(*base_factor: float = 1.0*, *base_fb_value: int = 50*, *dim_factors: Optional[List[float]] = None*)

Bases: *TimeEvalParameterHeuristic*

Heuristic to use a range of embedding dimensions as parameter value.

The base dimensionality is calculated based on the *PeriodSizeHeuristic*, base factor, and base fallback value. The base dimensionality is then multiplied by the factors specified in `dim_factors` to create the embedding dimension range.

#### Examples

```
>>> from timeeval.params import FixedParameters
>>> params = FixedParameters({
...     "embed_dim": "heuristic:EmbedDimRangeHeuristic(base_factor=1, base_fb_
↪value=50, dim_factors=[0.5, 1.0, 1.5])"
... })
```

**Parameters**

- **base_factor** (float) – Factor to use for the base dimensionality. Directly passed on to the PeriodSizeHeuristic. (default: 1.0)
- **base_fb_value** (int) – Fallback value to use for the base dimensionality. Directly passed on to the PeriodSizeHeuristic. (default: 50)
- **dim_factors** (List[float]) – Factors to use for the creation of the embedding dimension range. (default: [0.5, 1.0, 1.5])

### 3.5.9 timeeval.heuristics.ParameterDependenceHeuristic

class timeeval.heuristics.**ParameterDependenceHeuristic**(*source_parameter: str*, *fn: Optional[Callable[[Any], Any]] = None*, *factor: Optional[float] = None*)

Bases: *TimeEvalParameterHeuristic*

Heuristic to use the value of another parameter as parameter value.

`ParameterDependenceHeuristic` can be used to create a parameter value that depends on another parameter. This can be done by either supplying a mapping function or a factor. If a mapping function is supplied, it is called with the value of the source parameter as the only argument. If a factor is supplied, the value of the source parameter is multiplied by the factor. **You cannot supply both a mapping function and a factor!** This heuristic is evaluated after all other heuristics, so you can use it to create a parameter value that depends on the values of other parameters filled by heuristics.

#### Examples

```
>>> from timeeval.params import FixedParameters
>>> params = FixedParameters({
...     "latent_dim": "heuristic:ParameterDependenceHeuristic(source_parameter=
↪'window_size', factor=0.5)"
... })
```

```
>>> from timeeval.params import FixedParameters
>>> params = FixedParameters({
...     "latent_dims": "heuristic:ParameterDependenceHeuristic(source_parameter=
↪'window_size', fn=lambda x: [x // 2, x, x * 2])"
... })
```

**Parameters**

- **source_parameter** (`str`) – Name of the parameter to use as source.
- **fn** (`Callable[[Any], Any]`, *optional*) – Mapping function to use on the source parameter value. (default: None)
- **factor** (`float`, *optional*) – Factor to multiply the source parameter value with. (default: None)

### 3.5.10 timeeval.heuristics.PeriodSizeHeuristic

class timeeval.heuristics.**PeriodSizeHeuristic**(*factor: float = 1.0*, *fb_anomaly_length_agg_type: Optional[str] = None*, *fb_value: int = 1*)

Bases: *TimeEvalParameterHeuristic*

Heuristic to use the period size of the dataset as parameter value.

Not all datasets have a period size, so this heuristic uses the following fallbacks in order:

1. If `fb_anomaly_length_agg_type` is specified, the *AnomalyLengthHeuristic* with the specified aggregation type is used as fallback. 2. If `fb_value` is specified, it is directly used as fallback.

#### Examples

```
>>> from timeeval.params import FixedParameters
>>> params = FixedParameters({
...     "window_size": "heuristic:PeriodSizeHeuristic(factor=1.0, fb_anomaly_length_
↪agg_type='median', fb_value=100)"
... })
```

**Parameters**

- **factor** (float) – Factor to use for the period size. (default: 1.0)
- **fb_anomaly_length_agg_type** (str, *optional*) – Aggregation type to use for the *AnomalyLengthHeuristic* fallback. (default: None)
- **fb_value** (int, *optional*) – Value to use as fallback if no period size is available. (default: 1)

### 3.5.11 timeeval.heuristics.RelativeDatasetSizeHeuristic

**class** timeeval.heuristics.**RelativeDatasetSizeHeuristic**(*factor: float = 0.1*)

Bases: *TimeEvalParameterHeuristic*

Heuristic to set a parameter value depending on the size of the dataset (length of the time series).

#### Examples

```
>>> from timeeval.params import FixedParameters
>>> params = FixedParameters({"n_init":
↪"heuristic:RelativeDatasetSizeHeuristic(factor=0.1)"})
```

**Parameters**

**factor** (float) – Factor to multiply the dataset length with to get the parameter value. (default: 0.1)

## 3.6 timeeval.metrics package

This module contains all metrics that can be used with TimeEval. The metrics are divided into five different categories:

- **Classification-metrics:** These metrics are defined over binary classification predictions (zeros or ones), thus they require a thresholding strategy to convert anomaly scorings to binary classification results.
  - *Precision*
  - *Recall*
  - *F1Score*
- **AUC-metrics:** All AUC-Metrics support continuous scorings, and calculate the area under a custom curve function.
  - *RocAUC*
  - *PrAUC*

- **Range-metrics:** Range-metrics compute the quality scores for anomaly ranges (windows) instead of each point in the time series.
  - *RangePrecision*
  - *RangeRecall*
  - *RangeFScore*
  - *RangePrecisionRangeRecallAUC*
- **VUS-metrics:** The metrics of this category share a custom definition of range-based recall and range-based precision [PaparrizosEtAl2022].
  - *RangePrAUC*
  - *RangeRocAUC*
  - *RangePrVUS*
  - *RangeRocVUS*
- **Other-metrics:** Metrics that don't belong to any of the above categories:
  - *AveragePrecision*
  - *PrecisionAtK*
  - *FScoreAtK*

All metrics inherit from the abstract base class *Metric*, and implement the `__call__` method, the `supports_continuous_scorings` method, and the `name` property. This allows them to be used within TimeEval and on their own. You can also implement your own metrics by inheriting from *timeeval.metrics.Metric* (see its documentation for more information).

### Examples

Using the default metric list that just contains ROC_AUC:

```
>>> from timeeval import TimeEval, DefaultMetrics
>>> TimeEval(dataset_mgr=..., datasets=[], algorithms=[],
>>>          metrics=DefaultMetrics.default_list())
```

Using a custom selection of metrics:

```
>>> from timeeval import TimeEval
>>> from timeeval.metrics import RangeRocAUC, RangeRocVUS, RangePrAUC, RangePrVUS
>>> TimeEval(dataset_mgr=..., datasets=[], algorithms=[],
>>>          metrics=[RangeRocAUC(buffer_size=100), RangeRocVUS(max_buffer_size=100),
>>>                   RangePrAUC(buffer_size=100), RangePrVUS(max_buffer_size=100)])
```

Using the metrics without TimeEval:

```
>>> import numpy as np
>>> from timeeval import DefaultMetrics
>>> from timeeval.metrics import RangePrAUC
>>> from timeeval.metrics.thresholding import PercentileThresholding
>>> rng = np.random.default_rng(42)
>>> y_true = rng.random(100) > 0.5
>>> y_score = rng.random(100)
```

(continues on next page)

(continued from previous page)

```
>>> metrics = [
>>>     # default metrics are already parameterized objects:
>>>     DefaultMetrics.ROC_AUC,
>>>     # all metrics (in general) are classes that need to be instantiated with their␣
↪parameterization:
>>>     RangePrAUC(buffer_size=100),
>>>     # classification metrics need a thresholding strategy for continuous scorings:
>>>     F1Score(PercentileThresholding(percentile=95))
>>> ]
>>> # compute the metrics
>>> for m in metrics:
>>>     metric_value = m(y_true, y_score)
>>>     print(f"{m.name} = {metric_value}")
```

### 3.6.1 timeeval.metrics.Metric

**class** timeeval.metrics.**Metric**

Bases: ABC

Base class for metric implementations that score anomaly scorings against ground truth binary labels. Every subclass must implement *name()*, *score()*, and *supports_continuous_scorings()*.

#### Examples

You can implement a new TimeEval metric easily by inheriting from this base class. A simple metric, for example, uses a fixed threshold to get binary labels and computes the false positive rate:

```
>>> from timeeval.metrics import Metric
>>> class FPR(Metric):
>>>     def __init__(self, threshold: float = 0.8):
>>>         self._threshold = threshold
>>>     @property
>>>     def name(self) -> str:
>>>         return f"FPR@{self._threshold}"
>>>     def score(self, y_true: np.ndarray, y_score: np.ndarray) -> float:
>>>         y_pred = y_score >= self._threshold
>>>         fp = np.sum(y_pred & ~y_true)
>>>         return fp / (fp + np.sum(y_true))
>>>     def supports_continuous_scorings(self) -> bool:
>>>         return True
```

This metric can then be used in TimeEval:

```
>>> from timeeval import TimeEval
>>> from timeeval.metrics import DefaultMetrics
>>> timeeval = TimeEval(dmgr=..., datasets=[], algorithms=[],
>>>                     metrics=[FPR(threshold=0.8), DefaultMetrics.ROC_AUC])
```

**abstract property name:** str

Returns the unique name of this metric.

**abstract score**(*y_true: ndarray*, *y_score: ndarray*) → float

Implementation of the metric's scoring function.

Please use `__call__()` instead of calling this function directly!

#### Examples

Instantiate a metric and call it using the `__call__` method:

```
>>> import numpy as np
>>> from timeeval.metrics import RocAUC
>>> metric = RocAUC(plot=False)
>>> metric(np.array([0, 1, 1, 0]), np.array([0.1, 0.4, 0.35, 0.8]))
0.5
```

**abstract supports_continuous_scorings**() → bool

Whether this metric accepts continuous anomaly scorings as input (`True`) or binary classification labels (`False`).

### 3.6.2 timeeval.metrics.RocAUC

**class** timeeval.metrics.**RocAUC**(*plot: bool = False*, *plot_store: bool = False*)

Bases: `AucMetric`

Computes the area under the receiver operating characteristic curve.

**Parameters**

- **plot** (bool) – Set this parameter to `True` to plot the curve.
- **plot_store** (bool) – If this parameter is `True` the curve plot will be saved in the current working directory under the name template "fig-{metric-name}.pdf".

**See also:**

https://en.wikipedia.org/wiki/Receiver_operating_characteristic : Explanation of the ROC-curve.

**property name: str**

Returns the unique name of this metric.

**score**(*y_true: ndarray*, *y_score: ndarray*) → float

Implementation of the metric's scoring function.

Please use `__call__()` instead of calling this function directly!

#### Examples

Instantiate a metric and call it using the `__call__` method:

```
>>> import numpy as np
>>> from timeeval.metrics import RocAUC
>>> metric = RocAUC(plot=False)
>>> metric(np.array([0, 1, 1, 0]), np.array([0.1, 0.4, 0.35, 0.8]))
0.5
```

**supports_continuous_scorings**() → bool

Whether this metric accepts continuous anomaly scorings as input (True) or binary classification labels (False).

### 3.6.3 timeeval.metrics.PrAUC

**class** timeeval.metrics.**PrAUC**(*plot: bool = False*, *plot_store: bool = False*)

Bases: AucMetric

Computes the area under the precision recall curve.

**Parameters**

- **plot** (bool) – Set this parameter to True to plot the curve.
- **plot_store** (bool) – If this parameter is True the curve plot will be saved in the current working directory under the name template "fig-{metric-name}.pdf".

**property name:** str

Returns the unique name of this metric.

**score**(*y_true: ndarray*, *y_score: ndarray*) → float

Implementation of the metric's scoring function.

Please use __call__() instead of calling this function directly!

#### Examples

Instantiate a metric and call it using the __call__ method:

```
>>> import numpy as np
>>> from timeeval.metrics import RocAUC
>>> metric = RocAUC(plot=False)
>>> metric(np.array([0, 1, 1, 0]), np.array([0.1, 0.4, 0.35, 0.8]))
0.5
```

**supports_continuous_scorings**() → bool

Whether this metric accepts continuous anomaly scorings as input (True) or binary classification labels (False).

### 3.6.4 timeeval.metrics.RangePrecisionRangeRecallAUC

**class** timeeval.metrics.**RangePrecisionRangeRecallAUC**(*max_samples: int = 50*, *r_alpha: float = 0.5*, *p_alpha: float = 0*, *cardinality: str = 'reciprocal'*, *bias: str = 'flat'*, *plot: bool = False*, *plot_store: bool = False*, *name: str = 'RANGE_PR_AUC'*)

Bases: AucMetric

Computes the area under the precision recall curve when using the range-based precision and range-based recall metric introduced by Tatbul et al. at NeurIPS 2018 [TatbulEtAl2018].

**Parameters**

- **max_samples** (int) – TimeEval uses a community implementation of the range-based precision and recall metrics, which is quite slow. To prevent long runtimes caused by scorings with high precision (many thresholds), just a specific amount of possible thresholds is sampled. This parameter controls the maximum number of thresholds; too low numbers degrade the metrics' quality.
- **r_alpha** (float) – Weight of the existence reward for the range-based recall.
- **p_alpha** (float) – Weight of the existence reward for the range-based precision. For most - when not all - cases, *p_alpha* should be set to 0.
- **cardinality** ({'reciprocal', 'one', 'udf_gamma'}) – Cardinality type.
- **bias** ({'flat', 'front', 'middle', 'back'}) – Positional bias type.
- **plot** (bool) –
- **plot_store** (bool) –
- **name** (str) – Custom name for this metric (e.g. including your parameter changes).

#### References

**property name:** str

Returns the unique name of this metric.

**score**(*y_true: ndarray*, *y_score: ndarray*) → float

Implementation of the metric's scoring function.

Please use `__call__()` instead of calling this function directly!

#### Examples

Instantiate a metric and call it using the `__call__` method:

```
>>> import numpy as np
>>> from timeeval.metrics import RocAUC
>>> metric = RocAUC(plot=False)
>>> metric(np.array([0, 1, 1, 0]), np.array([0.1, 0.4, 0.35, 0.8]))
0.5
```

**supports_continuous_scorings**() → bool

Whether this metric accepts continuous anomaly scorings as input (`True`) or binary classification labels (`False`).

### 3.6.5 timeeval.metrics.AveragePrecision

**class** timeeval.metrics.**AveragePrecision**(***kwargs*)

Bases: *Metric*

Computes the average precision metric aver all possible thresholds.

This metric is an approximation of the *timeeval.metrics.PrAUC*-metric.

**Parameters**

**kwargs** (dict) – Keyword arguments that get passed down to sklearn.metrics.average_precision_score()

**See also:**

`sklearn.metrics.average_precision_score`
Implementation of the average precision metric.

**property name:** `str`

Returns the unique name of this metric.

`score`(*y_true: ndarray*, *y_score: ndarray*) → float

Implementation of the metric's scoring function.

Please use `__call__()` instead of calling this function directly!

#### Examples

Instantiate a metric and call it using the `__call__` method:

```
>>> import numpy as np
>>> from timeeval.metrics import RocAUC
>>> metric = RocAUC(plot=False)
>>> metric(np.array([0, 1, 1, 0]), np.array([0.1, 0.4, 0.35, 0.8]))
0.5
```

`supports_continuous_scorings`() → bool

Whether this metric accepts continuous anomaly scorings as input (`True`) or binary classification labels (`False`).

### 3.6.6 timeeval.metrics.Precision

`class timeeval.metrics.`**`Precision`**(*thresholding_strategy: ThresholdingStrategy*)

Bases: `ClassificationMetric`

Computes the precision metric.

**Parameters**
**thresholding_strategy** (`ThresholdingStrategy`) – Thresholding strategy used to transform the anomaly scorings to binary classification predictions.

**See also:**

`sklearn.metrics.precision_score`
Implementation of the precision metric.

**property name:** `str`

Returns the unique name of this metric.

`score`(*y_true: ndarray*, *y_score: ndarray*) → float

Implementation of the metric's scoring function.

Please use `__call__()` instead of calling this function directly!

#### Examples

Instantiate a metric and call it using the `__call__` method:

```
>>> import numpy as np
>>> from timeeval.metrics import RocAUC
>>> metric = RocAUC(plot=False)
>>> metric(np.array([0, 1, 1, 0]), np.array([0.1, 0.4, 0.35, 0.8]))
0.5
```

**supports_continuous_scorings**() → bool

Whether this metric accepts continuous anomaly scorings as input (`True`) or binary classification labels (`False`).

### 3.6.7 timeeval.metrics.Recall

**class** timeeval.metrics.**Recall**(*thresholding_strategy:* ThresholdingStrategy)

Bases: `ClassificationMetric`

Computes the recall metric.

**Parameters**

**thresholding_strategy** (`ThresholdingStrategy`) – Thresholding strategy used to transform the anomaly scorings to binary classification predictions.

**See also:**

`sklearn.metrics.recall_score`
Implementation of the recall metric.

**property name:** str

Returns the unique name of this metric.

**score**(*y_true: ndarray*, *y_score: ndarray*) → float

Implementation of the metric's scoring function.

Please use `__call__()` instead of calling this function directly!

#### Examples

Instantiate a metric and call it using the `__call__` method:

```
>>> import numpy as np
>>> from timeeval.metrics import RocAUC
>>> metric = RocAUC(plot=False)
>>> metric(np.array([0, 1, 1, 0]), np.array([0.1, 0.4, 0.35, 0.8]))
0.5
```

**supports_continuous_scorings**() → bool

Whether this metric accepts continuous anomaly scorings as input (`True`) or binary classification labels (`False`).

### 3.6.8 timeeval.metrics.F1Score

**class** timeeval.metrics.**F1Score**(*thresholding_strategy:* ThresholdingStrategy)

Bases: ClassificationMetric

Computes the F1 metric, which is the harmonic mean of precision and recall.

**Parameters**

**thresholding_strategy** (ThresholdingStrategy) – Thresholding strategy used to transform the anomaly scorings to binary classification predictions.

**See also:**

sklearn.metrics.f1_score
Implementation of the F1 metric.

**property name:** str

Returns the unique name of this metric.

**score**(*y_true: ndarray*, *y_score: ndarray*) → float

Implementation of the metric's scoring function.

Please use __call__() instead of calling this function directly!

#### Examples

Instantiate a metric and call it using the __call__ method:

```
>>> import numpy as np
>>> from timeeval.metrics import RocAUC
>>> metric = RocAUC(plot=False)
>>> metric(np.array([0, 1, 1, 0]), np.array([0.1, 0.4, 0.35, 0.8]))
0.5
```

**supports_continuous_scorings**() → bool

Whether this metric accepts continuous anomaly scorings as input (True) or binary classification labels (False).

### 3.6.9 timeeval.metrics.RangePrecision

**class** timeeval.metrics.**RangePrecision**(*thresholding_strategy:* ThresholdingStrategy = *NoThresholding()*, *alpha: float = 0*, *cardinality: str = 'reciprocal'*, *bias: str = 'flat'*, *name: str = 'RANGE_PRECISION'*)

Bases: *Metric*

Computes the range-based precision metric introduced by Tatbul et al. at NeurIPS 2018 [TatbulEtAl2018].

Range precision is the average precision of each predicted anomaly range. For each predicted continuous anomaly range the overlap size, position, and cardinality is considered.

**Parameters**

- **thresholding_strategy** (ThresholdingStrategy) – Strategy used to find a threshold over continuous anomaly scores to get binary labels. Use *timeeval.metrics.thresholding.NoThresholding* for results that already contain binary labels.

- **alpha** (float) – Weight of the existence reward. Because precision by definition emphasizes on prediction quality, there is no need for an existence reward and this value should always be set to 0.
- **cardinality** ({'reciprocal', 'one', 'udf_gamma'}) – Cardinality type.
- **bias** ({'flat', 'front', 'middle', 'back'}) – Positional bias type.
- **name** (str) – Custom name for this metric (e.g. including your parameter changes).

**property name: str**

Returns the unique name of this metric.

**score**(*y_true: ndarray*, *y_score: ndarray*) → float

Implementation of the metric's scoring function.

Please use `__call__()` instead of calling this function directly!

#### Examples

Instantiate a metric and call it using the `__call__` method:

```
>>> import numpy as np
>>> from timeeval.metrics import RocAUC
>>> metric = RocAUC(plot=False)
>>> metric(np.array([0, 1, 1, 0]), np.array([0.1, 0.4, 0.35, 0.8]))
0.5
```

**supports_continuous_scorings**() → bool

Whether this metric accepts continuous anomaly scorings as input (`True`) or binary classification labels (`False`).

### 3.6.10 timeeval.metrics.RangeRecall

**class** timeeval.metrics.**RangeRecall**(*thresholding_strategy: ThresholdingStrategy = NoThresholding()*, *alpha: float = 0*, *cardinality: str = 'reciprocal'*, *bias: str = 'flat'*, *name: str = 'RANGE_RECALL'*)

Bases: *Metric*

Computes the range-based recall metric introduced by Tatbul et al. at NeurIPS 2018 [TatbulEtAl2018].

Range recall is the average recall of each real anomaly range. For each real anomaly range the overlap size, position, and cardinality with predicted anomaly ranges are considered. In addition, an existence reward can be given that boosts the recall even if just a single point of the real anomaly is in the predicted ranges.

**Parameters**

- **thresholding_strategy** (ThresholdingStrategy) – Strategy used to find a threshold over continuous anomaly scores to get binary labels. Use *timeeval.metrics.thresholding.NoThresholding* for results that already contain binary labels.
- **alpha** (float) – Weight of the existence reward. If 0: no existence reward, if 1: only existence reward. The existence reward is given if the real anomaly range has overlap with even a single point of the predicted anomaly range.
- **cardinality** ({'reciprocal', 'one', 'udf_gamma'}) – Cardinality type.
- **bias** ({'flat', 'front', 'middle', 'back'}) – Positional bias type.

- **name** (str) – Custom name for this metric (e.g. including your parameter changes).

**property name:** str

Returns the unique name of this metric.

**score**(*y_true: ndarray*, *y_score: ndarray*) → float

Implementation of the metric's scoring function.

Please use `__call__()` instead of calling this function directly!

#### Examples

Instantiate a metric and call it using the `__call__` method:

```
>>> import numpy as np
>>> from timeeval.metrics import RocAUC
>>> metric = RocAUC(plot=False)
>>> metric(np.array([0, 1, 1, 0]), np.array([0.1, 0.4, 0.35, 0.8]))
0.5
```

**supports_continuous_scorings**() → bool

Whether this metric accepts continuous anomaly scorings as input (`True`) or binary classification labels (`False`).

### 3.6.11 timeeval.metrics.RangeFScore

**class** timeeval.metrics.**RangeFScore**(*thresholding_strategy: ThresholdingStrategy = NoThresholding()*, *beta: float = 1*, *p_alpha: float = 0*, *r_alpha: float = 0.5*, *cardinality: str = 'reciprocal'*, *p_bias: str = 'flat'*, *r_bias: str = 'flat'*, *name: Optional[str] = None*)

Bases: *Metric*

Computes the range-based F-score using the recall and precision metrics by Tatbul et al. at NeurIPS 2018 [TatbulEtAl2018].

The F-beta score is the weighted harmonic mean of precision and recall, reaching its optimal value at 1 and its worst value at 0. This implementation uses the range-based precision and range-based recall as basis.

**Parameters**

- **thresholding_strategy** (ThresholdingStrategy) – Strategy used to find a threshold over continuous anomaly scores to get binary labels. Use *timeeval.metrics.thresholding.NoThresholding* for results that already contain binary labels.
- **beta** (float) – F-score beta determines the weight of recall in the combined score. beta < 1 lends more weight to precision, while beta > 1 favors recall.
- **p_alpha** (float) – Weight of the existence reward for the range-based precision. For most - when not all - cases, *p_alpha* should be set to 0.
- **r_alpha** (float) – Weight of the existence reward. If 0: no existence reward, if 1: only existence reward.
- **cardinality** ({'reciprocal', 'one', 'udf_gamma'}) – Cardinality type.
- **p_bias** ({'flat', 'front', 'middle', 'back'}) – Positional bias type.
- **r_bias** ({'flat', 'front', 'middle', 'back'}) – Positional bias type.

- **name** (str) – Custom name for this metric (e.g. including your parameter changes). If *None*, will include the beta-value in the name: "RANGE_F{beta}_SCORE".

**property name:** str

Returns the unique name of this metric.

**score**(*y_true: ndarray*, *y_score: ndarray*) → float

Implementation of the metric's scoring function.

Please use `__call__()` instead of calling this function directly!

#### Examples

Instantiate a metric and call it using the `__call__` method:

```
>>> import numpy as np
>>> from timeeval.metrics import RocAUC
>>> metric = RocAUC(plot=False)
>>> metric(np.array([0, 1, 1, 0]), np.array([0.1, 0.4, 0.35, 0.8]))
0.5
```

**supports_continuous_scorings**() → bool

Whether this metric accepts continuous anomaly scorings as input (`True`) or binary classification labels (`False`).

### 3.6.12 timeeval.metrics.FScoreAtK

**class** timeeval.metrics.**FScoreAtK**(*k: Optional[int] = None*)

Bases: *Metric*

Computes the F-score at k based on anomaly ranges.

This metric only considers the top-k predicted anomaly ranges within the scoring by finding a threshold on the scoring that produces at least k anomaly ranges. If *k* is not specified, the number of anomalies within the ground truth is used as *k*.

**Parameters**

**k** (int (optional)) – Number of top anomalies used to calculate precision. If *k* is not specified (*None*) the number of true anomalies (based on the ground truth values) is used.

**See also:**

*timeeval.metrics.thresholding.TopKRangesThresholding*
Thresholding approach used.

**property name:** str

Returns the unique name of this metric.

**score**(*y_true: ndarray*, *y_score: ndarray*) → float

Implementation of the metric's scoring function.

Please use `__call__()` instead of calling this function directly!

#### Examples

Instantiate a metric and call it using the `__call__` method:

```
>>> import numpy as np
>>> from timeeval.metrics import RocAUC
>>> metric = RocAUC(plot=False)
>>> metric(np.array([0, 1, 1, 0]), np.array([0.1, 0.4, 0.35, 0.8]))
0.5
```

**supports_continuous_scorings**() → bool

Whether this metric accepts continuous anomaly scorings as input (`True`) or binary classification labels (`False`).

### 3.6.13 timeeval.metrics.PrecisionAtK

**class** timeeval.metrics.**PrecisionAtK**(*k: Optional[int] = None*)

Bases: *Metric*

Computes the Precision at k based on anomaly ranges.

This metric only considers the top-k predicted anomaly ranges within the scoring by finding a threshold on the scoring that produces at least k anomaly ranges. If *k* is not specified, the number of anomalies within the ground truth is used as *k*.

**Parameters**

**k** (int (optional)) – Number of top anomalies used to calculate precision. If *k* is not specified (*None*) the number of true anomalies (based on the ground truth values) is used.

**See also:**

*timeeval.metrics.thresholding.TopKRangesThresholding*
Thresholding approach used.

**property name:** str

Returns the unique name of this metric.

**score**(*y_true: ndarray*, *y_score: ndarray*) → float

Implementation of the metric's scoring function.

Please use `__call__()` instead of calling this function directly!

#### Examples

Instantiate a metric and call it using the `__call__` method:

```
>>> import numpy as np
>>> from timeeval.metrics import RocAUC
>>> metric = RocAUC(plot=False)
>>> metric(np.array([0, 1, 1, 0]), np.array([0.1, 0.4, 0.35, 0.8]))
0.5
```

**supports_continuous_scorings**() → bool

Whether this metric accepts continuous anomaly scorings as input (`True`) or binary classification labels (`False`).

### 3.6.14 timeeval.metrics.RangePrAUC

**class** timeeval.metrics.**RangePrAUC**(*buffer_size: Optional[int] = None*, *compatibility_mode: bool = False*, *max_samples: int = 250*, *plot: bool = False*, *plot_store: bool = False*)

Bases: RangeAucMetric

Computes the area under the precision-recall-curve using the range-based precision and range-based recall definition from Paparrizos et al. published at VLDB 2022 [PaparrizosEtAl2022].

We first extend the anomaly labels by two slopes of `buffer_size//2` length on both sides of each anomaly, uniformly sample thresholds from the anomaly score, and then compute the confusion matrix for all thresholds. Using the resulting precision and recall values, we can plot a curve and compute its area.

We make some changes to the original implementation from [PaparrizosEtAl2022] because we do not agree with the original assumptions. To reproduce the original results, you can set the parameter `compatibility_mode=True`. This will compute exactly the same values as the code by the authors of the paper.

The following things are different in TimeEval compared to the original version:

- For the recall (FPR) existence reward, we count anomalies as separate events, even if the added slopes overlap.
- Overlapping slopes don't sum up in their anomaly weight, but we just take to maximum anomaly weight for each point in the ground truth.
- The original slopes are asymmetric: The slopes at the end of anomalies are a single point shorter than the ones at the beginning of anomalies. We use symmetric slopes of the same size for the beginning and end of anomalies.
- We use a linear approximation of the slopes instead of the convex slope shape presented in the paper.

**Parameters**

- **buffer_size** (Optional[int]) – Size of the buffer region around an anomaly. We add an increasing slope of size `buffer_size//2` to the beginning of anomalies and a decreasing slope of size `buffer_size//2` to the end of anomalies. Per default (when `buffer_size==None`), `buffer_size` is the median length of the anomalies within the time series. However, you can also set it to the period size of the dominant frequency or any other desired value.
- **compatibility_mode** (bool) – When set to `True`, produces exactly the same output as the metric implementation by the original authors. Otherwise, TimeEval uses a slightly improved implementation that fixes some bugs and uses linear slopes.
- **max_samples** (int) – Calculating precision and recall for many thresholds is quite slow. We, therefore, uniformly sample thresholds from the available score space. This parameter controls the maximum number of thresholds; too low numbers degrade the metrics' quality.
- **plot** (bool) –
- **plot_store** (bool) –

**anomaly_bounds**(*y_true: ndarray*) → Tuple[ndarray, ndarray]

corresponds to range_convers_new

**property name: str**

Returns the unique name of this metric.

**score**(*y_true: ndarray*, *y_score: ndarray*) → float

Implementation of the metric's scoring function.

Please use `__call__()` instead of calling this function directly!

#### Examples

Instantiate a metric and call it using the `__call__` method:

```
>>> import numpy as np
>>> from timeeval.metrics import RocAUC
>>> metric = RocAUC(plot=False)
>>> metric(np.array([0, 1, 1, 0]), np.array([0.1, 0.4, 0.35, 0.8]))
0.5
```

**supports_continuous_scorings**() → bool

Whether this metric accepts continuous anomaly scorings as input (`True`) or binary classification labels (`False`).

### 3.6.15 timeeval.metrics.RangeRocAUC

**class** timeeval.metrics.**RangeRocAUC**(*buffer_size: Optional[int] = None*, *compatibility_mode: bool = False*, *max_samples: int = 250*, *plot: bool = False*, *plot_store: bool = False*)

Bases: `RangeAucMetric`

Computes the area under the receiver-operating-characteristic-curve using the range-based TPR and range-based FPR definition from Paparrizos et al. published at VLDB 2022 [PaparrizosEtAl2022].

We first extend the anomaly labels by two slopes of `buffer_size//2` length on both sides of each anomaly, uniformly sample thresholds from the anomaly score, and then compute the confusion matrix for all thresholds. Using the resulting false positive (FPR) and false positive rates (FPR), we can plot a curve and compute its area.

We make some changes to the original implementation from [PaparrizosEtAl2022] because we do not agree with the original assumptions. To reproduce the original results, you can set the parameter `compatibility_mode=True`. This will compute exactly the same values as the code by the authors of the paper.

The following things are different in TimeEval compared to the original version:

- For the recall (FPR) existence reward, we count anomalies as separate events, even if the added slopes overlap.
- Overlapping slopes don't sum up in their anomaly weight, but we just take to maximum anomaly weight for each point in the ground truth.
- The original slopes are asymmetric: The slopes at the end of anomalies are a single point shorter than the ones at the beginning of anomalies. We use symmetric slopes of the same size for the beginning and end of anomalies.
- We use a linear approximation of the slopes instead of the convex slope shape presented in the paper.

**Parameters**

- **buffer_size** (`Optional[int]`) – Size of the buffer region around an anomaly. We add an increasing slope of size `buffer_size//2` to the beginning of anomalies and a decreasing slope of size `buffer_size//2` to the end of anomalies. Per default (when `buffer_size==None`), `buffer_size` is the median length of the anomalies within the time

series. However, you can also set it to the period size of the dominant frequency or any other desired value.

- **compatibility_mode** (bool) – When set to True, produces exactly the same output as the metric implementation by the original authors. Otherwise, TimeEval uses a slightly improved implementation that fixes some bugs and uses linear slopes.
- **max_samples** (int) – Calculating precision and recall for many thresholds is quite slow. We, therefore, uniformly sample thresholds from the available score space. This parameter controls the maximum number of thresholds; too low numbers degrade the metrics' quality.
- **plot** (bool) –
- **plot_store** (bool) –

**See also:**

https://en.wikipedia.org/wiki/Receiver_operating_characteristic : Explanation of the ROC-curve.

**anomaly_bounds**(*y_true: ndarray*) → Tuple[ndarray, ndarray]

corresponds to range_convers_new

**property name: str**

Returns the unique name of this metric.

**score**(*y_true: ndarray*, *y_score: ndarray*) → float

Implementation of the metric's scoring function.

Please use `__call__()` instead of calling this function directly!

#### Examples

Instantiate a metric and call it using the `__call__` method:

```
>>> import numpy as np
>>> from timeeval.metrics import RocAUC
>>> metric = RocAUC(plot=False)
>>> metric(np.array([0, 1, 1, 0]), np.array([0.1, 0.4, 0.35, 0.8]))
0.5
```

**supports_continuous_scorings**() → bool

Whether this metric accepts continuous anomaly scorings as input (True) or binary classification labels (False).

### 3.6.16 timeeval.metrics.RangePrVUS

**class** timeeval.metrics.**RangePrVUS**(*max_buffer_size: int = 500*, *compatibility_mode: bool = False*, *max_samples: int = 250*)

Bases: RangeAucMetric

Computes the volume under the precision-recall-buffer_size-surface using the range-based precision and range-based recall definition from Paparrizos et al. published at VLDB 2022 [PaparrizosEtAl2022].

For all buffer sizes from 0 to max_buffer_size, we first extend the anomaly labels by two slopes of buffer_size//2 length on both sides of each anomaly, uniformly sample thresholds from the anomaly score, and then compute the confusion matrix for all thresholds. Using the resulting precision and recall values, we can plot a curve and compute its area.

This metric includes similar changes as *RangePrAUC*, which can be disabled using the `compatibility_mode` parameter.

**Parameters**

- **max_buffer_size** (int) – Maximum size of the buffer region around an anomaly. We iterate over all buffer sizes from 0 to `may_buffer_size` to create the surface.
- **compatibility_mode** (bool) – When set to `True`, produces exactly the same output as the metric implementation by the original authors. Otherwise, TimeEval uses a slightly improved implementation that fixes some bugs and uses linear slopes.
- **max_samples** (int) – Calculating precision and recall for many thresholds is quite slow. We, therefore, uniformly sample thresholds from the available score space. This parameter controls the maximum number of thresholds; too low numbers degrade the metrics' quality.

**See also:**

*timeeval.metrics.RangePrAUC*
Area under the curve version using a single buffer size.

**anomaly_bounds**(*y_true: ndarray*) → Tuple[ndarray, ndarray]

corresponds to range_convers_new

**property name: str**

Returns the unique name of this metric.

**score**(*y_true: ndarray*, *y_score: ndarray*) → float

Implementation of the metric's scoring function.

Please use `__call__()` instead of calling this function directly!

#### Examples

Instantiate a metric and call it using the `__call__` method:

```
>>> import numpy as np
>>> from timeeval.metrics import RocAUC
>>> metric = RocAUC(plot=False)
>>> metric(np.array([0, 1, 1, 0]), np.array([0.1, 0.4, 0.35, 0.8]))
0.5
```

**supports_continuous_scorings**() → bool

Whether this metric accepts continuous anomaly scorings as input (`True`) or binary classification labels (`False`).

### 3.6.17 timeeval.metrics.RangeRocVUS

**class** timeeval.metrics.**RangeRocVUS**(*max_buffer_size: int = 500*, *compatibility_mode: bool = False*, *max_samples: int = 250*)

Bases: `RangeAucMetric`

Computes the volume under the receiver-operating-characteristic-buffer_size-surface using the range-based TPR and range-based FPR definition from Paparrizos et al. published at VLDB 2022 [PaparrizosEtAl2022].

For all buffer sizes from 0 to `max_buffer_size`, we first extend the anomaly labels by two slopes of `buffer_size//2` length on both sides of each anomaly, uniformly sample thresholds from the anomaly score, and then compute the confusion matrix for all thresholds. Using the resulting false positive (FPR) and false positive rates (FPR), we can plot a curve and compute its area.

This metric includes similar changes as *RangeRocAUC*, which can be disabled using the `compatibility_mode` parameter.

**Parameters**

- **max_buffer_size** (int) – Maximum size of the buffer region around an anomaly. We iterate over all buffer sizes from 0 to `may_buffer_size` to create the surface.
- **compatibility_mode** (bool) – When set to `True`, produces exactly the same output as the metric implementation by the original authors. Otherwise, TimeEval uses a slightly improved implementation that fixes some bugs and uses linear slopes.
- **max_samples** (int) – Calculating precision and recall for many thresholds is quite slow. We, therefore, uniformly sample thresholds from the available score space. This parameter controls the maximum number of thresholds; too low numbers degrade the metrics' quality.

**See also:**

**https://en.wikipedia.org/wiki/Receiver_operating_characteristic :**
Explanation of the ROC-curve.

**timeeval.metrics.RangeRocAUC :**
Area under the curve version using a single buffer size.

### References

**anomaly_bounds**(*y_true: ndarray*) → Tuple[ndarray, ndarray]

corresponds to range_convers_new

**property name:** str

Returns the unique name of this metric.

**score**(*y_true: ndarray*, *y_score: ndarray*) → float

Implementation of the metric's scoring function.

Please use `__call__()` instead of calling this function directly!

#### Examples

Instantiate a metric and call it using the `__call__` method:

```
>>> import numpy as np
>>> from timeeval.metrics import RocAUC
>>> metric = RocAUC(plot=False)
>>> metric(np.array([0, 1, 1, 0]), np.array([0.1, 0.4, 0.35, 0.8]))
0.5
```

**supports_continuous_scorings**() → bool

Whether this metric accepts continuous anomaly scorings as input (`True`) or binary classification labels (`False`).

### 3.6.18 timeeval.metrics.DefaultMetrics

**class** timeeval.metrics.**DefaultMetrics**

Default metrics of TimeEval that can be used directly for time series anomaly detection algorithms without further configuration.

#### Examples

Using the default metric list that just contains ROC_AUC:

```
>>> from timeeval import TimeEval, DefaultMetrics
>>> TimeEval(dataset_mgr=..., datasets=[], algorithms=[],
>>>          metrics=DefaultMetrics.default_list())
```

You can also specify multiple default metrics:

```
>>> from timeeval import TimeEval, DefaultMetrics
>>> TimeEval(dataset_mgr=..., datasets=[], algorithms=[],
>>>          metrics=[DefaultMetrics.ROC_AUC, DefaultMetrics.PR_AUC, DefaultMetrics.
↪FIXED_RANGE_PR_AUC])
```

**AVERAGE_PRECISION = <timeeval.metrics.other_metrics.AveragePrecision object>**

**FIXED_RANGE_PR_AUC = <timeeval.metrics.range_metrics.RangePrecisionRangeRecallAUC object>**

**PR_AUC = <timeeval.metrics.auc_metrics.PrAUC object>**

**RANGE_F1 = <timeeval.metrics.range_metrics.RangeFScore object>**

**RANGE_PRECISION = <timeeval.metrics.range_metrics.RangePrecision object>**

**RANGE_PR_AUC = <timeeval.metrics.range_metrics.RangePrecisionRangeRecallAUC object>**

**RANGE_RECALL = <timeeval.metrics.range_metrics.RangeRecall object>**

**ROC_AUC = <timeeval.metrics.auc_metrics.RocAUC object>**

**static default**() → *Metric*

TimeEval's default metric ROC_AUC.

**static default_list**() → List[*Metric*]

The list containing TimeEval's single default metric ROC_AUC. For your convenience and usage as default parameter in many TimeEval library functions.

## 3.7 timeeval.metrics.thresholding package

### 3.7.1 timeeval.metrics.thresholding.ThresholdingStrategy

**class** timeeval.metrics.thresholding.**ThresholdingStrategy**

Bases: ABC

Takes an anomaly scoring and ground truth labels to compute and apply a threshold to the scoring.

Subclasses of this abstract base class define different strategies to put a threshold over the anomaly scorings. All strategies produce binary labels (0 or 1; 1 for anomalous) in the form of an integer NumPy array. The strategy *NoThresholding* is a special no-op strategy that checks for already existing binary labels and keeps them untouched. This allows applying the metrics on existing binary classification results.

**abstract find_threshold**(*y_true: ndarray*, *y_score: ndarray*) → float

Abstract method containing the actual code to determine the threshold. Must be overwritten by subclasses!

**fit**(*y_true: ndarray*, *y_score: ndarray*) → None

Calls *find_threshold()* to compute and set the threshold.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**fit_transform**(*y_true: ndarray*, *y_score: ndarray*) → ndarray

Determines the threshold and applies it to the scoring in one go.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**

**y_pred** – Array of binary labels; 0 for normal points and 1 for anomalous points.

**Return type**

np.ndarray

**See also:**

*fit*

fit-function to determine the threshold.

*transform*

transform-function to calculate the binary predictions.

**transform**(*y_score: ndarray*) → ndarray

Applies the threshold to the anomaly scoring and returns the corresponding binary labels.

**Parameters**

**y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**

**y_pred** – Array of binary labels; 0 for normal points and 1 for anomalous points.

**Return type**

np.ndarray

### 3.7.2 timeeval.metrics.thresholding.NoThresholding

**class** timeeval.metrics.thresholding.**NoThresholding**

Bases: *ThresholdingStrategy*

Special no-op strategy that checks for already existing binary labels and keeps them untouched. This allows applying the metrics on existing binary classification results.

**find_threshold**(*y_true: ndarray*, *y_score: ndarray*) → float

Does nothing (no-op).

**Parameters**

- **y_true** (np.ndarray) – Ignored.
- **y_score** (np.ndarray) – Ignored.

**Return type**
None

**fit**(*y_true: ndarray*, *y_score: ndarray*) → None

Does nothing (no-op).

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**transform**(*y_score: ndarray*) → ndarray

Checks if the provided scoring *y_score* is actually a binary classification prediction of integer type. If this is the case, the prediction is returned. If not, a ValueError is raised.

**Parameters**
**y_score** (np.ndarray) – Anomaly scoring with binary predictions.

**Returns**
**y_pred** – Array of binary labels; 0 for normal points and 1 for anomalous points.

**Return type**
np.ndarray

### 3.7.3 timeeval.metrics.thresholding.FixedValueThresholding

**class** timeeval.metrics.thresholding.**FixedValueThresholding**(*threshold: float = 0.8*)

Bases: *ThresholdingStrategy*

Thresholding approach using a fixed threshold value.

**Parameters**
**threshold** (float) – Fixed threshold to use. All anomaly scorings are scaled to the interval [0, 1]

**find_threshold**(*y_true: ndarray*, *y_score: ndarray*) → float

Returns the fixed threshold.

**fit**(*y_true: ndarray*, *y_score: ndarray*) → None

Calls *find_threshold()* to compute and set the threshold.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**fit_transform**(*y_true: ndarray*, *y_score: ndarray*) → ndarray

Determines the threshold and applies it to the scoring in one go.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**
**y_pred** – Array of binary labels; 0 for normal points and 1 for anomalous points.

**Return type**
np.ndarray

**See also:**

*fit*
fit-function to determine the threshold.

*transform*
transform-function to calculate the binary predictions.

**transform**(*y_score: ndarray*) → ndarray

Applies the threshold to the anomaly scoring and returns the corresponding binary labels.

**Parameters**
**y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**
**y_pred** – Array of binary labels; 0 for normal points and 1 for anomalous points.

**Return type**
np.ndarray

### 3.7.4 timeeval.metrics.thresholding.PercentileThresholding

**class** timeeval.metrics.thresholding.**PercentileThresholding**(*percentile: int = 90*)

Bases: *ThresholdingStrategy*

Use the xth-percentile of the anomaly scoring as threshold.

**Parameters**
**percentile** (int) – The percentile of the anomaly scoring to use. Must be between 0 and 100.

**find_threshold**(*y_true: ndarray*, *y_score: ndarray*) → float

Computes the xth-percentile ignoring NaNs and using a linear interpolation.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**
**threshold** – The xth-percentile of the anomaly scoring as threshold.

**Return type**
float

**fit**(*y_true: ndarray*, *y_score: ndarray*) → None

Calls *find_threshold()* to compute and set the threshold.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**fit_transform**(*y_true: ndarray*, *y_score: ndarray*) → ndarray

Determines the threshold and applies it to the scoring in one go.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**
**y_pred** – Array of binary labels; 0 for normal points and 1 for anomalous points.

**Return type**
np.ndarray

**See also:**

*fit*
fit-function to determine the threshold.

*transform*
transform-function to calculate the binary predictions.

**transform**(*y_score: ndarray*) → ndarray

Applies the threshold to the anomaly scoring and returns the corresponding binary labels.

**Parameters**
**y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**
**y_pred** – Array of binary labels; 0 for normal points and 1 for anomalous points.

**Return type**
np.ndarray

### 3.7.5 timeeval.metrics.thresholding.TopKPointsThresholding

**class** timeeval.metrics.thresholding.**TopKPointsThresholding**(*k: Optional[int] = None*)

Bases: *ThresholdingStrategy*

Calculates a threshold so that exactly *k* points are marked anomalous.

**Parameters**
**k** (optional int) – Number of expected anomalous points. If *k* is *None*, the ground truth data is used to calculate the real number of anomalous points.

**find_threshold**(*y_true: ndarray*, *y_score: ndarray*) → float

Computes a threshold based on the number of expected anomalous points.

The threshold is determined by taking the reciprocal ratio of expected anomalous points to all points as target percentile. We, again, ignore NaNs and use a linear interpolation. If *k* is *None*, the ground truth data is used to calculate the real ratio of anomalous points to all points. Otherwise, *k* is used as the number of expected anomalous points.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**
**threshold** – Threshold that yields k anomalous points.

**Return type**
float

**fit**(*y_true: ndarray*, *y_score: ndarray*) → None

Calls *find_threshold()* to compute and set the threshold.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**fit_transform**(*y_true: ndarray*, *y_score: ndarray*) → ndarray

Determines the threshold and applies it to the scoring in one go.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**
**y_pred** – Array of binary labels; 0 for normal points and 1 for anomalous points.

**Return type**
np.ndarray

**See also:**

*fit*
fit-function to determine the threshold.

*transform*
transform-function to calculate the binary predictions.

**transform**(*y_score: ndarray*) → ndarray

Applies the threshold to the anomaly scoring and returns the corresponding binary labels.

**Parameters**
**y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**
**y_pred** – Array of binary labels; 0 for normal points and 1 for anomalous points.

**Return type**
np.ndarray

### 3.7.6 timeeval.metrics.thresholding.TopKRangesThresholding

**class** timeeval.metrics.thresholding.**TopKRangesThresholding**(*k: Optional[int] = None*)

Bases: *ThresholdingStrategy*

Calculates a threshold so that exactly *k* anomalies are found. The anomalies are either single-points anomalies or continuous anomalous ranges.

**Parameters**
**k** (optional int) – Number of expected anomalies. If *k* is *None*, the ground truth data is used to calculate the real number of anomalies.

**find_threshold**(*y_true: ndarray*, *y_score: ndarray*) → float

Computes a threshold based on the number of expected anomalous subsequences / ranges (number of anomalies).

This method iterates over all possible thresholds from high to low to find the first threshold that yields *k* or more continuous anomalous ranges.

If *k* is *None*, the ground truth data is used to calculate the real number of anomalies (anomalous ranges).

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**
**threshold** – Threshold that yields k anomalies.

**Return type**
float

**fit**(*y_true: ndarray*, *y_score: ndarray*) → None

Calls *find_threshold()* to compute and set the threshold.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**fit_transform**(*y_true: ndarray*, *y_score: ndarray*) → ndarray

Determines the threshold and applies it to the scoring in one go.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**

**y_pred** – Array of binary labels; 0 for normal points and 1 for anomalous points.

**Return type**

np.ndarray

**See also:**

*fit*

fit-function to determine the threshold.

*transform*

transform-function to calculate the binary predictions.

**transform**(*y_score: ndarray*) → ndarray

Applies the threshold to the anomaly scoring and returns the corresponding binary labels.

**Parameters**

**y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**

**y_pred** – Array of binary labels; 0 for normal points and 1 for anomalous points.

**Return type**

np.ndarray

### 3.7.7 timeeval.metrics.thresholding.SigmaThresholding

**class** timeeval.metrics.thresholding.**SigmaThresholding**(*factor: float = 3.0*)

Bases: *ThresholdingStrategy*

Computes a threshold $\theta$ based on the anomaly scoring's mean $\mu_s$ and the standard deviation $\sigma_s$:

$$\theta = \mu_s + x \cdot \sigma_s$$

**Parameters**

**factor** (float) – Multiples of the standard deviation to be added to the mean to compute the threshold ($x$).

**find_threshold**(*y_true: ndarray*, *y_score: ndarray*) → float

Determines the mean and standard deviation ignoring NaNs of the anomaly scoring and computes the threshold using the mentioned equation.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**
**threshold** – Computed threshold based on mean and standard deviation.

**Return type**
float

**fit**(*y_true: ndarray*, *y_score: ndarray*) → None

Calls *find_threshold()* to compute and set the threshold.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**fit_transform**(*y_true: ndarray*, *y_score: ndarray*) → ndarray

Determines the threshold and applies it to the scoring in one go.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**
**y_pred** – Array of binary labels; 0 for normal points and 1 for anomalous points.

**Return type**
np.ndarray

**See also:**

*fit*
fit-function to determine the threshold.

*transform*
transform-function to calculate the binary predictions.

**transform**(*y_score: ndarray*) → ndarray

Applies the threshold to the anomaly scoring and returns the corresponding binary labels.

**Parameters**
**y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**
**y_pred** – Array of binary labels; 0 for normal points and 1 for anomalous points.

**Return type**
np.ndarray

### 3.7.8 timeeval.metrics.thresholding.PyThreshThresholding

**class** timeeval.metrics.thresholding.**PyThreshThresholding**(*pythresh_thresholder: BaseThresholder*, *random_state: Any = None*)

Bases: *ThresholdingStrategy*

Uses a thresholder from the PyThresh package to find a scoring threshold and to transform the continuous anomaly scoring into binary anomaly predictions.

**Warning:** You need to install PyThresh before you can use this thresholding strategy:

```
pip install pythresh>=0.2.8
```

Please note the additional package requirements for some available thresholders of PyThresh.

**Parameters**

- **pythresh_thresholder** (pythresh.thresholds.base.BaseThresholder) – Initiated PyThresh thresholder.
- **random_state** (Any) – Seed used to seed the numpy random number generator used in some thresholders of PyThresh. Note that PyThresh uses the legacy global RNG (np.random) and we try to reset the global RNG after calling PyThresh. Can be left at its default value for most thresholders that don't use random numbers or provide their own way of seeding. Please consult the PyThresh Documentation for details about the individual thresholders.

  Deprecated since version 1.2.8: Since pythresh version 0.2.8, thresholders provide a way to set their RNG state correctly. So the parameter random_state is not needed anymore. Please use the pythresh thresholder's parameter to seed it. This function's parameter is kept for compatibility with pythresh<0.2.8.

#### Examples

```
from timeeval.metrics.thresholding import PyThreshThresholding
from pythresh.thresholds.regr import REGR
import numpy as np

thresholding = PyThreshThresholding(
    REGR(method="theil")
)

y_scores = np.random.default_rng().random(1000)
y_labels = np.zeros(1000)
y_pred = thresholding.fit_transform(y_labels, y_scores)
```

**find_threshold**(*y_true: ndarray*, *y_score: ndarray*) → float

Uses the passed thresholder from the PyThresh package to determine the threshold. Beforehand, the scores are forced to be finite by replacing NaNs with 0 and (Neg)Infs with 1.

PyThresh thresholders directly compute the binary predictions. Thus, we cache the predictions in the member _predictions and return them when calling *transform()*.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.

- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**
**threshold** – Threshold computed by the internal thresholder.

**Return type**
float

**fit**(*y_true: ndarray*, *y_score: ndarray*) → None

Calls *find_threshold()* to compute and set the threshold.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**fit_transform**(*y_true: ndarray*, *y_score: ndarray*) → ndarray

Determines the threshold and applies it to the scoring in one go.

**Parameters**

- **y_true** (np.ndarray) – Ground truth binary labels.
- **y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**
**y_pred** – Array of binary labels; 0 for normal points and 1 for anomalous points.

**Return type**
np.ndarray

**See also:**

*fit*
fit-function to determine the threshold.

*transform*
transform-function to calculate the binary predictions.

**transform**(*y_score: ndarray*) → ndarray

Applies the threshold to the anomaly scoring and returns the corresponding binary labels.

**Parameters**
**y_score** (np.ndarray) – Anomaly scoring with continuous anomaly scores (same length as *y_true*).

**Returns**
**y_pred** – Array of binary labels; 0 for normal points and 1 for anomalous points.

**Return type**
np.ndarray

## 3.8 timeeval.params package

### 3.8.1 timeeval.params.ParameterConfig

**class** timeeval.params.**ParameterConfig**

Bases: ABC, Sized

Base class for algorithm hyperparameter configurations.

Currently, TimeEval supports three kinds of parameter configurations:

1. *FixedParameters*: A single parameter setting with one value for each parameter.
2. *IndependentParameterGrid* and *FullParameterGrid*: Parameter search using the specification of a parameter grid, where each parameter can have multiple values. Depending on the parameter grid, TimeEval will build a parameter search space and test all combinations of parameters.
3. *BayesianParameterSearch*: Parameter search using Bayesian optimization.

**static defaults**() → *ParameterConfig*

Returns the default parameter configuration that has only a single parameter setting with no parameters.

**abstract iter**(*algorithm:* Algorithm, *dataset:* Dataset) → Iterator[Params]

Iterate over the points in the grid.

**Returns**
**params** – Yields a params object that maps each parameter to a single value.

**Return type**
iterator over Params

### 3.8.2 timeeval.params.FixedParameters

**class** timeeval.params.**FixedParameters**(*params: Mapping[str, Any]*)

Bases: *ParameterConfig*

Single parameters setting with one value for each.

Iterating over this grid yields the input setting as the first and only element.

**Parameters**
**params** (dict of str to Any) – The parameter setting to be evaluated, as a dictionary mapping parameters to allowed values. An empty dict signifies default parameters.

#### Examples

```
>>> from timeeval.params import FixedParameters
>>> params = {"a": 2, "b": True}
>>> list(FixedParameters(params)) == (
...     [{"a": 2, "b": True}])
True
>>> FixedParameters(params)[0] == {"a": 2, "b": True}
True
```

**iter**(*algorithm:* Algorithm, *dataset:* Dataset) → Iterator[Params]

Iterate over the points in the grid.

**Returns**

**params** – Yields a params object that maps each parameter to a single value.

**Return type**

iterator over Params

### 3.8.3 timeeval.params.FullParameterGrid

**class** timeeval.params.**FullParameterGrid**(*param_grid: Mapping[str, Any]*)

Bases: ParameterGridConfig

Grid of parameters with a discrete number of values for each.

Iterating over this grid yields the full cartesian product of all available parameter combinations. Uses the sklearn.model_selection.ParameterGrid internally.

**Parameters**

**param_grid** (dict of str to sequence) – The parameter grid to explore, as a dictionary mapping parameters to sequences of allowed values. An empty dict signifies default parameters.

#### Examples

```
>>> from timeeval.params import FullParameterGrid
>>> params = {"a": [1, 2], "b": [True, False]}
>>> list(FullParameterGrid(params)) == (
...     [{"a": 1, "b": True}, {"a": 1, "b": False},
...      {"a": 2, "b": True}, {"a": 2, "b": False}])
True
>>> FullParameterGrid(params)[1] == {"a": 1, "b": False}
True
```

**See also:**

**sklearn.model_selection.ParameterGrid**

Used internally to represent the parameter grids.

**property param_grid: ParameterGrid**

The parameter search grid.

**Returns**

**param_grid** – A parameter search grid compatible with sklearn: sklearn.model_selection.ParameterGrid

**Return type**

sklearn parameter grid object

### 3.8.4 timeeval.params.IndependentParameterGrid

**class** timeeval.params.**IndependentParameterGrid**(*param_grid: Mapping[str, Any]*, *default_params: Optional[Mapping[str, Any]] = None*)

Bases: ParameterGridConfig

Grid of parameters with a discrete number of values for each.

The parameters in the dict are considered independent and explored one after the other (no cartesian product). Uses the sklearn.model_selection.ParameterGrid internally.

**Parameters**

- **param_grid** (dict of str to sequence, or sequence of such) – The parameter grid to explore, as either - a dictionary mapping parameters to sequences of allowed values, or - a sequence of dicts signifying a sequence of grids to search. An empty dict signifies default parameters.
- **default_params** (dict of str to any values) – Default values for the parameters that are not in the current parameter grid.

#### Examples

```
>>> from timeeval.params import IndependentParameterGrid
>>> params = {"a": [1, 2], "b": [True, False]}
>>> default_params = {"a": 1, "b": True, "c": "auto"}
>>> list(IndependentParameterGrid(params)) == ([
...     {"a": 1, "b": True, "c": "auto"},
...     {"a": 2, "b": True, "c": "auto"},
...     {"a": 1, "b": True, "c": "auto"},
...     {"a": 1, "b": False, "c": "auto"}
... ])
True
```

**See also:**

sklearn.model_selection.ParameterGrid
: Used internally to represent the parameter grids.

**property param_grid: ParameterGrid**

The parameter search grid.

**Returns**

**param_grid** – A parameter search grid compatible with sklearn: sklearn.model_selection.ParameterGrid

**Return type**

sklearn parameter grid object

### 3.8.5 timeeval.params.BayesianParameterSearch

**class** timeeval.params.**BayesianParameterSearch**(*config:* OptunaStudyConfiguration, *params:* *Mapping[str, BaseDistribution]*, *include_default_params:* *bool* = *False*)

Bases: *ParameterConfig*

Performs Bayesian optimization using Optuna integration.

---

**Note:** Please install the Optuna package to use this class. If you use the recommended PostgreSQL storage backend, you also need to install the *psycopg2* or *psycopg2-binary* package:

```
pip install optuna>=3.1.0 psycopg2
```

---

**Warning:** Parameter search using this class and the Optuna integration is **non-deterministic**. The results may vary between different runs, even if the same seed is used (e.g., for the Optuna sampler or pruner). This is because TimeEval needs to re-seed the Optuna samplers for every trial in distributed mode. This is necessary to ensure that initial random samples are different over all workers.

**Parameters**

- **config** (OptunaStudyConfiguration) – Configuration for the Optuna study. Optional parameters are filled in with the default values from the global Optuna configuration.
- **params** (Mapping[str, BaseDistribution]) – Mapping from parameter names to the corresponding Optuna distributions, such as IntDistribution, FloatDistribution, or CategoricalDistribution.

#### Examples

```
>>> from timeeval.params import BayesianParameterSearch
>>> from timeeval.metrics import RangePrAUC
>>> from timeeval.integration.optuna import OptunaStudyConfiguration
>>> from optuna.distributions import FloatDistribution, IntDistribution
>>> config = OptunaStudyConfiguration(n_trials=10, metric=RangePrAUC())
>>> distributions = {
...     "max_features": FloatDistribution(low=0.0, high=1.0, step=0.01),
...     "window_size": IntDistribution(low=5, high=1000, step=5),
... }
>>> BayesianParameterSearch(config, distributions)
<timeeval.params.bayesian.BayesianParameterSearch object at 0x7f9cdc8faf50>
```

**See also:**

**https://optuna.readthedocs.io:**
Optuna documentation.

***timeeval.integration.optuna.OptunaModule*:**
Optuna integration TimeEval module.

**iter**(*algorithm:* Algorithm, *dataset:* Dataset) → Iterator[Params]

Iterate over the points in the grid.

**Returns**
**params** – Yields a params object that maps each parameter to a single value.

**Return type**
`iterator over Params`

**update_config**(*global_config:* OptunaConfiguration) → None

Updates unset / default values in the study configuration with the global configuration values.

**Parameters**
**global_config** (`OptunaConfiguration`) – Global Optuna configuration.

## 3.9 timeeval.utils package

### 3.9.1 timeeval.utils.datasets module

timeeval.utils.datasets.**extract_features**(*df: DataFrame*) → ndarray

timeeval.utils.datasets.**extract_labels**(*df: DataFrame*) → ndarray

timeeval.utils.datasets.**load_dataset**(*path: Path*) → DataFrame

timeeval.utils.datasets.**load_labels_only**(*path: Path*) → ndarray

### 3.9.2 timeeval.utils.encode_params module

timeeval.utils.encode_params.**dump_params**(*params: Params*, *fh: Union[str, Path, TextIO]*) → None

timeeval.utils.encode_params.**dumps_params**(*params: Params*) → str

### 3.9.3 timeeval.utils.hash_dict module

timeeval.utils.hash_dict.**hash_dict**(*x: Mapping[Any, Any]*) → str

### 3.9.4 timeeval.utils.label_formatting module

timeeval.utils.label_formatting.**id2labels**(*ids: ndarray*, *data_length: int*) → ndarray

timeeval.utils.label_formatting.**labels2id**(*labels: ndarray*) → ndarray

### 3.9.5 timeeval.utils.results_path module

timeeval.utils.results_path.**generate_experiment_path**(*base_results_dir: Path*, *algorithm_name: str*, *hyper_params_id: str*, *collection_name: str*, *dataset_name: str*, *repetition_number: int*) → Path

### 3.9.6 timeeval.utils.tqdm_joblib module

timeeval.utils.tqdm_joblib.**tqdm_joblib**(*tqdm_object: tqdm*) → Generator[tqdm, None, None]

Context manager to patch joblib to report into tqdm progress bar given as argument.

Directly taken from https://stackoverflow.com/a/58936697.

#### Examples

```
>>> import time
>>> from joblib import Parallel, delayed
>>>
>>> def some_method(wait_time):
>>>     time.sleep(wait_time)
>>>
>>> with tqdm_joblib(tqdm(desc="Sleeping method", total=10)):
>>>     Parallel(n_jobs=2)(delayed(some_method)(0.2) for i in range(10))
```

### 3.9.7 timeeval.utils.window module

**class** timeeval.utils.window.**Method**(*value*)

Bases: Enum

An enumeration.

**MEAN = 0**

**MEDIAN = 1**

**SUM = 2**

**fn**(*x: ndarray*, *axis: Optional[int] = None*) → ndarray

**class** timeeval.utils.window.**ReverseWindowing**(*window_size: int*, *reduction: Method = Method.MEAN*, *n_jobs: int = 1*, *chunksize: Optional[int] = None*, *force_iterative: bool = False*)

Bases: TransformerMixin

**fit_transform**(*X: ndarray*, *y=None*, ***fit_params*) → ndarray

Fit to data, then transform it.

Fits transformer to *X* and *y* with optional parameters *fit_params* and returns a transformed version of *X*.

**Parameters**

- **X** (array-like of shape (n_samples, n_features)) – Input samples.

- **y** (array-like of shape (n_samples,) or (n_samples, n_outputs), default=None) – Target values (None for unsupervised transformations).
- ****fit_params** (dict) – Additional fit parameters.

**Returns**
**X_new** – Transformed array.

**Return type**
ndarray array of shape (n_samples, n_features_new)

timeeval.utils.window.**padding_borders**(*scores: ndarray*, *input_size: int*) → ndarray

## 3.10 timeeval._core package

**Warning:** This package contains TimeEval-internal classes and functions. Do not use or change!

## 3.11 timeeval.integration package

Available integration modules:

### 3.11.1 Optuna integration

Optuna is an automatic hyperparameter optimization framework and this integration allows you to use it within TimeEval. TimeEval will load the *OptunaModule* automatically if at least one algorithm uses *BayesianParameterSearch* as its parameter search strategy. Please make sure that you install the required dependencies for Optuna before using this integration (we also recommend to install *psycopg2* to use the PostgreSQL storage backend):

```
pip install 'optuna>=3.1.0' psycopg2
```

The following Optuna features are supported:

- Definition of search spaces using Optuna distributions for each algorithm (one study per algorithm): *BayesianParameterSearch*.
- Configurable samplers.
- Configurable storage backends (in-memory, RDB, Journal, etc.).
- Resuming of existing studies (via RDB storage backend).
- Parallel and distributed parameter search of a single or multiple studies (synchronized via RDB storage backend).

**Warning:** Parameter search using the Optuna integration is **non-deterministic**. The results may vary between different runs, even if the same seed is used (e.g., for the Optuna sampler or pruner). This is because TimeEval needs to re-seed the Optuna samplers for every trial in distributed mode. This is necessary to ensure that initial random samples are different over all workers.

TimeEval will automatically manage an RDB storage backend if you use the default configuration. This allows you to start TimeEval in distributed mode and perform the parameter search in parallel and distributed.

### timeeval.integration.optuna.OptunaModule

**class** timeeval.integration.optuna.**OptunaModule**(*config:* OptunaConfiguration)

Bases: *TimeEvalModule*

This module is automatically loaded when at least one algorithm uses *timeeval.params.BayesianParameterSearch* as parameter config.

TimeEval provides the option to use an automatically managed PostgreSQL database as the storage backend for the Optuna studies. The database is started as an additional Docker container either on the local machine or on the scheduler node in distributed execution mode. The database is automatically stopped when TimeEval is finished. The database storage backend allows you to monitor the studies using the Optuna dashboard (that can also be started automatically using another Docker container) and the distributed execution of the studies. This is the default behavior if no storage backend is specified in the configuration.

**Parameters**
**config** (*OptunaConfiguration*) – The configuration for the Optuna module.

**finalize**(*timeeval:* TimeEval) → None

Called during the FINALIZE-phase of TimeEval and after the individual algorithms' finalize-functions were executed.

**Parameters**
**timeeval** (TimeEval) – The TimeEval instance that is currently running.

**load_studies**() → List[StudySummary]

Load all studies from the default storage. This does not include studies, which were stored in a different storage backend (i.e. where the storage backend was changed using the *timeeval.integration.optuna.OptunaStudyConfiguration*).

**Returns**
**study_summaries** – A list of study summaries.

**Return type**
List[StudySummary]

**See also:**

**optuna.study.get_all_study_summaries()**
Optuna function which is used to load the studies.

**optuna.study.StudySummary**
Optuna class which is used to represent the study summaries.

**prepare**(*timeeval:* TimeEval) → None

Called during the PREPARE-phase of TimeEval and before the individual algorithms' prepare-functions are executed.

**Parameters**
**timeeval** (TimeEval) – The TimeEval instance that is currently running.

### timeeval.integration.optuna.OptunaConfiguration

class timeeval.integration.optuna.**OptunaConfiguration**(*default_storage: Union[str, Callable[[], BaseStorage]]*, *default_sampler: Optional[BaseSampler] = None*, *default_pruner: Optional[BasePruner] = None*, *continue_existing_studies: bool = False*, *dashboard: bool = False*, *remove_managed_containers: bool = False*, *use_default_logging: bool = False*, *log_level: Union[int, str] = 20*)

Bases: object

Configuration options for the Optuna module. This includes default options for all Optuna studies.

**Parameters**

- **default_storage** (str or Lambda returning instance of optuna.storages.BaseStorage) – Storage to store and synchronize the results of the studies. Per default, TimeEval will use a journal file in local execution mode and a PostgreSQL database in distributed execution mode. The database is automatically started and stopped by TimeEval using the latest postgres-Docker image. Use "postgresql" to let TimeEval handle starting and stopping a PostgreSQL database using Docker. Use "journal-file" to let TimeEval create a local file as the storage backend. This only works in non-distributed mode.
- **default_sampler** (optuna.samplers.BaseSampler, *optional*) – Sampler to use for the study. If not provided, the default sampler is used.
- **default_pruner** (optuna.pruners.BasePruner, *optional*) – Pruner to use for the study. If not provided, the default pruner is used.
- **continue_existing_studies** (bool, *optional*) – If True, continue a study with the given name if it already exists in the storage backend. If False, raise an error if a study with the same name already exists.
- **dashboard** (bool, *optional*) – If True, start the Optuna dashboard (within its own Docker container) to monitor the studies. In distributed execution mode, the dashboard is started on the scheduler node.
- **remove_managed_containers** (bool, *optional*) – If True, remove the containers managed by TimeEval (e.g., the PostgreSQL database) when TimeEval is finished.
- **use_default_logging** (bool, *optional*) – If True, use the default logging configuration of the Optuna library. This will log the progress of the studies to *stderr*. If False, use the logging configuration of TimeEval and propagates the Optuna log messages.
- **log_level** (int or str, *optional*) – The log level to use for the Optuna logger. The default is info = logging.INFO = 20.

**See also:**

optuna.create_study()
: Used to create the Optuna study object; includes detailed explanation of the parameters.

*timeeval.integration.optuna.OptunaModule*
: Optuna integration module for TimeEval.

**continue_existing_studies: bool = False**

**dashboard:** bool = False

**static default**(*distributed: bool*) → *OptunaConfiguration*

**default_pruner:** Optional[BasePruner] = None

**default_sampler:** Optional[BaseSampler] = None

**default_storage:** Union[str, Callable[[], BaseStorage]]

**log_level:** Union[int, str] = 20

**remove_managed_containers:** bool = False

**use_default_logging:** bool = False

### timeeval.integration.optuna.OptunaStudyConfiguration

**class** timeeval.integration.optuna.**OptunaStudyConfiguration**(*n_trials: int*, *metric:* Metric, *storage: Optional[Union[str, Callable[[], BaseStorage]]] = None*, *sampler: Optional[BaseSampler] = None*, *pruner: Optional[BasePruner] = None*, *direction: Optional[Union[str, StudyDirection]] = 'maximize'*, *continue_existing_study: bool = False*)

Bases: object

Configuration for *BayesianParameterSearch*.

The parameters `n_trials` and `metric` are required. All other parameters are optional and will be filled with the default values from the global Optuna configuration if not provided.

**Parameters**

- **n_trials** (int) – Number of trials to perform.
- **metric** (Metric) – TimeEval metric to use as the studies objective function.
- **storage** (str or Lambda returning instance of optuna.storages.BaseStorage, *optional*) – Storage to store the results of the study.
- **sampler** (optuna.samplers.BaseSampler, *optional*) – Sampler to use for the study. If not provided, the default sampler is used.
- **pruner** (optuna.pruners.BasePruner, *optional*) – Pruner to use for the study. If not provided, the default pruner is used.
- **direction** (str or optuna.study.StudyDirection, *optional*) – Direction of the optimization (minimize or *maximize*). If None, the Optuna default direction is used.
- **continue_existing_study** (bool, *optional*) – If True, continue a study with the given name if it already exists in the storage backend. If False, raise an error if a study with the same name already exists.

**See also:**

optuna.create_study()
: Used to create the Optuna study object; includes detailed explanation of the parameters.

*timeeval.integration.optuna.OptunaModule*
Optuna integration module for TimeEval.

**continue_existing_study:** bool = False

**copy**(*n_trials: Optional[int] = None*, *metric: Optional[Metric] = None*, *storage: Optional[Union[str, Callable[[], BaseStorage]]] = None*, *sampler: Optional[BaseSampler] = None*, *pruner: Optional[BasePruner] = None*, *direction: Optional[Union[str, StudyDirection]] = None*, *continue_existing_study: Optional[bool] = None*) → *OptunaStudyConfiguration*
Create a copy of this configuration with the given parameters replaced.

**direction:** Optional[Union[str, StudyDirection]] = 'maximize'

**metric:** *Metric*

**n_trials:** int

**pruner:** Optional[BasePruner] = None

**sampler:** Optional[BaseSampler] = None

**storage:** Optional[Union[str, Callable[[], BaseStorage]]] = None

**update_unset_options**(*global_config:* OptunaConfiguration) → *OptunaStudyConfiguration*

### 3.11.2 Base class for TimeEval modules

**class** timeeval.integration.**TimeEvalModule**
Bases: ABC

Base class for TimeEval modules that add additional functionality to TimeEval.

Inheriting classes can implement any of the following lifecycle-hooks:

1. *prepare()*
2. *pre_run()*
3. *post_run()*
4. *finalize()*

These methods are called at the corresponding time in the TimeEval run loop. Modules can assume that the TimeEval configuration is already loaded and checked for user errors. If TimeEval is executed in distributed mode, `timeeval.TimeEval.distributed` is set to `True` and remoting is already set up before the first call to *prepare()*.

---

**Note:** Implementing a TimeEval module is an advanced usage scenario and requires a good understanding of the internals of TimeEval.

---

**finalize**(*timeeval:* TimeEval) → None
Called during the FINALIZE-phase of TimeEval and after the individual algorithms' finalize-functions were executed.

**Parameters**
**timeeval** (TimeEval) – The TimeEval instance that is currently running.

**post_run**(*timeeval:* TimeEval) → None

Called after the EVALUATION-phase of TimeEval.

**Parameters**
**timeeval** (TimeEval) – The TimeEval instance that is currently running.

**pre_run**(*timeeval:* TimeEval) → None

Called before the EVALUATION-phase of TimeEval.

**Parameters**
**timeeval** (TimeEval) – The TimeEval instance that is currently running.

**prepare**(*timeeval:* TimeEval) → None

Called during the PREPARE-phase of TimeEval and before the individual algorithms' prepare-functions are executed.

**Parameters**
**timeeval** (TimeEval) – The TimeEval instance that is currently running.

CHAPTER

FOUR

# CONTRIBUTOR'S GUIDE

## 4.1 Installation from source

**tl;dr**

```
git clone git@github.com:timeeval/timeeval.git
cd timeeval/
conda create -n timeeval python=3.7
conda activate timeeval
pip install -r requirements.txt
python setup.py install
```

### 4.1.1 Prerequisites

The following tools are required to install TimeEval from source:

- git
- Python > 3.7 and Pip (anaconda or miniconda is preferred)

### 4.1.2 Steps

1. Clone this repository using git and change into its root directory.
2. Create a conda-environment and install all required dependencies.

   ```
   conda create -n timeeval python=3.7
   conda activate timeeval
   pip install -r requirements.txt
   ```

3. Build TimeEval: `python setup.py bdist_wheel`. This should create a Python wheel in the `dist/`-folder.
4. Install TimeEval and all of its dependencies: `pip install dist/TimeEval-*-py3-none-any.whl`.
5. If you want to make changes to TimeEval or run the tests, you need to install the development dependencies from `requirements.dev`: `pip install -r requirements.dev`.

## 4.2 Tests

Run tests in `./tests/` as follows

```
python setup.py test
```

or

```
pytest tests
```

If you want to run the tests that include docker and dask, you need to fulfill some prerequisites:

- Docker is installed and running.
- Your SSH-server is running, and you can SSH to `localhost` with your user without supplying a password.
- You have installed all TimeEval dev dependencies.

You can then run:

```
pytest tests --docker --dask
```

### 4.2.1 Default Tests

By default, tests that are marked with the following keys are skipped:

- docker
- dask

To run these tests, add the respective keys as parameters:

```
pytest --[key] # e.g. --docker
```

CHAPTER

# FIVE

# OVERVIEW

TimeEval is an evaluation tool for time series anomaly detection algorithms. It defines common interfaces for datasets and algorithms to allow the efficient comparison of the algorithms' quality and runtime performance. TimeEval can be configured using a simple Python API and comes with a large collection of compatible datasets and algorithms.

TimeEval takes your input and automatically creates experiment configurations by taking the cross-product of your inputs. It executes all experiment configurations one after the other or — when distributed — in parallel and records the anomaly detection quality and the runtime of the algorithms.

TimeEval takes four inputs for the experiment creation:

1. Algorithms
2. Datasets
3. Algorithm hyperparameter specifications
4. A repetition number

The following code snippet shows a simple example experiment evaluating COF and a simple baseline algorithm on some test data:

Listing 1: Example usage of TimeEval

```
#!/usr/bin/env python3
from pathlib import Path
from typing import Any, Dict
import numpy as np

from timeeval import TimeEval, DatasetManager, DefaultMetrics, Algorithm, TrainingType,␣
↪InputDimensionality
from timeeval.adapters import FunctionAdapter # for defining customized algorithm
from timeeval.algorithms import cof
from timeeval.data_types import AlgorithmParameter
from timeeval.params import FixedParameters

def your_algorithm_function(data: AlgorithmParameter, args: Dict[str, Any]) -> np.
↪ndarray:
    if isinstance(data, np.ndarray):
        return np.zeros_like(data)
    else: # isinstance(data, pathlib.Path)
        return np.genfromtxt(data, delimiter=",", skip_header=1)[:, 1]

def main():
    dm = DatasetManager(Path("tests/example_data"), create_if_missing=False)
```

(continues on next page)

(continued from previous page)

```
20      datasets = dm.select()
21      algorithms = [
22      # list of algorithms which will be executed on the selected dataset(s)
23          cof(
24              params=FixedParameters({
25                  "n_neighbors": 20,
26                  "random_state": 42})
27          ),
28          # calling customized algorithm
29          Algorithm(
30              name="MyPythonFunctionAlgorithm",
31              main=FunctionAdapter(your_algorithm_function),
32              data_as_file=False)
33      ]
34
35      timeeval = TimeEval(dm, datasets, algorithms, metrics=[DefaultMetrics.ROC_AUC,␣
↪DefaultMetrics.RANGE_PR_AUC])
36      timeeval.run()
37      results = timeeval.get_results(aggregated=False)
38      print(results)
39
40
41  if __name__ == "__main__":
42      main()
```

## 5.1 Features

Listing *Example usage of TimeEval* illustrates some main features of TimeEval:

**Dataset API:**
*Interface* to available dataset collection to select datasets easily (L21-22).

**Algorithm Adapter Architecture:**
TimeEval supports different algorithm adapters to execute simple Python functions or whole pipelines and applications (L27, L38).

**Hyperparameter Specification:**
Algorithm hyperparameters can be specified using different search grids (L28-31).

**Metrics:**
TimeEval provides various evaluation metrics (such as `timeeval.utils.metrics.DefaultMetrics.ROC_AUC`, `timeeval.utils.metrics.DefaultMetrics.RANGE_PR_AUC`, or `timeeval.utils.metrics.FScoreAtK`) and measures algorithm runtimes automatically (L43).

**Distributed Execution:**
TimeEval can be deployed on a compute cluster to execute evaluation tasks distributedly.

## 5.2 Installation

Prerequisites:

- Python 3.7, 3.8, or 3.9
- Docker
- rsync (if you want to use distributed TimeEval)

TimeEval is published to PyPI and you can install it using:

```
pip install TimeEval
```

**Attention:** Currently, TimeEval is tested only on Linux systems and relies on unix-oid capabilities.

## 5.3 License

The project is licensed under the MIT license.

If you use TimeEval in your project or research, please cite our demonstration paper:

> Phillip Wenig, Sebastian Schmidl, and Thorsten Papenbrock. TimeEval: A Benchmarking Toolkit for Time Series Anomaly Detection Algorithms. PVLDB, 15(12): 3678 - 3681, 2022. doi:10.14778/3554821.3554873

You can use the following BibTeX entry:

```
@article{WenigEtAl2022TimeEval,
  title = {TimeEval: {{A}} Benchmarking Toolkit for Time Series Anomaly Detection␣
↪Algorithms},
  author = {Wenig, Phillip and Schmidl, Sebastian and Papenbrock, Thorsten},
  date = {2022},
  journaltitle = {Proceedings of the {{VLDB Endowment}} ({{PVLDB}})},
  volume = {15},
  number = {12},
  pages = {3678--3681},
  doi = {10.14778/3554821.3554873}
}
```

CHAPTER

SIX

# USER GUIDE

New to TimeEval? Check out our *User Guides* to get started with TimeEval. The user guides explain TimeEval's API and how to use it to achieve your goal.

*To the user guide*

CHAPTER

SEVEN

# TIMEEVAL CONCEPTS

Background information and in-depth explanations about how TimeEval works can be found in the *TimeEval concepts reference*.

*To the concepts*

CHAPTER

EIGHT

# API REFERENCE

The API reference guide contains a detailed description of the functions, modules, and objects included in TimeEval. The API reference describes how the methods work and which parameters can be used.

*To the API reference*

CHAPTER

NINE

# CONTRIBUTOR'S GUIDE

Want to add to the codebase? You can help with the documentation? The contributing guidelines will guide you through the process of improving TimeEval and its ecosystem.

*To the contributor's guide*

CHAPTER

TEN

# ADDITIONAL LINKS

- TimeEval Github repository
- TimeEval algorithms
- Datasets for TimeEval
- TimeEval GUI (prototype)
- Time series anomaly dataset generator GutenTAG
- modindex
- search

# BIBLIOGRAPHY

[TatbulEtAl2018] Tatbul, Nesime, Tae Jun Lee, Stan Zdonik, Mejbah Alam, and Justin Gottschlich. "Precision and Recall for Time Series." In Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS), 1920–30. 2018. http://papers.nips.cc/paper/7462-precision-and-recall-for-time-series.pdf.

[PaparrizosEtAl2022] John Paparrizos, Paul Boniol, Themis Palpanas, Ruey S. Tsay, Aaron Elmore, and Michael J. Franklin. Volume Under the Surface: A New Accuracy Evaluation Measure for Time-Series Anomaly Detection. PVLDB, 15(11): 2774 - 2787, 2022. doi:10.14778/3551793.3551830

# PYTHON MODULE INDEX

## t


timeeval, 29
timeeval.adapters.base, 42
timeeval.adapters.distributed, 42
timeeval.adapters.docker, 43
timeeval.adapters.function, 45
timeeval.adapters.jar, 45
timeeval.adapters.multivar, 45
timeeval.constants, 40
timeeval.datasets.analyzer, 115
timeeval.datasets.custom, 117
timeeval.datasets.custom_base, 118
timeeval.datasets.custom_noop, 118
timeeval.datasets.dataset, 119
timeeval.datasets.dataset_manager, 119
timeeval.datasets.datasets, 126
timeeval.datasets.metadata, 130
timeeval.datasets.multi_dataset_manager, 132
timeeval.heuristics, 136
timeeval.metrics, 143
timeeval.resource_constraints, 40
timeeval.utils.datasets, 176
timeeval.utils.encode_params, 176
timeeval.utils.hash_dict, 176
timeeval.utils.label_formatting, 176
timeeval.utils.results_path, 177
timeeval.utils.tqdm_joblib, 177
timeeval.utils.window, 177

# INDEX


## A

Adapter (*class in timeeval.adapters.base*), 42
add_dataset() (*timeeval.datasets.dataset_manager.DatasetManager method*), 120
add_datasets() (*timeeval.datasets.dataset_manager.DatasetManager method*), 120
AggregationMethod (*class in timeeval.adapters.multivar*), 45
akita_dataset_paths (*timeeval.constants.HPI_CLUSTER attribute*), 40
Algorithm (*class in timeeval*), 34
AlgorithmInterface (*class in timeeval.adapters.docker*), 43
anomaly_bounds() (*timeeval.metrics.RangePrAUC method*), 156
anomaly_bounds() (*timeeval.metrics.RangePrVUS method*), 159
anomaly_bounds() (*timeeval.metrics.RangeRocAUC method*), 158
anomaly_bounds() (*timeeval.metrics.RangeRocVUS method*), 160
anomaly_length (*timeeval.datasets.metadata.DatasetMetadata attribute*), 130
AnomalyLength (*class in timeeval.datasets.metadata*), 130
AnomalyLengthHeuristic (*class in timeeval.heuristics*), 138
arima() (*in module timeeval.algorithms*), 46
autoencoder() (*in module timeeval.algorithms*), 47
AVERAGE_PRECISION (*timeeval.metrics.DefaultMetrics attribute*), 161
AveragePrecision (*class in timeeval.metrics*), 148

## B

bagel() (*in module timeeval.algorithms*), 48
baseline_increasing() (*in module timeeval.algorithms*), 49
baseline_normal() (*in module timeeval.algorithms*), 49
baseline_random() (*in module timeeval.algorithms*), 50
BayesianParameterSearch (*class in timeeval.params*), 175
BENCHMARK (*timeeval.constants.HPI_CLUSTER attribute*), 40

## C

cblof() (*in module timeeval.algorithms*), 50
CDEntry (*class in timeeval.datasets.custom*), 117
channels (*timeeval.datasets.metadata.DatasetMetadata property*), 130
CleanStartSequenceSizeHeuristic (*class in timeeval.heuristics*), 139
coef (*timeeval.datasets.metadata.Trend attribute*), 132
cof() (*in module timeeval.algorithms*), 51
collection_name (*timeeval.datasets.dataset.Dataset property*), 119
collection_name (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 124
confidence_r2 (*timeeval.datasets.metadata.Trend attribute*), 132
contamination (*timeeval.datasets.dataset.Dataset attribute*), 119
contamination (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 124
contamination (*timeeval.datasets.metadata.DatasetMetadata attribute*), 130
ContaminationHeuristic (*class in timeeval.heuristics*), 139
continue_existing_studies (*timeeval.integration.optuna.OptunaConfiguration attribute*), 180
continue_existing_study (*timeeval.integration.optuna.OptunaStudyConfiguration attribute*), 182
copod() (*in module timeeval.algorithms*), 52

copy() (*timeeval.integration.optuna.OptunaStudyConfiguration method*), 182
CORRELATION_ANOMALIES (*timeeval.constants.HPI_CLUSTER attribute*), 40
count() (*timeeval.datasets.dataset_manager.DatasetRecord method*), 124
CustomDatasets (*class in timeeval.datasets.custom*), 117
CustomDatasetsBase (*class in timeeval.datasets.custom_base*), 118
customParameters (*timeeval.adapters.docker.AlgorithmInterface attribute*), 43

## D

dae() (*in module timeeval.algorithms*), 52
damp() (*in module timeeval.algorithms*), 53
dashboard (*timeeval.integration.optuna.OptunaConfiguration attribute*), 180
dask_logging_console_level (*timeeval.RemoteConfiguration attribute*), 37
dask_logging_file_level (*timeeval.RemoteConfiguration attribute*), 37
dask_logging_filename (*timeeval.RemoteConfiguration attribute*), 37
data_as_file (*timeeval.Algorithm attribute*), 35
dataInput (*timeeval.adapters.docker.AlgorithmInterface attribute*), 43
dataOutput (*timeeval.adapters.docker.AlgorithmInterface attribute*), 43
Dataset (*class in timeeval.datasets.dataset*), 119
dataset_id (*timeeval.datasets.metadata.DatasetMetadata attribute*), 130
dataset_name (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125
dataset_type (*timeeval.datasets.dataset.Dataset attribute*), 119
dataset_type (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125
DatasetAnalyzer (*class in timeeval.datasets.analyzer*), 115
datasetId (*timeeval.datasets.dataset.Dataset attribute*), 119
DatasetIdHeuristic (*class in timeeval.heuristics*), 139
DatasetManager (*class in timeeval.datasets.dataset_manager*), 119
DatasetMetadata (*class in timeeval.datasets.metadata*), 130
DatasetMetadataEncoder (*class in timeeval.datasets.metadata*), 131
DatasetRecord (*class in timeeval.datasets.dataset_manager*), 124
Datasets (*class in timeeval.datasets.datasets*), 126
datetime_index (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125
dbstream() (*in module timeeval.algorithms*), 54
deepant() (*in module timeeval.algorithms*), 55
deepnap() (*in module timeeval.algorithms*), 56
default() (*timeeval.adapters.docker.DockerJSONEncoder method*), 44
default() (*timeeval.datasets.metadata.DatasetMetadataEncoder method*), 131
default() (*timeeval.integration.optuna.OptunaConfiguration static method*), 181
default() (*timeeval.metrics.DefaultMetrics static method*), 161
default_constraints() (*timeeval.ResourceConstraints static method*), 38
default_list() (*timeeval.metrics.DefaultMetrics static method*), 161
default_pruner (*timeeval.integration.optuna.OptunaConfiguration attribute*), 181
DEFAULT_RESULT_PATH (*timeeval.TimeEval attribute*), 32
default_sampler (*timeeval.integration.optuna.OptunaConfiguration attribute*), 181
default_storage (*timeeval.integration.optuna.OptunaConfiguration attribute*), 181
DefaultExponentialFactorHeuristic (*class in timeeval.heuristics*), 140
DefaultFactorHeuristic (*class in timeeval.heuristics*), 140
DefaultMetrics (*class in timeeval.metrics*), 161
defaults() (*timeeval.params.ParameterConfig static method*), 172
details (*timeeval.datasets.custom.CDEntry attribute*), 117
df() (*timeeval.datasets.dataset_manager.DatasetManager method*), 120
df() (*timeeval.datasets.datasets.Datasets method*), 126
df() (*timeeval.datasets.multi_dataset_manager.MultiDatasetManager method*), 132
DIFFERENCE_STATIONARY (*timeeval.datasets.metadata.Stationarity attribute*), 131
dimensions (*timeeval.datasets.dataset.Dataset attribute*), 119
dimensions (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125
dimensions (*timeeval.datasets.metadata.DatasetMetadata attribute*), 130
direction (*timeeval.integration.optuna.OptunaStudyConfiguration*

*attribute*), 182
DistributedAdapter (*class in timeeval.adapters.distributed*), 42
DockerAdapter (*class in timeeval.adapters.docker*), 43
DockerAdapterInternalError, 44
DockerAlgorithmFailedError, 44
DockerJSONEncoder (*class in timeeval.adapters.docker*), 44
DockerMemoryError, 44
DockerTimeoutError, 44
donut() (*in module timeeval.algorithms*), 57
dspot() (*in module timeeval.algorithms*), 58
dump_params() (*in module timeeval.utils.encode_params*), 176
dumps_params() (*in module timeeval.utils.encode_params*), 176
dwt_mlead() (*in module timeeval.algorithms*), 59

## E

eif() (*in module timeeval.algorithms*), 59
EmbedDimRangeHeuristic (*class in timeeval.heuristics*), 141
encdec_ad() (*in module timeeval.algorithms*), 60
ensemble_gi() (*in module timeeval.algorithms*), 61
ERROR (*timeeval.Status attribute*), 34
EXECUTE (*timeeval.data_types.ExecutionType attribute*), 42
execute_timeout (*timeeval.ResourceConstraints attribute*), 38
ExecutionType (*class in timeeval.data_types*), 42
executionType (*timeeval.adapters.docker.AlgorithmInterface attribute*), 43
extract_features() (*in module timeeval.utils.datasets*), 176
extract_labels() (*in module timeeval.utils.datasets*), 176

## F

F1Score (*class in timeeval.metrics*), 151
fast_mcd() (*in module timeeval.algorithms*), 62
fft() (*in module timeeval.algorithms*), 63
finalize() (*timeeval.integration.optuna.OptunaModule method*), 179
finalize() (*timeeval.integration.TimeEvalModule method*), 182
find_threshold() (*timeeval.metrics.thresholding.FixedValueThresholding method*), 163
find_threshold() (*timeeval.metrics.thresholding.NoThresholding method*), 163
find_threshold() (*timeeval.metrics.thresholding.PercentileThresholding method*), 164
find_threshold() (*timeeval.metrics.thresholding.PyThreshThresholding method*), 170
find_threshold() (*timeeval.metrics.thresholding.SigmaThresholding method*), 168
find_threshold() (*timeeval.metrics.thresholding.ThresholdingStrategy method*), 162
find_threshold() (*timeeval.metrics.thresholding.TopKPointsThresholding method*), 166
find_threshold() (*timeeval.metrics.thresholding.TopKRangesThresholding method*), 167
fit() (*timeeval.metrics.thresholding.FixedValueThresholding method*), 163
fit() (*timeeval.metrics.thresholding.NoThresholding method*), 163
fit() (*timeeval.metrics.thresholding.PercentileThresholding method*), 165
fit() (*timeeval.metrics.thresholding.PyThreshThresholding method*), 171
fit() (*timeeval.metrics.thresholding.SigmaThresholding method*), 169
fit() (*timeeval.metrics.thresholding.ThresholdingStrategy method*), 162
fit() (*timeeval.metrics.thresholding.TopKPointsThresholding method*), 166
fit() (*timeeval.metrics.thresholding.TopKRangesThresholding method*), 167
fit_transform() (*timeeval.metrics.thresholding.FixedValueThresholding method*), 164
fit_transform() (*timeeval.metrics.thresholding.PercentileThresholding method*), 165
fit_transform() (*timeeval.metrics.thresholding.PyThreshThresholding method*), 171
fit_transform() (*timeeval.metrics.thresholding.SigmaThresholding method*), 169
fit_transform() (*timeeval.metrics.thresholding.ThresholdingStrategy method*), 162
fit_transform() (*timeeval.metrics.thresholding.TopKPointsThresholding method*), 166
fit_transform() (*timeeval.metrics.thresholding.TopKRangesThresholding method*), 167
fit_transform() (*timee-*

*val.utils.window.ReverseWindowing method*), 177
FIXED_RANGE_PR_AUC (*timeeval.metrics.DefaultMetrics attribute*), 161
FixedParameters (*class in timeeval.params*), 172
FixedValueThresholding (*class in timeeval.metrics.thresholding*), 163
fn() (*timeeval.utils.window.Method method*), 177
from_dimensions() (*timeeval.InputDimensionality static method*), 36
from_json() (*timeeval.datasets.metadata.DatasetMetadata static method*), 130
from_name() (*timeeval.datasets.metadata.Stationarity static method*), 131
from_order() (*timeeval.datasets.metadata.TrendType static method*), 132
from_text() (*timeeval.TrainingType static method*), 36
FScoreAtK (*class in timeeval.metrics*), 154
FullParameterGrid (*class in timeeval.params*), 173
FunctionAdapter (*class in timeeval.adapters.function*), 45

## G

GB (*in module timeeval.resource_constraints*), 40
generate_experiment_path() (*in module timeeval.utils.results_path*), 177
generic_rf() (*in module timeeval.algorithms*), 63
generic_xgb() (*in module timeeval.algorithms*), 65
get() (*timeeval.datasets.custom.CustomDatasets method*), 117
get() (*timeeval.datasets.custom_base.CustomDatasetsBase method*), 118
get() (*timeeval.datasets.custom_noop.NoOpCustomDatasets method*), 118
get() (*timeeval.datasets.dataset_manager.DatasetManager method*), 121
get() (*timeeval.datasets.datasets.Datasets method*), 126
get() (*timeeval.datasets.multi_dataset_manager.MultiDatasetManager method*), 133
get_collection_names() (*timeeval.datasets.custom.CustomDatasets method*), 117
get_collection_names() (*timeeval.datasets.custom_base.CustomDatasetsBase method*), 118
get_collection_names() (*timeeval.datasets.custom_noop.NoOpCustomDatasets method*), 118
get_collection_names() (*timeeval.datasets.dataset_manager.DatasetManager method*), 121
get_collection_names() (*timeeval.datasets.datasets.Datasets method*), 127
get_collection_names() (*timeeval.datasets.multi_dataset_manager.MultiDatasetManager method*), 133
get_compute_resource_limits() (*timeeval.ResourceConstraints method*), 38
get_dataset_df() (*timeeval.datasets.dataset_manager.DatasetManager method*), 121
get_dataset_df() (*timeeval.datasets.datasets.Datasets method*), 127
get_dataset_df() (*timeeval.datasets.multi_dataset_manager.MultiDatasetManager method*), 133
get_dataset_names() (*timeeval.datasets.custom.CustomDatasets method*), 117
get_dataset_names() (*timeeval.datasets.custom_base.CustomDatasetsBase method*), 118
get_dataset_names() (*timeeval.datasets.custom_noop.NoOpCustomDatasets method*), 118
get_dataset_names() (*timeeval.datasets.dataset_manager.DatasetManager method*), 122
get_dataset_names() (*timeeval.datasets.datasets.Datasets method*), 127
get_dataset_names() (*timeeval.datasets.multi_dataset_manager.MultiDatasetManager method*), 134
get_dataset_ndarray() (*timeeval.datasets.dataset_manager.DatasetManager method*), 122
get_dataset_ndarray() (*timeeval.datasets.datasets.Datasets method*), 127
get_dataset_ndarray() (*timeeval.datasets.multi_dataset_manager.MultiDatasetManager method*), 134
get_dataset_path() (*timeeval.datasets.dataset_manager.DatasetManager method*), 122
get_dataset_path() (*timeeval.datasets.datasets.Datasets method*), 128
get_dataset_path() (*timeeval.datasets.multi_dataset_manager.MultiDatasetManager method*), 134
get_detailed_metadata() (*timeeval.datasets.dataset_manager.DatasetManager method*), 122
get_detailed_metadata() (*timeeval.datasets.datasets.Datasets method*), 128
get_detailed_metadata() (*timeeval.datasets.multi_dataset_manager.MultiDatasetManager method*), 134

get_execute_timeout() (*timeeval.ResourceConstraints method*), 39
get_finalize_fn() (*timeeval.adapters.base.Adapter method*), 42
get_finalize_fn() (*timeeval.adapters.docker.DockerAdapter method*), 44
get_finalize_fn() (*timeeval.adapters.multivar.MultivarAdapter method*), 46
get_param_names() (*timeeval.heuristics.TimeEvalParameterHeuristic class method*), 138
get_path() (*timeeval.datasets.custom.CustomDatasets method*), 117
get_path() (*timeeval.datasets.custom_base.CustomDatasetsBase method*), 118
get_path() (*timeeval.datasets.custom_noop.NoOpCustomDatasets method*), 118
get_prepare_fn() (*timeeval.adapters.base.Adapter method*), 42
get_prepare_fn() (*timeeval.adapters.docker.DockerAdapter method*), 44
get_prepare_fn() (*timeeval.adapters.multivar.MultivarAdapter method*), 46
get_results() (*timeeval.TimeEval method*), 32
get_stationarity_name() (*timeeval.datasets.metadata.DatasetMetadata method*), 130
get_train_timeout() (*timeeval.ResourceConstraints method*), 39
get_training_type() (*timeeval.datasets.dataset_manager.DatasetManager method*), 123
get_training_type() (*timeeval.datasets.datasets.Datasets method*), 128
get_training_type() (*timeeval.datasets.multi_dataset_manager.MultiDatasetManager method*), 135
grammarviz3() (*in module timeeval.algorithms*), 66
grammarviz3_multi() (*in module timeeval.algorithms*), 67

## H

has_anomalies (*timeeval.datasets.dataset.Dataset property*), 119
hash_dict() (*in module timeeval.utils.hash_dict*), 176
hbos() (*in module timeeval.algorithms*), 68
health_esn() (*in module timeeval.algorithms*), 68
hif() (*in module timeeval.algorithms*), 69
hotsax() (*in module timeeval.algorithms*), 70
HPI_CLUSTER (*class in timeeval.constants*), 40
hybrid_knn() (*in module timeeval.algorithms*), 70

## I

id2labels() (*in module timeeval.utils.label_formatting*), 176
identity() (*timeeval.adapters.function.FunctionAdapter static method*), 45
if_lof() (*in module timeeval.algorithms*), 71
iforest() (*in module timeeval.algorithms*), 72
img_embedding_cae() (*in module timeeval.algorithms*), 73
IndependentParameterGrid (*class in timeeval.params*), 174
index() (*timeeval.datasets.dataset_manager.DatasetRecord method*), 125
INDEX_FILENAME (*timeeval.datasets.dataset_manager.DatasetManager attribute*), 120
INDEX_FILENAME (*timeeval.datasets.datasets.Datasets attribute*), 126
INDEX_FILENAME (*timeeval.datasets.multi_dataset_manager.MultiDatasetManager attribute*), 132
inject_heuristic_values() (*in module timeeval.heuristics*), 137
input_dimensionality (*timeeval.Algorithm attribute*), 35
input_dimensionality (*timeeval.datasets.dataset.Dataset property*), 119
input_type (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125
InputDimensionality (*class in timeeval*), 35
is_train (*timeeval.datasets.metadata.DatasetMetadata attribute*), 130
iter() (*timeeval.params.BayesianParameterSearch method*), 175
iter() (*timeeval.params.FixedParameters method*), 172
iter() (*timeeval.params.ParameterConfig method*), 172

## J

JarAdapter (*class in timeeval.adapters.jar*), 45

## K

kmeans() (*in module timeeval.algorithms*), 74
knn() (*in module timeeval.algorithms*), 75
KUBIC (*timeeval.datasets.metadata.TrendType attribute*), 132
kwargs_overwrites (*timeeval.RemoteConfiguration attribute*), 37

## L

labels2id() (*in module timeeval.utils.label_formatting*), 176

laser_dbn() (*in module timeeval.algorithms*), 76
left_stampi() (*in module timeeval.algorithms*), 76
length (*timeeval.datasets.dataset.Dataset attribute*), 119
length (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125
length (*timeeval.datasets.metadata.DatasetMetadata attribute*), 130
LINEAR (*timeeval.datasets.metadata.TrendType attribute*), 132
load_custom_datasets() (*timeeval.datasets.dataset_manager.DatasetManager method*), 123
load_custom_datasets() (*timeeval.datasets.datasets.Datasets method*), 129
load_custom_datasets() (*timeeval.datasets.multi_dataset_manager.MultiDatasetManager method*), 135
load_dataset() (*in module timeeval.utils.datasets*), 176
load_from_json() (*timeeval.datasets.analyzer.DatasetAnalyzer static method*), 116
load_labels_only() (*in module timeeval.utils.datasets*), 176
load_studies() (*timeeval.integration.optuna.OptunaModule method*), 179
lof() (*in module timeeval.algorithms*), 77
log_level (*timeeval.integration.optuna.OptunaConfiguration attribute*), 181
lstm_ad() (*in module timeeval.algorithms*), 78
lstm_vae() (*in module timeeval.algorithms*), 79

## M

main (*timeeval.Algorithm attribute*), 35
MAX (*timeeval.adapters.multivar.AggregationMethod attribute*), 45
max (*timeeval.datasets.metadata.AnomalyLength attribute*), 130
max_anomaly_length (*timeeval.datasets.dataset.Dataset attribute*), 119
max_anomaly_length (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125
MB (*in module timeeval.resource_constraints*), 40
MEAN (*timeeval.adapters.multivar.AggregationMethod attribute*), 45
mean (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125
mean (*timeeval.datasets.metadata.DatasetMetadata property*), 130
MEAN (*timeeval.utils.window.Method attribute*), 177
means (*timeeval.datasets.metadata.DatasetMetadata attribute*), 130
MEDIAN (*timeeval.adapters.multivar.AggregationMethod attribute*), 45
median (*timeeval.datasets.metadata.AnomalyLength attribute*), 130
MEDIAN (*timeeval.utils.window.Method attribute*), 177
median_anomaly_length (*timeeval.datasets.dataset.Dataset attribute*), 119
median_anomaly_length (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125
median_method() (*in module timeeval.algorithms*), 80
metadata (*timeeval.datasets.analyzer.DatasetAnalyzer property*), 116
METADATA_FILENAME_SUFFIX (*timeeval.datasets.dataset_manager.DatasetManager attribute*), 120
METADATA_FILENAME_SUFFIX (*timeeval.datasets.datasets.Datasets attribute*), 126
METADATA_FILENAME_SUFFIX (*timeeval.datasets.multi_dataset_manager.MultiDatasetManager attribute*), 132
Method (*class in timeeval.utils.window*), 177
Metric (*class in timeeval.metrics*), 145
metric (*timeeval.integration.optuna.OptunaStudyConfiguration attribute*), 182
min (*timeeval.datasets.metadata.AnomalyLength attribute*), 130
min_anomaly_length (*timeeval.datasets.dataset.Dataset attribute*), 119
min_anomaly_length (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125
modelInput (*timeeval.adapters.docker.AlgorithmInterface attribute*), 43
modelOutput (*timeeval.adapters.docker.AlgorithmInterface attribute*), 43
module
    timeeval, 29
    timeeval.adapters.base, 42
    timeeval.adapters.distributed, 42
    timeeval.adapters.docker, 43
    timeeval.adapters.function, 45
    timeeval.adapters.jar, 45
    timeeval.adapters.multivar, 45
    timeeval.constants, 40
    timeeval.datasets.analyzer, 115
    timeeval.datasets.custom, 117
    timeeval.datasets.custom_base, 118
    timeeval.datasets.custom_noop, 118
    timeeval.datasets.dataset, 119
    timeeval.datasets.dataset_manager, 119
    timeeval.datasets.datasets, 126
    timeeval.datasets.metadata, 130

timeeval.datasets.multi_dataset_manager, 132
timeeval.heuristics, 136
timeeval.metrics, 143
timeeval.resource_constraints, 40
timeeval.utils.datasets, 176
timeeval.utils.encode_params, 176
timeeval.utils.hash_dict, 176
timeeval.utils.label_formatting, 176
timeeval.utils.results_path, 177
timeeval.utils.tqdm_joblib, 177
timeeval.utils.window, 177
mscred() (*in module timeeval.algorithms*), 80
mstamp() (*in module timeeval.algorithms*), 81
mtad_gat() (*in module timeeval.algorithms*), 82
multi_hmm() (*in module timeeval.algorithms*), 83
multi_norma() (*in module timeeval.algorithms*), 83
multi_subsequence_lof() (*in module timeeval.algorithms*), 84
MultiDatasetManager (*class in timeeval.datasets.multi_dataset_manager*), 132
MultivarAdapter (*class in timeeval.adapters.multivar*), 45
MULTIVARIATE (*timeeval.InputDimensionality attribute*), 35
MULTIVARIATE_ANOMALY_TEST_CASES (*timeeval.constants.HPI_CLUSTER attribute*), 40
MULTIVARIATE_TEST_CASES (*timeeval.constants.HPI_CLUSTER attribute*), 40
mvalmod() (*in module timeeval.algorithms*), 85

## N

n_trials (*timeeval.integration.optuna.OptunaStudyConfiguration attribute*), 182
name (*timeeval.Algorithm attribute*), 35
name (*timeeval.datasets.dataset.Dataset property*), 119
name (*timeeval.datasets.metadata.Trend property*), 132
name (*timeeval.heuristics.TimeEvalParameterHeuristic property*), 138
name (*timeeval.metrics.AveragePrecision property*), 149
name (*timeeval.metrics.F1Score property*), 151
name (*timeeval.metrics.FScoreAtK property*), 154
name (*timeeval.metrics.Metric property*), 145
name (*timeeval.metrics.PrAUC property*), 147
name (*timeeval.metrics.Precision property*), 149
name (*timeeval.metrics.PrecisionAtK property*), 155
name (*timeeval.metrics.RangeFScore property*), 154
name (*timeeval.metrics.RangePrAUC property*), 156
name (*timeeval.metrics.RangePrecision property*), 152
name (*timeeval.metrics.RangePrecisionRangeRecallAUC property*), 148
name (*timeeval.metrics.RangePrVUS property*), 159
name (*timeeval.metrics.RangeRecall property*), 153
name (*timeeval.metrics.RangeRocAUC property*), 158
name (*timeeval.metrics.RangeRocVUS property*), 160
name (*timeeval.metrics.Recall property*), 150
name (*timeeval.metrics.RocAUC property*), 146
nodes (*timeeval.constants.HPI_CLUSTER attribute*), 41
nodes_ip (*timeeval.constants.HPI_CLUSTER attribute*), 41
NoOpCustomDatasets (*class in timeeval.datasets.custom_noop*), 118
norma() (*in module timeeval.algorithms*), 86
normalizing_flows() (*in module timeeval.algorithms*), 87
NOT_STATIONARY (*timeeval.datasets.metadata.Stationarity attribute*), 131
NoThresholding (*class in timeeval.metrics.thresholding*), 163
novelty_svr() (*in module timeeval.algorithms*), 88
num_anomalies (*timeeval.datasets.dataset.Dataset attribute*), 119
num_anomalies (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125
num_anomalies (*timeeval.datasets.metadata.DatasetMetadata attribute*), 131
numenta_htm() (*in module timeeval.algorithms*), 89

## O

object_hook() (*timeeval.datasets.metadata.DatasetMetadataEncoder static method*), 131
ocean_wnn() (*in module timeeval.algorithms*), 91
odin01 (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin01_ip (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin02 (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin02_ip (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin03 (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin03_ip (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin04 (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin04_ip (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin05 (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin05_ip (*timeeval.constants.HPI_CLUSTER attribute*), 41

odin06 (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin06_ip (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin07 (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin07_ip (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin08 (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin08_ip (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin09 (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin09_ip (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin10 (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin10_ip (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin11 (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin11_ip (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin12 (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin12_ip (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin13 (*timeeval.constants.HPI_CLUSTER attribute*), 41
odin13_ip (*timeeval.constants.HPI_CLUSTER attribute*), 42
odin14 (*timeeval.constants.HPI_CLUSTER attribute*), 42
odin14_ip (*timeeval.constants.HPI_CLUSTER attribute*), 42
OK (*timeeval.Status attribute*), 34
omnianomaly() (*in module timeeval.algorithms*), 92
OOM (*timeeval.Status attribute*), 34
OptunaConfiguration (*class in timeeval.integration.optuna*), 180
OptunaModule (*class in timeeval.integration.optuna*), 179
OptunaStudyConfiguration (*class in timeeval.integration.optuna*), 181
order (*timeeval.datasets.metadata.Trend property*), 132

## P

padding_borders() (*in module timeeval.utils.window*), 178
param_config (*timeeval.Algorithm attribute*), 35
param_grid (*timeeval.params.FullParameterGrid property*), 173
param_grid (*timeeval.params.IndependentParameterGrid property*), 174
param_schema (*timeeval.Algorithm attribute*), 35
ParameterConfig (*class in timeeval.params*), 172
ParameterDependenceHeuristic (*class in timeeval.heuristics*), 142
parameters() (*timeeval.heuristics.TimeEvalParameterHeuristic method*), 138
pcc() (*in module timeeval.algorithms*), 93
pci() (*in module timeeval.algorithms*), 94
PercentileThresholding (*class in timeeval.metrics.thresholding*), 164
period_size (*timeeval.datasets.dataset.Dataset attribute*), 119
period_size (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125
PeriodSizeHeuristic (*class in timeeval.heuristics*), 142
phasespace_svm() (*in module timeeval.algorithms*), 95
post_run() (*timeeval.integration.TimeEvalModule method*), 182
postprocess (*timeeval.Algorithm attribute*), 35
PR_AUC (*timeeval.metrics.DefaultMetrics attribute*), 161
PrAUC (*class in timeeval.metrics*), 147
pre_run() (*timeeval.integration.TimeEvalModule method*), 183
Precision (*class in timeeval.metrics*), 149
PrecisionAtK (*class in timeeval.metrics*), 155
prepare() (*timeeval.integration.optuna.OptunaModule method*), 179
prepare() (*timeeval.integration.TimeEvalModule method*), 183
preprocess (*timeeval.Algorithm attribute*), 35
pruner (*timeeval.integration.optuna.OptunaStudyConfiguration attribute*), 182
pst() (*in module timeeval.algorithms*), 96
PyThreshThresholding (*class in timeeval.metrics.thresholding*), 170

## Q

QUADRATIC (*timeeval.datasets.metadata.TrendType attribute*), 132

## R

random_black_forest() (*in module timeeval.algorithms*), 97
RANGE_F1 (*timeeval.metrics.DefaultMetrics attribute*), 161
RANGE_PR_AUC (*timeeval.metrics.DefaultMetrics attribute*), 161
RANGE_PRECISION (*timeeval.metrics.DefaultMetrics attribute*), 161
RANGE_RECALL (*timeeval.metrics.DefaultMetrics attribute*), 161

RangeFScore (*class in timeeval.metrics*), 153
RangePrAUC (*class in timeeval.metrics*), 156
RangePrecision (*class in timeeval.metrics*), 151
RangePrecisionRangeRecallAUC (*class in timeeval.metrics*), 147
RangePrVUS (*class in timeeval.metrics*), 158
RangeRecall (*class in timeeval.metrics*), 152
RangeRocAUC (*class in timeeval.metrics*), 157
RangeRocVUS (*class in timeeval.metrics*), 159
Recall (*class in timeeval.metrics*), 150
refresh() (*timeeval.datasets.dataset_manager.DatasetManager method*), 123
refresh() (*timeeval.datasets.datasets.Datasets method*), 129
refresh() (*timeeval.datasets.multi_dataset_manager.MultiDatasetManager method*), 135
RelativeDatasetSizeHeuristic (*class in timeeval.heuristics*), 143
remote_python (*timeeval.RemoteConfiguration attribute*), 37
RemoteConfiguration (*class in timeeval*), 36
remove_managed_containers (*timeeval.integration.optuna.OptunaConfiguration attribute*), 181
ResourceConstraints (*class in timeeval*), 37
RESULT_KEYS (*timeeval.TimeEval attribute*), 32
ReverseWindowing (*class in timeeval.utils.window*), 177
robust_pca() (*in module timeeval.algorithms*), 98
ROC_AUC (*timeeval.metrics.DefaultMetrics attribute*), 161
RocAUC (*class in timeeval.metrics*), 146
rsync_results() (*timeeval.TimeEval method*), 33
rsync_results_from() (*timeeval.TimeEval static method*), 33
run() (*timeeval.TimeEval method*), 33

## S

s_h_esd() (*in module timeeval.algorithms*), 98
sampler (*timeeval.integration.optuna.OptunaStudyConfiguration attribute*), 182
sand() (*in module timeeval.algorithms*), 99
sarima() (*in module timeeval.algorithms*), 100
save() (*timeeval.datasets.dataset_manager.DatasetManager method*), 123
save_results() (*timeeval.TimeEval method*), 33
save_to_json() (*timeeval.datasets.analyzer.DatasetAnalyzer method*), 117
scheduler_host (*timeeval.RemoteConfiguration attribute*), 37
scheduler_port (*timeeval.RemoteConfiguration attribute*), 37
score() (*timeeval.metrics.AveragePrecision method*), 149
score() (*timeeval.metrics.F1Score method*), 151
score() (*timeeval.metrics.FScoreAtK method*), 154
score() (*timeeval.metrics.Metric method*), 145
score() (*timeeval.metrics.PrAUC method*), 147
score() (*timeeval.metrics.Precision method*), 149
score() (*timeeval.metrics.PrecisionAtK method*), 155
score() (*timeeval.metrics.RangeFScore method*), 154
score() (*timeeval.metrics.RangePrAUC method*), 156
score() (*timeeval.metrics.RangePrecision method*), 152
score() (*timeeval.metrics.RangePrecisionRangeRecallAUC method*), 148
score() (*timeeval.metrics.RangePrVUS method*), 159
score() (*timeeval.metrics.RangeRecall method*), 153
score() (*timeeval.metrics.RangeRocAUC method*), 158
score() (*timeeval.metrics.RangeRocVUS method*), 160
score() (*timeeval.metrics.Recall method*), 150
score() (*timeeval.metrics.RocAUC method*), 146
select() (*timeeval.datasets.custom.CustomDatasets method*), 117
select() (*timeeval.datasets.custom_base.CustomDatasetsBase method*), 118
select() (*timeeval.datasets.custom_noop.NoOpCustomDatasets method*), 118
select() (*timeeval.datasets.dataset_manager.DatasetManager method*), 123
select() (*timeeval.datasets.datasets.Datasets method*), 129
select() (*timeeval.datasets.multi_dataset_manager.MultiDatasetManager method*), 136
SEMI_SUPERVISED (*timeeval.TrainingType attribute*), 36
series2graph() (*in module timeeval.algorithms*), 101
shape (*timeeval.datasets.metadata.DatasetMetadata property*), 131
SigmaThresholding (*class in timeeval.metrics.thresholding*), 168
split_at (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125
sr() (*in module timeeval.algorithms*), 102
sr_cnn() (*in module timeeval.algorithms*), 102
ssa() (*in module timeeval.algorithms*), 103
stamp() (*in module timeeval.algorithms*), 104
stationarities (*timeeval.datasets.metadata.DatasetMetadata attribute*), 131
Stationarity (*class in timeeval.datasets.metadata*), 131
stationarity (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125
stationarity (*timeeval.datasets.metadata.DatasetMetadata property*), 131
STATIONARY (*timeeval.datasets.metadata.Stationarity attribute*), 131
Status (*class in timeeval*), 34
stddev (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125

stddev (*timeeval.datasets.metadata.DatasetMetadata property*), 131
stddevs (*timeeval.datasets.metadata.DatasetMetadata attribute*), 131
stomp() (*in module timeeval.algorithms*), 105
storage (*timeeval.integration.optuna.OptunaStudyConfiguration attribute*), 182
subsequence_fast_mcd() (*in module timeeval.algorithms*), 106
subsequence_if() (*in module timeeval.algorithms*), 106
subsequence_knn() (*in module timeeval.algorithms*), 107
subsequence_lof() (*in module timeeval.algorithms*), 108
SUM (*timeeval.utils.window.Method attribute*), 177
SUM_BEFORE (*timeeval.adapters.multivar.AggregationMethod attribute*), 45
SUPERVISED (*timeeval.TrainingType attribute*), 36
supports_continuous_scorings() (*timeeval.metrics.AveragePrecision method*), 149
supports_continuous_scorings() (*timeeval.metrics.F1Score method*), 151
supports_continuous_scorings() (*timeeval.metrics.FScoreAtK method*), 155
supports_continuous_scorings() (*timeeval.metrics.Metric method*), 146
supports_continuous_scorings() (*timeeval.metrics.PrAUC method*), 147
supports_continuous_scorings() (*timeeval.metrics.Precision method*), 150
supports_continuous_scorings() (*timeeval.metrics.PrecisionAtK method*), 155
supports_continuous_scorings() (*timeeval.metrics.RangeFScore method*), 154
supports_continuous_scorings() (*timeeval.metrics.RangePrAUC method*), 157
supports_continuous_scorings() (*timeeval.metrics.RangePrecision method*), 152
supports_continuous_scorings() (*timeeval.metrics.RangePrecisionRangeRecallAUC method*), 148
supports_continuous_scorings() (*timeeval.metrics.RangePrVUS method*), 159
supports_continuous_scorings() (*timeeval.metrics.RangeRecall method*), 153
supports_continuous_scorings() (*timeeval.metrics.RangeRocAUC method*), 158
supports_continuous_scorings() (*timeeval.metrics.RangeRocVUS method*), 160
supports_continuous_scorings() (*timeeval.metrics.Recall method*), 150
supports_continuous_scorings() (*timeeval.metrics.RocAUC method*), 146

## T

tanogan() (*in module timeeval.algorithms*), 109
tarzan() (*in module timeeval.algorithms*), 110
task_cpu_limit (*timeeval.ResourceConstraints attribute*), 39
task_memory_limit (*timeeval.ResourceConstraints attribute*), 39
tasks_per_host (*timeeval.ResourceConstraints attribute*), 39
telemanom() (*in module timeeval.algorithms*), 111
test_path (*timeeval.datasets.custom.CDEntry attribute*), 117
test_path (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125
ThresholdingStrategy (*class in timeeval.metrics.thresholding*), 161
timeeval
    module, 29
TimeEval (*class in timeeval*), 29
timeeval.adapters.base
    module, 42
timeeval.adapters.distributed
    module, 42
timeeval.adapters.docker
    module, 43
timeeval.adapters.function
    module, 45
timeeval.adapters.jar
    module, 45
timeeval.adapters.multivar
    module, 45
timeeval.constants
    module, 40
timeeval.datasets.analyzer
    module, 115
timeeval.datasets.custom
    module, 117
timeeval.datasets.custom_base
    module, 118
timeeval.datasets.custom_noop
    module, 118
timeeval.datasets.dataset
    module, 119
timeeval.datasets.dataset_manager
    module, 119
timeeval.datasets.datasets
    module, 126
timeeval.datasets.metadata
    module, 130
timeeval.datasets.multi_dataset_manager
    module, 132
timeeval.heuristics
    module, 136
timeeval.metrics

module, 143
timeeval.resource_constraints
module, 40
timeeval.utils.datasets
module, 176
timeeval.utils.encode_params
module, 176
timeeval.utils.hash_dict
module, 176
timeeval.utils.label_formatting
module, 176
timeeval.utils.results_path
module, 177
timeeval.utils.tqdm_joblib
module, 177
timeeval.utils.window
module, 177
TimeEvalHeuristic() (*in module timeeval.heuristics*), 136
TimeEvalModule (*class in timeeval.integration*), 182
TimeEvalParameterHeuristic (*class in timeeval.heuristics*), 138
TIMEOUT (*timeeval.Status attribute*), 34
to_json() (*timeeval.datasets.metadata.DatasetMetadata method*), 131
to_json_string() (*timeeval.adapters.docker.AlgorithmInterface method*), 43
TopKPointsThresholding (*class in timeeval.metrics.thresholding*), 166
TopKRangesThresholding (*class in timeeval.metrics.thresholding*), 167
torsk() (*in module timeeval.algorithms*), 112
tpe (*timeeval.datasets.metadata.Trend attribute*), 132
tqdm_joblib() (*in module timeeval.utils.tqdm_joblib*), 177
TRAIN (*timeeval.data_types.ExecutionType attribute*), 42
train_is_normal (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125
train_path (*timeeval.datasets.custom.CDEntry attribute*), 117
train_path (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 125
train_timeout (*timeeval.ResourceConstraints attribute*), 39
train_type (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 126
training_type (*timeeval.Algorithm attribute*), 35
training_type (*timeeval.datasets.dataset.Dataset attribute*), 119
TrainingType (*class in timeeval*), 36
transform() (*timeeval.metrics.thresholding.FixedValueThresholding method*), 164
transform() (*timeeval.metrics.thresholding.NoThresholding method*), 163
transform() (*timeeval.metrics.thresholding.PercentileThresholding method*), 165
transform() (*timeeval.metrics.thresholding.PyThreshThresholding method*), 171
transform() (*timeeval.metrics.thresholding.SigmaThresholding method*), 169
transform() (*timeeval.metrics.thresholding.ThresholdingStrategy method*), 162
transform() (*timeeval.metrics.thresholding.TopKPointsThresholding method*), 167
transform() (*timeeval.metrics.thresholding.TopKRangesThresholding method*), 168
Trend (*class in timeeval.datasets.metadata*), 131
trend (*timeeval.datasets.dataset_manager.DatasetRecord attribute*), 126
trend (*timeeval.datasets.metadata.DatasetMetadata property*), 131
TREND_STATIONARY (*timeeval.datasets.metadata.Stationarity attribute*), 131
trends (*timeeval.datasets.metadata.DatasetMetadata attribute*), 131
TrendType (*class in timeeval.datasets.metadata*), 132
triple_es() (*in module timeeval.algorithms*), 113
ts_bitmap() (*in module timeeval.algorithms*), 114

## U

UNIVARIATE (*timeeval.InputDimensionality attribute*), 35
UNIVARIATE_ANOMALY_TEST_CASES (*timeeval.constants.HPI_CLUSTER attribute*), 40
UNSUPERVISED (*timeeval.TrainingType attribute*), 36
update_config() (*timeeval.params.BayesianParameterSearch method*), 176
update_unset_options() (*timeeval.integration.optuna.OptunaStudyConfiguration method*), 182
use_default_logging (*timeeval.integration.optuna.OptunaConfiguration attribute*), 181
use_preliminary_model_on_train_timeout (*timeeval.ResourceConstraints attribute*), 40
use_preliminary_scores_on_execute_timeout (*timeeval.ResourceConstraints attribute*), 40

## V

valmod() (*in module timeeval.algorithms*), 115
VARIABLE_LENGTH_TEST_CASES (*timeeval.constants.HPI_CLUSTER attribute*), 40

# W

worker_hosts (*timeeval.RemoteConfiguration attribute*), 37